

Built-in ACS functions

These are all the functions that ACS has built-in. Note that these can only be used in scripts and not on linedefs or assigned to things. In those contexts, only Action Specials may be used, though action specials may also be used in scripts as if they were ACS functions.

When adding to or updating the definition of these functions, please update the Doom Builder ACS configuration as well. Thank you.

Script control

Break
Continue
Restart
Suspend
Terminate
Named scripts
ACS_NamedExecute
ACS_NamedSuspend
ACS_NamedTerminate
ACS_NamedLockedExecute
ACS_NamedLockedExecuteDoor
ACS_NamedExecuteWithResult
ACS_NamedExecuteAlways
Scripting
ScriptCall
Waiting
ACS_ExecuteWait
ACS_NamedExecuteWait
Delay
NamedScriptWait
PolyWait
ScriptWait
TagWait
Math
Ceil
cos
FixedDiv
FixedMul
FixedSqrt
Floor
Random
Round
sin
Sqrt
StrLen
VectorAngle
VectorLength
Information
ActivatorTID
CanRaiseActor
CheckActorCeilingTexture
CheckActorClass

CheckActorFloorTexture
CheckActorProperty
CheckActorState
CheckClass
CheckFlag
CheckFont
CheckPlayerCamera
CheckProximity
CheckSight
ClassifyActor
GameSkill
GameType
GetActorAngle
GetActorCeilingZ
GetActorClass
GetActorFloorTerrain
GetActorFloorTexture
GetActorFloorZ
GetActorLightLevel
GetActorPitch
GetActorPowerupTics
GetActorProperty
GetActorRoll
GetActorVelX
GetActorVelY
GetActorVelZ
GetActorViewHeight
GetActorX
GetActorY
GetActorZ
GetAirSupply
GetAmmoCapacity
GetArmorInfo
GetArmorType
GetChar
GetCVar
GetCVarString
GetLevelInfo
GetLineActivation
GetLineRowOffset
GetLineX
GetLineY
GetPlayerInfo
GetPlayerInput
GetPolyobjX
GetPolyobjY
GetScreenHeight
GetScreenWidth
GetSectorCeilingZ
GetSectorFloorZ
GetSectorLightLevel
GetSigilPieces
GetUserArray
GetUserCVar

GetUserCVarString
GetUserVariable
GetWeapon
IsPointerEqual
IsTIDUsed
LineSide
PlayerClass
PlayerCount
PlayerFrag
PlayerInGame
PlayerIsBot
PlayerNumber
SetResultValue
StrArg
StrCmp
StrIcmp
ThingCount
ThingCountName
ThingCountNameSector
ThingCountSector
Timer
UniqueTID
UDMF
GetLineUDMFInt
GetLineUDMFFixed
GetSectorUDMFInt
GetSectorUDMFFixed
GetSideUDMFInt
GetSideUDMFFixed
GetThingUDMFInt
GetThingUDMFFixed
Sounds
ActivatorSound
AmbientSound
LocalAmbientSound
LocalSetMusic
PlayActorSound
PlaySound
SectorSound
SetMusic
SetMusicVolume
SoundSequence
SoundSequenceOnActor
SoundSequenceOnSector
SoundSequenceOnPolyobj
SoundVolume
StopSound
ThingSound
Inventory
CheckActorInventory
CheckInventory
CheckWeapon
ClearActorInventory
ClearInventory

DropInventory
DropItem
GetMaxInventory
GiveActorInventory
GiveInventory
SetWeapon
TakeActorInventory
TakeInventory
UseActorInventory
UseInventory
Display
HudMessage
HudMessageBold
Log
Print
PrintBold
SetFont
SetHudClipRect
SetHudSize
SetHudWrapWidth
SetMugShotState
StrLeft
StrMid
StrParam
StrRight
StrCpy
Level alteration
ChangeCeiling
ChangeFloor
ChangeLevel
ChangeSky
ClearLineSpecial
QuakeEx
Radius_Quake2
ReplaceTextures
SectorDamage
SetAirControl
SetCameraToTexture
SetCeilingTrigger
SetCVar
SetCVarString
SetFloorTrigger
SetFogDensity
SetGravity
SetLineActivation
SetLineBlocking
SetLineMonsterBlocking
SetLineSpecial
SetLineTexture
SetSectorDamage
SetSectorGlow
SetSectorTerrain
SetSkyScrollSpeed
SetUserCVar

SetUserCVarString
SpawnParticle
Actor control
CancelFade
ChangeActorAngle
ChangeActorPitch
ChangeActorRoll
CreateTranslation
DamageActor
FadeRange
FadeTo
LineAttack
MorphActor
PickActor
SetActivator
SetActivatorToTarget
SetActorAngle
SetActorFlag
SetActorPitch
SetActorPosition
SetActorProperty
SetActorRoll
SetActorState
SetActorTeleFog
SetActorVelocity
SetAirSupply
SetAmmoCapacity
SetMarineSprite
SetMarineWeapon
SetPointer
SetSubtitleNumber
SetThingSpecial
SetTranslation
SetUserArray
SetUserVariable
Spawn
SpawnDecal
SpawnForced
SpawnProjectile
SpawnSpot
SpawnSpotFacing
SpawnSpotFacingForced
SpawnSpotForced
SwapActorTeleFog
Thing_Damage2
Thing_Projectile2
UnMorphActor
Warp

CancelFade

```
void CancelFade (void);
```

Usage

If either FadeTo or FadeRange is currently in progress, CancelFade stops it and turns off the palette flash. If neither of these in progress, it does nothing.

Examples

This alarm flashes the screen red and makes an alarm sound.

```
script 1 (void)
{
    AmbientSound("special/alarm",127);
    FadeTo(255, 0, 0, 0.9, 1.0);
    Delay(1*35);
    FadeTo(255, 0, 0, 0.9, 1.0);
    Delay(1*35);
}
```

The only way to shut it of is to hit a switch which executes script 2.

```
script 2 (void)
{
    ACS_Terminate(1,0);
    CancelFade();
}
```

ChangeActorAngle

`void ChangeActorAngle (int tid, fixed angle [, bool interpolate])`

Usage

Sets the angle for the actors with the specified tid. If tid is 0, it sets the angle for the activator of the script.

This function duplicates and extends on `SetActorAngle` by having the added option to interpolate the view.

Parameters

tid: the tid of the actor.

angle: the angle to set. This is a fixed point angle in the range of 0.0 to 1.0.

interpolate: whether to interpolate the view or not. If true, the view is interpolated, otherwise it is not. Default is false, i.e. no interpolation.

This works to set the facing of monsters, players, or any other actor.

North = 0.25

West = 0.5

South = 0.75

East = 1.0

Examples

This is a similar script to the one on the `SetActorAngle` page. The difference in this is that it takes advantage of the added interpolation option, thus making the spinning smoother.

Script 1 (int spintime)

```
{
    While(spintime-- > 0)
    {
        ChangeActorAngle(100, GetActorAngle(100) - 0.02, TRUE);
        Delay(1);
        Print(s:"You spin me right round, baby right round like a record, baby right round, round, round");
    }
}
```

ChangeActorPitch

`void ChangeActorPitch (int tid, fixed pitch [, bool interpolate])`

Usage

Sets the pitch for the actors with the specified tid. If tid is 0, it sets the pitch for the activator of the script.

This function duplicates and extends on SetActorPitch by having the added option to interpolate the view.

Parameters

tid: the tid of the actor.

pitch: the pitch to set. This is a fixed point angle. See GetActorPitch for the possible range.

interpolate: whether to interpolate the view or not. If true, the view is interpolated, otherwise it is not. Default is false, i.e. no interpolation.

ChangeActorRoll

Warning: This feature is GZDoom specific, and is not compatible with ZDoom!
To see all of GZDoom's specific features, see [GZDoom features](#).

`void ChangeActorRoll (int tid, fixed angle [, bool interpolate])`

Usage

Sets the roll angle for the actors with the specified tid. If tid is 0, it sets the roll angle for the activator of the script. Note that the function has a visual effect on models and player's view (when called for players) only. It has no effect on sprites.

This function duplicates and extends on `SetActorRoll` by having the added option to interpolate the view.

Parameters

tid: the tid of the actor.

angle: the roll angle to set. This is a fixed point angle in the range of 0.0 to 1.0, with:

0.25 → right

0.5 → down

0.75 → left

1.0 → up

interpolate: whether to interpolate the view or not. If true, the view is interpolated, otherwise it is not. Default is false, i.e. no interpolation.

CreateTranslation

```
void CreateTranslation(int transnumber, a:b=c:d, ...);  
void CreateTranslation(int transnumber, a:b=[red1,green1,blue1]:[red2,green2,blue2], ...);
```

CreateTranslation creates a new palette translation, and Thing_SetTranslation causes a thing to use the palette translation you have created. A palette translation is what Doom uses to create the different colored players in multiplayer games. Translations are not remembered across maps, so each map that uses a translation needs to create that translation in a script. You can define up to 65535 different palette translations for each map.

Creating a translation

There are three basic forms for the CreateTranslation command. In its simplest form, you just pass CreateTranslation a translation number, and it resets it so that no translation occurs. This form looks like CreateTranslation(x); where x is some number between 1 and 65536.

In its other two forms, CreateTranslation can change the translation to actually recolor things. After the translation number, you include the range of palette entries to change followed by the colors to change them to. The colors they get changed to can be either palette entries or RGB values. To remap using palette entries, use CreateTranslation(x, a:b=c:d);. This replaces palette entries aâ€œb with palette entries câ€œd. To remap using RGB values, use CreateTranslation(x, a:b=[red1,green1,blue1]:[red2,green2,blue2]);. This will replace palette entries aâ€œb with colors starting at [red1,green1,blue1] and spreading to [red2,green2,blue2]. You can also remap more than one range in a translation by separating them with commas.

Modders can also now define saturated translations with CreateTranslation(x,a:b=%[red1,green1,blue1]:[red2,green2,blue2]);. The minimum float values for this type of translation are 0.0, while the maximum is 2.0.

Examples

Here are some examples using the standard green range of the Doom palette (indices 112 to 127):

```
// This is the standard gray/indigo range  
CreateTranslation (1, 112:127=96:111);  
// This is a white range  
CreateTranslation (2, 112:127=80:95);  
// This is the standard brown range  
CreateTranslation (3, 112:127=64:79);  
// This is a peach range  
CreateTranslation (4, 112:127=48:63);  
// This is the standard red range  
CreateTranslation (5, 112:127=32:47);  
// This is a pink range  
CreateTranslation (6, 112:127=16:31);  
// This is a range from yellow (255,255,0) to black (0,0,0)  
CreateTranslation (14, 112:127=[255,255,0]:[0,0,0]);  
// This is two ranges in one translation.  
// The first range creates an ugly suit colored from green to red.  
// The second makes his skin reddish.  
CreateTranslation (15, 112:127=[0,255,0]:[255,0,0], 79:48=47:16);  
// This rather lengthy example almost completely recolors the Cyberdemon.  
// The first three ranges change the Cyberdemon's skin to red.  
// The next two ranges make his abdomen blue.  
// Finally, the last two ranges make his muzzle flash blue as well.  
CreateTranslation (1, 64:79=32:47, 48:63=47:32, 144:151=32:47, 32:47=192:207,  
168:191=192:207, 224:231=[255, 255, 255]:[128, 128, 255], 208:223=192:207);
```

How it's Done

Assigning these translations to things is done with `Thing_SetTranslation`. It can also be done in DECORATE with the `Translation` tag.

For reference, see [here](#) for numbered palettes for each supported game.

DamageActor

int DamageActor (int targettid, int targetptr, int inflictortid, int inflictorptr, int damage, str damagetype);

Usage

Damages a single actor with a specific damage amount and damage type. You can optionally specify another actor as having done the damage, affecting the obituary and targeting of the target actor.

Parameters

targettid: the tid of the actor to damage. You may specify 0 to refer to the activator.

targetptr: the actor pointer to use on the target actor.

inflictortid: the tid of the actor inflicting the damage. You may specify 0 to refer to the activator.

inflictorptr: the actor pointer to use on the inflicting actor. AAPTR_NULL will make the damage originate from the world.

damage: the amount of damage the target actor will receive. Negative values are clamped to zero (you cannot heal with this function).

damagetype: the damagetype to use. See Damage types.

Return value

The function returns the amount of damage the target actor actually received, after all the factors have been processed (such as skill level and damage factors). If the damaging was cancelled (e.g. by invulnerability), or if no target actor was found, -1 is returned.

FadeRange

`void FadeRange (int red1, int green1, int blue1, fixed amount1, int red2, int green2, int blue2, fixed amount2, fixed seconds);`

Usage

FadeRange sets the current flash to the first color set and then fades it to the second color set over the specified number of seconds.

This is more or less the ACS equivalent of `A_SetBlend`.

If the function is run with no activator (for example an OPEN script or removing an existing activator with `SetActivator`), it will run for and affect all active players in the game.

Parameters

`red1`: Amount of red in the initial flash from 0-255.

`green1`: Amount of green in the initial flash from 0-255.

`blue1`: Amount of blue in the initial flash from 0-255.

`amount1`: Intensity of the initial flash from 0-1.0.

`red2`: Amount of red in the final flash from 0-255.

`green2`: Amount of green in the final flash from 0-255.

`blue2`: Amount of blue in the final flash from 0-255.

`amount2`: Intensity of the final flash from 0-1.0.

`seconds`: Time needed to complete the fade in seconds.

Examples

script 1 (void)

```
{  
    // fade from 80% green to 80% blue in two seconds  
    FadeRange (0, 255, 0, 0.8, 0, 0, 255, 0.8, 2.0);  
}
```

FadeTo

`void FadeTo (int red, int green, int blue, fixed amount, fixed seconds);`

Usage

Fades the activator's view from the current palette flash to another one. This will also work if the activator is looking through another viewpoint. (Using a camera, etc)

If the function is run with no activator (for example an OPEN script or removing an existing activator with SetActivator), it will run for and affect all active players in the game.

Parameters

red: Amount of red in the flash from 0-255.

green: Amount of green in the flash from 0-255.

blue: Amount of blue in the flash from 0-255.

amount: Intensity of the flash from 0.0-1.0. Note this **REQUIRES** a decimal, or it will not work!

seconds: Time needed to complete the fade in seconds. This requires a decimal as well, or the transition will be instant.

Examples

script 100 ENTER

```
{
  //Fade to full intensity red in two seconds
  FadeTo (255, 0, 0, 1.0, 2.0);
  Delay(35 * 2);

  //Fade to half intensity black in two seconds, ...
  FadeTo (0, 0, 0, 0.5, 2.0);
}
```

LineAttack (ACS)

```
void LineAttack (int tid, fixed angle, fixed pitch, int damage [, str pufftype [, str damagetype [, fixed range [, int flags [, int pufftid]]]]]);
```

Usage

Fires a hitscan attack. If tid is 0, the activator of the script is the source of the attack.

Parameters

tid: The tid of the actor to fire the hitscan attack.

angle: The angle at which the attack should be fired.

pitch: The pitch by which the attack should be fired.

damage: The damage dealt by the attack.

pufftype: The puff to spawn. Default is "BulletPuff".

damagetype: The damage type of the attack. Default is "None".

range: The maximum distance at which the attack successfully hits. Default is 2048.0.

flags: The following flags can be combined by using | between the constant names:

FHF_NORANDOMPUFFZ " Disables the random z offset given to the puff when spawned.

FHF_NOIMPACTDECAL " Disables the generation of decals as a result of the attack.

pufftid: The tid to assign to the spawned puff. Default is 0, for no assignment.

MorphActor

`int MorphActor (int tid [, str playerclass [, str monsterclass [, int duration [, int style [, str morphflash [, str unmorphflash]]]]]])`

Usage

This function and its complement `UnMorphActor` give the ACS coder direct access to the engine's morph subsystem, instead of having to strategically place `DECORATE` items and suchlike.

Parameters

`tid`: The actor(s) to morph. The activator is used if this parameter is zero.

`playerclass`: Defines what class to morph a player into.

`monsterclass`: Defines what class to morph a monster into.

`duration`: Defines the duration of the morphing effects.

`style`: Defines the behaviour of the morphing effects.

`morphflash`: Defines the effect flash actor to spawn when the player morphs. If omitted, the game's default teleport fog is used.

`unmorphflash`: Defines the effect flash actor to spawn when the player unmorphs. If omitted, the game's default teleport fog is used.

This function does honor the `MRF_WHENINVULNERABLE` flag when used on a player, provided that player is also the activator of the function.

Except for `tid`, the parameters are the same special properties defined by the `MorphProjectile` class. They are summarised above; for full details and additional notes, please refer to the `MorphProjectile` class.

Note: for versions of ACC older than 1.58, all optional arguments must be treated as mandatory. Specify 0 for unused integer arguments and "" for unused string arguments.

Return value

The return value is the number of actors successfully morphed. This also means that for `TID = 0`, it is also a boolean (0 = failed, 1 = succeeded).

Examples

The following example assumes that a cyberdemon with the `DONTMORPH` actor flag disabled has a `TID` of one. The function call turns it into a pitiful little puppy demon, so you can kill him with ease!

Note that the Demon's `DECORATE` code is reproduced from `gzdoom.pk3` because the engine currently requires that all morphed monsters must inherit the `MorphedMonster` class, and multiple inheritance is not supported; at the moment, you cannot arbitrarily morph one kind of monster into another.

script 1 (void)

```
{
    MorphActor(1, "", "MorphDemon", 1048576, 0, "", "");
}
actor MorphableCyberDemon : Cyberdemon replaces Cyberdemon
{
    -DONTMORPH // The cyberdemon has this flag set, by default. The flag needs to be cleared so the monster can morph.
}
```

actor MorphDemon : MorphedMonster

```
{
    Game Doom
```



```
Health 150
PainChance 180
Speed 10
Radius 30
Height 56
Mass 400
Monster
+FLOORCLIP
SeeSound "demon/sight"
AttackSound "demon/melee"
PainSound "demon/pain"
DeathSound "demon/death"
ActiveSound "demon/active"
Obituary "%o was killed by a dem... cyberd... well, by something anyway"
States
{
Spawn:
    SARG AB 10 A_Look
    Loop
See:
    SARG AABBBCCDD 2 Fast A_Chase
    Loop
Melee:
    SARG EF 8 Fast A_FaceTarget
    SARG G 8 Fast A_SargAttack
    Goto See
Pain:
    SARG H 2 Fast
    SARG H 2 Fast A_Pain
    Goto See
Death:
    SARG I 8
    SARG J 8 A_Scream
    SARG K 4
    SARG L 4 A_NoBlocking
    SARG M 4
    SARG N -1
    Stop
Raise:
    SARG N 5
    SARG MLKJI 5
    Goto See
}
}
```

PickActor

int PickActor (int source, fixed angle, fixed pitch, fixed distance, int tid [, int actorMask [, int wallMask [, int flags]]])

Usage

Picks a single actor that is being aimed at, and assigns a tid to it.

This function is tricky to use correctly; calling it in the straightforward way can cause bugs. See below for a full example.

Note: Actors with the GHOST flag are ignored by the ray.

Parameters

source: the tid of the actor that the pick originates from. Use a tid of 0 to use the script's activator as the source.

angle: the angle to look at. This is a fixed point angle in the range of 0.0 to 1.0.

pitch: the pitch to look at. This is a fixed point angle in the range of 0.0 to 1.0.

distance: the maximum distance to pick actors in.

tid: the tid to assign to an actor that is being looked at.

actorMask: flags that an actor must have in order for it to be picked. The default is MF_SHOOTABLE. See Actor flags for a list of possible flags.

wallMask: linedef flags that will block the pick ray. The default is ML_BLOCKEVERYTHING | ML_BLOCKHITSCAN. See Linedef flags for a list of possible flags.

flags: The following flags can be combined with the | character between the constant names:

PICKAF_FORCETID " The picked actor is forcibly assigned the specified tid, otherwise it will only be assigned the tid if it does not have one to begin with.

PICKAF_RETURNTID " Instead of using a return value of 1, PickActor will use the actor's tid as the return value.

Return value

The function returns true if an actor was picked, false if not. If PICKAF_RETURNTID is specified, the function returns the picked actor's tid.

Actor flags

MF_SPECIAL: The actor can be picked up.

MF_SOLID: Set when the actor should be solid (blocking).

MF_SHOOTABLE: The actor can be damaged. If it's health goes below 0 it enters its death state.

MF_NOSECTOR: Object is not linked into the sector. This makes it invisible and excludes it from certain physics checks.

MF_NOBLOCKMAP: The actor is added to the internal blockmap for collision detection purposes.

MF_AMBUSH: Monster is set to ambush players.

MF_JUSTHIT: Try to attack right back (used in monster AI).

MF_JUSTATTACKED: Take at least one step before attacking (used in monster AI).

MF_SPAWNCEILING: Actor is spawned hanging from the ceiling as opposed to standing on the floor.

MF_NOGRAVITY: Actor is not subject to gravity.

MF_DROPOFF: Actor can walk over ledges/taller steps.

MF_PICKUP: This actor can pick up items.

MF_NOCLIP: Actor is totally excluded from collision detection and can walk through walls etc.

MF_INCHASE: Used internally to avoid recursion.

MF_FLOAT: Floating actor that can change height at will.

MF_TELEPORT: Used when teleporting an actor.

MF_MISSILE: Actor is a projectile.

MF_DROPPED: Actor always acts as if it was dropped.

MF_SHADOW: Actor is nearly invisible and makes monsters trying to target this actor to be inaccurate.

MF_NOBLOOD: Actor does not bleed when hurt.

MF_CORPSE: Actor is a corpse.

MF_INFLOAT: Used internally for floating monsters.
MF_INBOUNCE: Used internally by Heretic bouncing missiles.
MF_COUNTKILL: Counts toward kill percentage.
MF_COUNTITEM: Counts toward item percentage.
MF_SKULLFLY: Actor is a charging monster.
MF_NOTDMATCH: Actor is not spawned in deathmatch games.
MF_SPAWNSOUNDSOURCE: This projectile will play its seesound from the originating actor.
MF_FRIENDLY: This monster doesn't target the player. Instead it attacks other monsters.
MF_UNMORPHED: This actor is an unmorphed version of something else.
MF_NOLIFTDROP: Does not drop when a lift under it lowers.
MF_STEALTH: Actor is a stealth monster.
MF_ICECORPSE: Actor is a frozen corpse.

Linedef flags

ML_BLOCKING: The linedef is solid.
ML_BLOCKMONSTERS: The linedef blocks monsters.
ML_TWOSIDED: The linedef has two sides (two sidedefs).
ML_DONTPEGTOP: Upper texture unpegged.
ML_DONTPEGBOTTOM: Lower texture unpegged.
ML_SECRET: Displays on the automap as a one-sided line.
ML_SOUNDBLOCK: Stops sound propagation after two of these linedefs have been traversed.
ML_DONTDRAW: Prevents this linedef from displaying on the automap.
ML_MAPPED: This linedef will be present on the automap from the start of the level.
ML_REPEAT_SPECIAL: This linedef's special is repeatable.
ML_ADDTRANS: Drawn with additive translucency (internal flag).
ML_MONSTERSCANACTIVATE: Monsters can activate this linedef.
ML_BLOCK_PLAYERS: This linedef blocks players.
ML_BLOCKEVERYTHING: This linedef blocks everything.
ML_ZONEBOUNDARY: This linedef indicates a sound zone boundary.
ML_RAILING: This linedef uses Strife's railing collision logic.
ML_BLOCK_FLOATERS: This linedef blocks floating monsters.
ML_CLIP_MIDTEX: This linedef's middle texture is clipped by the ceiling and floor.
ML_WRAP_MIDTEX: This linedef's middle texture wraps around.
ML_3DMIDTEX: This linedef clips actor movement based on its middle texture.
ML_CHECKSWITCHRANGE: Forces a check to see if the player can really reach a switch.
ML_FIRSTSIDEONLY: This linedef will only be activated when crossed from its front side.
ML_BLOCKPROJECTILE: This linedef blocks projectiles.
ML_BLOCKUSE: This linedef blocks use actions.
ML_BLOCKSIGHT: This linedef blocks actor sight.
ML_BLOCKHITSCAN: This linedef blocks hitscan attacks.

Correct usage

PickActor has trouble if the chosen actor already has a TID. By default, it won't change an existing TID, so your code will do nothing. With PICKAF_FORCETID, you lose the actor's existing TID, which is likely to break other scripts. With PICKAF_RETURNTID, you might get an existing TID that's shared by several other actors.

The only way to be sure is to call PickActor twice with the same arguments.

```
// Get the existing TID, if there is one.  
// A TID of 0 is the same as not having a TID, so "setting" it to 0 will not  
// change anything's TID.  
int old_tid = PickActor(..., /* tid */ 0, actor_mask, wall_mask, PICKAF_RETURNTID);  
  
// Find a TID that's not currently in use.  
int new_tid = UniqueTID();
```

```
// Do the "real" call. This will return true if an actor was actually found,
// and also forcibly set its TID.
if (PickActor(..., /* tid */ new_tid, actor_mask, wall_mask, PICKAF_FORCETID)) {
    // Do some stuff with the temporary TID.
    do_some_stuff_with(new_tid);
    do_some_other_stuff_with(new_tid);

    // When you're done, restore the actor's TID to its original value.
    Thing_ChangeTID(new_tid, old_tid);
}
```

This works because nothing can move between the two PickActor calls, as long as they have the same arguments. While a script is executing, the game state doesn't change and other scripts don't run.

If you need to refer to the found actor later, this approach won't work. You may need to store the found actor as one of the source's actor pointers using SetPointer.

Examples

This example script, when called from a weapon state, will instantly gib any monster actor under the player's crosshairs.

```
Script "InstaGib" (void) {
```

```
    // Get a unique TID to temporarily use.
    int tid = UniqueTID();

    // Attempt to pick an actor where the activator is looking at. Only shootable actors are considered.
    if (PickActor(0, GetActorAngle(0), GetActorPitch(0), 256.0, tid, MF_SOLID | MF_SHOOTABLE)) {

        // Gib the picked actor.
        Thing_Damage2(tid, 9000, "Melee");

        // Reset the now gibbed actor's TID.
        Thing_ChangeTID(tid, 0);
    }
}
```

SetActivatorToTarget
Jump to navigationJump to search

bool SetActivatorToTarget (int tid)

Usage

This changes the activator of the script to the current target of the first actor found with the specified tid. Using a tid of 0 uses the current activator's target.

The search pattern works like this:

If the tid being referenced is a living player, the new activator is first thing the player is aiming at.

If the tid being referenced is a living monster, the new activator is the monster's target.

If the tid being referenced is something that has died, the new activator is its killer.

If the tid being referenced is a projectile, the new activator is the actor who fired it.

If the actor has no target, the activator is the actor itself.

If there are no actors with the given tid, the function returns 0 and leaves the current activator unchanged.

Parameters

tid: TID of the actor whose target will become the activator.

Return value

Returns true if the new activator exists. If there were no actors with the supplied tid, this function returns false and the activator is left unchanged.

Examples

This example will destroy the first non-player actor that the player aims at.

```
script 1 ENTER
{
  while (PlayerNumber() >= 0)
  {
    SetActivatorToTarget(0);
    Delay(1);
  }
  Thing_Destroy(0, YES);
}
```

SetActorAngle

void SetActorAngle (int tid, fixed angle)

Sets the actor's angle.

Use a value for the angle from 0.0 to 1.0.

tid: TID of thing.

angle: Angle to set. This is a fixed point angle.

This works to set the facing of monsters, players, or any other actor.

North = 0.25

West = 0.5

South = 0.75

East = 1.0

Example

This irritating script spins the player round and round for a while when it is activated.

```
script 1 (int spintime)
{
    while (spintime-- > 0)
    {
        SetActorAngle (100, GetActorAngle (100) - 0.02);
        Delay (1);
        print(s:"You spin me right round, baby right round like a record, baby right round, round, round");
    }
}
```

SetActorFlag

`int SetActorFlag (int tid, str flagname, bool value);`

Usage

Change the value of a flag on the actor(s) with the specified TID.

Parameters

tid: TID of the actor(s) to affect. Use 0 to refer to the activator.

flagname: Name of the flag to change.

value: The new flag value.

Return value

The number of actors affected. Actors that match the TID but don't have the flag are not counted (e.g. changing a player flag on a non-player).

Example

This script spawns a Soulsphere as a secret item.

```
script "SpawnSecretSoulsphere" (int spottid)
{
    int soulsphere_tid = 100;
    if (SpawnSpotForced("Soulsphere", spottid, soulsphere_tid))
    {
        SetActorFlag(soulsphere_tid, "COUNTSECRET", TRUE);
    }
}
```

SetActorPitch

`void SetActorPitch (int tid, int pitch)`

Usage

Sets the actor's Pitch. Pitch is a fixed point angle. See `GetActorPitch` for the possible range.

Parameters

`tid`: TID of thing.

`pitch`: Pitch to set.

Examples

This script will reset (or center) the pitch of the activator.

```
script 1 (void)
```

```
{
```

```
    SetActorPitch(0, 0);
```

```
}
```


SetActorPosition

bool SetActorPosition (int tid, fixed x, fixed y, fixed z, bool fog)

Usage

This function sets the x, y, and z coordinates of the specified actor, with or without teleport fog. The coordinates are specified in fixed point. See also: GetActorX, GetActorY, GetActorZ.

Return value

Returns true if the actor position was changed successfully, and false otherwise.

Example

Here is a script that will move a decoration with a tid of 1 to stay at an equal vertical plane with the player.

```
#include "zcommon.acs"
```

```
script 1 ENTER
```

```
{  
    while (TRUE)  
    {  
        SetActorPosition(1, GetActorX(1), GetActorY(1), GetActorZ(0), 0);  
        delay(1);  
    }  
}
```

SetActorProperty

```
void SetActorProperty (int tid, int property, int value)
void SetActorProperty (int tid, int property, float value)
void SetActorProperty (int tid, int property, str value)
```

Parameters

tid: TID of the actor. Use 0 to refer to the activator.

property: One of the properties listed below.

value: The value to which the property must be set.

Actor Properties

APROP_ActiveSound stringSound played when the actor is walking around, as defined in SNDINFO.

APROP_Accuracy integer Accuracy of the actor.

APROP_Alpha fixed point Alpha of the actor. Range is [0.0, 1.0]

APROP_Ambush bool Whether the actor's AMBUSH flag is set or not.

APROP_AttackSound stringSound played when the actor attacks, as defined in SNDINFO.

APROP_AttackZOffset fixed point The attack z offset of the player.

APROP_ChaseGoal bool Walks to goal instead of target if a valid goal is set

APROP_Damage integer Actor's damage.

APROP_DamageFactor fixed point Generic damage factor for the actor. It is applied before any specific DamageFactor.

APROP_DamageMultiplier fixed point Generic damage multiplier for the actor. This is typically used on actors (e.g monsters) to strengthen or weaken the damage they deal from their attacks.

APROP_DamageType stringActor's damage type.

APROP_DeathSound stringSound played when the actor dies, as defined in SNDINFO.

APROP_Dropped bool Whether or not actor has the DROPPED flag. Dropped items are destroyed by closing doors and crushers while non-dropped items are not; and in games with the "Weapons Stay" option of DMFlags turned on, only weapons with the DROPPED flag set to 0 stay. By default, any item not placed originally on the map has the DROPPED flag set to 1.

APROP_Friction fixed point Actor's current friction factor.

APROP_Friendly bool Actor is friendly to the player and hostile to enemies. In addition to setting and clearing the FRIENDLY flag, when the property is set to true, the total kill count of the map is decreased by one for each affected actor, provided they are countable (only hostile monsters are countable), and is increased when the property is set to false.

APROP_FriendlySeeBlocks integer (New from 4.10.0)

The range in which friendly monsters or hostile monsters with SEEFRIENDLYMONSTERS on can see other non-player actors. This value is measured in increments of 128 map units, e.g a value of 32 equals a range of 4096 map units.

APROP_Frightened bool Monster runs away from player.

APROP_Gravity fixed point Current gravity factor of actor.

APROP_Health integer Actor's current health. Setting this property to 0 or less, kills the actor.

APROP_Invulnerable bool Actor will not lose any health.

APROP_JumpZ fixed point Player's jump height. The formula for jumping distance is $(\text{JumpZ} * 2) / 2 + \text{MaxStepHeight}$, and to get a specific JumpZ from a jumping height you want to achieve, use $\text{Sqrt}((\text{jump height} - \text{MaxStepHeight}) / 2) * 2$.

APROP_Mass integer Actor's mass.

APROP_MasterTID integer The TID of the actor linked to by the actor's master field.

APROP_MaxDropOffHeight fixed point Defines the maximum height of a step this actor can climb down when moving.

APROP_MaxStepHeight fixed point Defines the maximum height of a step this actor can climb up when moving.

APROP_MeleeRange fixed point Actor's melee range.

APROP_NameTag stringName of the actor. If the actor has not been explicitly named by the Tag property or in a script, its name is by default the same as its class name.

APROP_NoTrigger bool Whether or not actor has the NOTRIGGER flag.

APROP_NoTarget bool Actor cannot be targeted by other monsters.
 APROP_PainSound stringSound played when the actor is injured, as defined in SNDINFO.
 APROP_ReactionTime integer The time in tics a monster needs to attack back. However, the main use of this property is to serve as a counter for A_Countdown.
 APROP_RenderStyle integer How the actor is rendered:
 STYLE_None Do not draw
 STYLE_Normal Normal; just copy the image to the screen
 STYLE_Fuzzy Draw silhouette using "fuzz" effect
 STYLE_SoulTrans Draw translucent with amount in transsouls CVAR
 STYLE_OptFuzzy Draw as fuzzy or translucent, based on user preference
 STYLE_Stencil Draw as single color
 STYLE_AddStencil Draw as single additive color
 STYLE_AddShaded Treats 8-bit indexed images as an alpha map while applying additive translucency. Index 0 = fully transparent, index 255 = fully opaque.
 STYLE_Translucent Draw translucent
 STYLE_Add Draw additive
 STYLE_Shaded Treats 8-bit indexed images as an alpha map. Index 0 = fully transparent, index 255 = fully opaque.
 STYLE_TranslucentStencil Draw as single translucent color
 STYLE_Shadow Draw dark translucent stencil
 STYLE_Subtract Draw subtractive
 APROP_ScaleX fixed point Horizontal scaling of the actor's sprite. Does not affect collision box size.
 APROP_ScaleY fixed point Vertical scaling of the actor's sprite. Does not affect collision box size.
 APROP_Score integer A simple counter. Score items automatically increase it by their amount.
 APROP_SeeSound stringSound played when actor sees the player, as defined in SNDINFO.
 APROP_SoundClass stringSound class of the player.
 APROP_SpawnHealth integer The current max health of the actor. Only players may have their max health set this way. Note that for them the default value is 0, which is interpreted as "100 unless modified by DeHackEd". The Player.MaxHealth property can change the default value.
 APROP_Species stringSpecies the actor belongs to.
 APROP_Speed fixed point Actor's speed.
 For monsters, this is the distance they move every time A_Chase is called. For projectiles, this is the distance they move each tic. For players, this is multiplied by the player's class speed to determine the final speed the player will move for each tic that they have a movement key held down. Consequently, the standard APROP_Speed for a player is always 1.0, not what their actual speed is.

APROP_Stamina integer Stamina of the actor.
 APROP_StencilColor color Stencil color of the actor, as a hexadecimal value, e.g. 0xFFFFFFFF (white).

APROP_ViewHeight fixed point The view height of the player.

Examples

script 1 (void)

```

{
    //makes things with tid 13 fuzzy, have 1000 health, the
    //boss brain death sound and doubles their current speed
    //(try it, it's fun! ;) )
    SetActorProperty(13, APROP_RENDERSTYLE, STYLE_FUZZY);
    SetActorProperty(13, APROP_HEALTH, 1000);
    SetActorProperty(13, APROP_DEATHSOUND, "brain/death");
    SetActorProperty(13, APROP_SPEED, (GetActorProperty(13, APROP_SPEED) * 2));
}

```

Monster health

Setting the monster's health may cause issues, especially if you're setting it 0 when a monster is already dead.

Do not do this

```
int mon_hp = GetActorProperty(mon_tid, APROP_Health);
if (mon_hp < 0) {
    mon_hp = 0;
    SetActorProperty(mon_tid, APROP_Health, mon_hp); // Do not do this.
}
```

Player speed

Setting the player's speed is an operation that seems to frequently be done incorrectly. As an example, consider an enter script that gives the player 75% of normal speed:

The wrong way

```
script 1 ENTER
```

```
{
```

```
    SetActorProperty(0, APROP_SPEED, (GetActorProperty(0, APROP_SPEED) * 3 / 4));
```

```
}
```

The problem with this script is that it alters the player's previous speed. If this script is used on multiple maps in a hub or even just a single map that gets revisited in a hub, players will find themselves going slower and slower as they switch maps in the hub. There is a much simpler way to do this, knowing that a player's normal speed is always 1.0:

The right way

```
script 1 ENTER
```

```
{
```

```
    SetActorProperty(0, APROP_SPEED, 0.75);
```

```
}
```

SetActorRoll

Warning: This feature is GZDoom specific, and is not compatible with ZDoom!
To see all of GZDoom's specific features, see GZDoom features.
void SetActorRoll (int tid, fixed angle)

Usage

Sets the roll angle for the actors with the specified tid. If tid is 0, it sets the roll angle for the activator of the script. Note that the function has a visual effect on models and player's view (when called for players) only. It has no effect on sprites.

Parameters

tid: the tid of the actor.

angle: the roll angle to set. This is a fixed point angle in the range of 0.0 to 1.0, with:

0.25 → right

0.5 → down

0.75 → left

1.0 → up

SetActorState

```
int SetActorState(int tid, str statename[, bool exact]);
```

Usage

Forces the actor(s) with the matching tid into the specified state, as defined in Decorate. If tid is 0, the activator is affected.

The final parameter exact specifies whether or not partial state name matches are accepted. If you do not specify it or set it to false, if you try to do something like:

```
SetActorState (0, "Foo.Bar");
```

and the actor has a "Foo" state but no "Foo.Bar" state, it will enter the "Foo" state. If you set exact to true:

```
SetActorState (0, "Foo.Bar", TRUE);
```

then the actor must have a "Foo.Bar" state, or it will not change state at all, even if it has a "Foo" state.

The return value for this function is the number of actors that successfully changed state.

Note that you should refrain from using this function for any actors that use the monster AI, or unpredictable results could occur.

SetActorTeleFog

void SetActorTeleFog (int tid, classname telefogsrc, classname telefogdest)

Sets the corresponding actors with the given tid TeleFogSourceType and TeleFogDestType.

If tid is 0, the calling actor is considered the activator. Setting the fog to "none" will cause nothing to be spawned for the respective field. Using an empty string will preserve the current fog for that field.

SetActorVelocity

bool SetActorVelocity (int tid, fixed velx, fixed vely, fixed velz, bool add, bool setbob)

Changes actor velocity.

Arguments

tid: TID of things to affect. If 0, the activator is used.

velx, vely, velz: The desired velocity for the affected things.

add: If true, each affected actor's velocity is modified by the velx, vely and velz parameters, rather than replaced by them.

setbob: If true, the speed adjustment influences the bobbing of any concerned player actor.

Examples

Put a Cacodemon on your map with the matching tid and have some target practice with this script.

```
script 1 (int tid)
{
    int angle, pitch, velx, vely, velz;
    while (GetActorProperty(tid, APROP_Health) > 0)
    {
        angle = random(0, 1.0);
        pitch = random(-0.25, 0.25);
        velx = FixedMul(cos(angle), FixedMul(cos(pitch), 10.0));
        vely = FixedMul(sin(angle), FixedMul(cos(pitch), 10.0));
        velz = FixedMul(sin(pitch), 10.0);
        SetActorVelocity(tid, velx, vely, velz, FALSE, FALSE);
        delay(random(1, 7) * 5);
    }
}
```


SetAirSupply

bool SetAirSupply (int playernum, int tics)

Sets the amount of tics remaining in a player's air supply.

Parameters

playernum: Player number. This information can be obtained through PlayerNumber.

tics: the desired duration in tics before the player will start to drown.

This function targets only players since monsters do not have an air supply – drowning is not implemented for them.

Return value

The function returns TRUE if a player's air supply was successfully altered, FALSE if there are no players for this number.

Examples

This script checks the player's health and gives the player an air supply bonus when above 100 health. It should be executed from an Eyes Go Below Surface actor. The extra air is calculated as a percentage of the players health above 100 where 100 health adds 0% and 200 health adds 100% (giving 700 for 100 health, 1050 for 150 health, up to 1400 for 200 health).

script 1 (void)

```
{
    int health, air;
    health = GetActorProperty(0, APROP_HEALTH);
    air = GetAirSupply(PlayerNumber());

    if (health > 100 && air <= 700)
    {
        air = 700 * (health - 100) / 100 + 700;
        SetAirSupply(PlayerNumber(), air);
    }
    else if (health <= 100 && air > 700)
    {
        SetAirSupply(PlayerNumber(), 700);
    }
}
```

SetAmmoCapacity (ACS)

`void SetAmmoCapacity(str typename, int maxamount);`

Usage

Sets the maximum amount of a type of ammo the player can carry. Any item from the list Ammo is accepted, as well as derived types from Ammo, while any other valid Inventory item will simply be silently ignored.

Examples

An example of backpacks that can be stacked, as in, each backpack increases the max ammo a little:

```
bool playerbackpack[8] = { FALSE };
```

```
script 1 (void)
```

```
{
    if (playerbackpack[PlayerNumber()])
    {
        if (GetAmmoCapacity("Clip") < 800)
        {
            SetAmmoCapacity("Clip", GetAmmoCapacity("Clip") + 100);
            SetAmmoCapacity("Shell", GetAmmoCapacity("Shell") + 25);
            SetAmmoCapacity("RocketAmmo", GetAmmoCapacity("RocketAmmo") + 25);
            SetAmmoCapacity("Cell", GetAmmoCapacity("Cell") + 75);
        }
    }
    else
        playerbackpack[PlayerNumber()] = TRUE;
}
```

```
script 998 ENTER
```

```
{
    if (GetAmmoCapacity("Clip") > 200)
        playerbackpack[PlayerNumber()] = TRUE;
}
```

```
script 999 RESPAWN
```

```
{
    playerbackpack[PlayerNumber()] = FALSE;
}
```

As the backpack will force a change in the amount of ammo the player has the first time it is picked up, this script records the whether each player has picked up a backpack already. There are eight potential players and by default each has their variable set to FALSE. There are two exceptions, which are handled by scripts 998 and 999. 998 checks if the player has a backpack when entering the level by testing their capacity for bullets. 999 resets the player's variable should they die and therefore lose the backpack.

The actual stacking script is number 1, and should be set as the thing special of every backpack. That is, the backpack should have special number 80 (ACS_Execute) with parameters (1, 0, 0, 0, 0). The script checks if the player has already picked up a backpack previously to this, and if so, starts incrementing each type of ammo a little. It does this by first checking if the ammo has reached the absolute maximum amount. If not, it adds a little bit on to each. Note that it only checks if "Clip" has reached the maximum amount, but as each value is incremented at the same time, they will all hit their limits at the same time as the "Clip" type. If the player hasn't picked up a backpack before, they have now, so their variable is set to TRUE.

SetMarineSprite

```
void SetMarineSprite(int tid, str actorclass);
```

Usage

Changes a Scripted Marine's sprite to match the sprite used in the specified actor's spawn state. The sprites you change it to need to correspond to the Marine sprites for it to work 100%, as the same frames are used for the same purposes, so 4x walking, 2x shooting, 1 pain frame, and so on. actorclass is a string, and case sensitive, so "revenant" is not the same as "Revenant".

SetMarineWeapon

```
void SetMarineWeapon(int tid, int weapon);
```

Usage

Sets a Scripted Marine's weapon on the fly. Choices defined in zdefs.acs are as follows:

MARINEWEAPON_Dummy (no weapon)

MARINEWEAPON_Fist

MARINEWEAPON_BerserkFist

MARINEWEAPON_Chainsaw

MARINEWEAPON_Pistol

MARINEWEAPON_Shotgun

MARINEWEAPON_SuperShotgun

MARINEWEAPON_Chaingun

MARINEWEAPON_RocketLauncher

MARINEWEAPON_PlasmaRifle

MARINEWEAPON_Railgun

MARINEWEAPON_BFG

Note in the case of rocket marines they are not immune to splash damage, so you may want to knock up their health a bit with SetActorProperty since they will probably kill themselves pretty quickly otherwise.

SetPointer

Jump to navigationJump to search

bool SetPointer(int assign_slot, int tid[, int pointer_selector[, int flags]])

Usage

Set the value of one of the activator's stored actor pointers.

Note: The function tests for circular reference on master and target fields, setting them to NULL if one is found. That test does not occur when assigning to the tracer field. Any attempt to make the actor point directly to itself will also result in a NULL assignment.

Parameters

int assign_slot: an actor pointer selector. Must refer to an assignable pointer type (target, master, tracer).

int tid: TID of the actor to be stored in the selected slot. 0 selects the activator, but the caller can only be an intermediate selection.

int pointer_selector: if this is specified, the actor specified by TID is used as an intermediate actor. The final value is determined by selecting any available actor pointer from the intermediate actor.

int flags:
PTROP_UNSAFETARGET: don't nullify assignments that result in an infinite chain of missiles referencing each other.
PTROP_UNSAFEMASTER: don't nullify assignments that result in an infinite chain of actors referencing each other.

Return value

TRUE (1) if a non-NULL assignment was made (a suitable actor was found and stored).

FALSE (0) if there is no activator, or if a NULL assignment was made. This function assigns data to the activator, and cannot operate without one.

Examples

This script will cause any monster with provided TID to attack the calling player's current target. It could be used for summoned minions and the script could be activated via a hot-key.

Script "AttackTarget" (int TID) NET

```
{
  int oldTID, target;
  if(SetActivator(0,AAPTR_PLAYER_GETTARGET)) // Get the target of the activator. If it fails because there is none
  nothing will happen.
  {
    oldTID = ActivatorTID(); // Save the TID of the target.
    target = UniqueTID(); // Get a new unique TID for the target actor.
    Thing_ChangeTID(0, target); // Set that TID to the actor.
    SetActivator(TID); // Set the activator to the actor with the given TID.
    SetPointer(AAPTR_TARGET, target); // Give a new target to it.
    Thing_ChangeTID(target,oldTID); // Restore the previous TID, because it might be used already by other scripts.
  }
}
```

SetSubtitleNumber

void SetSubtitleNumber (int num, str sound)

Usage

Parameters

num:

sound:

SetThingSpecial

void SetThingSpecial (int tid, int special [, int arg0 [, int arg1 [, int arg2 [, int arg3 [, int arg4]]]])

Usage

Sets the special for any things with the same TID. This is similar to Thing_SetSpecial, except it can only be used from ACS, and it can set all of a thing's special arguments. If tid is 0, then the activator is used.

Examples

This script will give all with the tid of 1 the special ACS_ExecuteAlways to tally a score when killed.

```
int score[8];
```

```
script 1 OPEN
```

```
{
    SetThingSpecial(1, ACS_ExecuteAlways, 2);
}
```

```
script 2 (void)
```

```
{
    if (PlayerNumber() >= 0)
    {
        score[PlayerNumber()]++;
    }
}
```

SetTranslation

void SetTranslation (int tid, str transname)

Usage

Applies transname translation to the actor or actors with the matching TID. Only translations defined in a TRANSLATE lump are acceptable.

Parameters

tid: The TID of the actor or actors to apply the translation to. If this is 0, the translation is applied to the script's activator.

transname: The name of the translation to apply.

SetUserArray

int SetUserArray (int tid, str name, int pos, int value)

Usage

Sets one of the affected actor's user array-bound variables. Variables of native arrays, arrays which are preceded by the keyword native when declared, cannot be set by this function.

Parameters

tid: the TID of the affected actors. If 0, the script's activator is used.

name: the name of the array. Acceptable array types are int, double and bool.

pos: the position in the array, 0-indexed.

value: the value to give to the array position. If the type of the array is double, this must be passed as a fixed-point value.

Return value

The total number of actors the function iterated through in an attempt to set their array's variable.

Examples

This function adds a rank to the specified 'skill' in the activator.

```
function void SetRanks (int skill, int amt)
{
    SetUserArray(0, "user_skills", skill, amt + GetUserArray(0, "user_skills", skill));
}
```

SetUserVariable

int SetUserVariable (int tid, str name, value)

Usage

Sets one of the affected actor's user variables. Native variables, variables which are preceded by the keyword native when declared, cannot be set by this function.

Parameters

tid: the TID of the affected actors. If 0, the script's activator is used.

name: the name of the variable. Acceptable variable types are int, double and bool.

value: the value to give to the variable. If the type of the variable is double, this must be passed as a fixed-point value.

Return value

The total number of actors the function iterated through in an attempt to set their variable.

Examples

This script increases the amount of XP that the activating enemy rewards upon death.

```
script "EnemyExp" (int amt)
{
    SetUserVariable(0, "user_rewardxp", amt + GetUserVariable(0, "user_rewardxp"));
}
```

Spawn

int Spawn (str classname, fixed x, fixed y, fixed z [, int tid [, int angle]])

Not to be confused with Spawn (ZScript).

Usage

Spawns an actor at the given X, Y and Z coordinates. Optionally a TID and a byte angle can be specified. The coordinates are specified in fixed point.

The Z coordinate is absolute, i.e. not relative to the floor height of the sector. So if you want something to spawn 128 units above a floor that is at a height of 64, then the Z coordinate needs to be $128 + 64 = 192$.

Unlike Thing_Spawn, Spawn does not create teleport fog. If you want teleport fog to appear, use the function again to spawn it manually.

To spawn something at the location of another actor, e.g. a MapSpot, use SpawnSpot instead. To get a list of the things you can spawn in the game, visit the [Classes](#) page.

The return value is the number of things spawned.

Examples

This script spawns a Cacodemon above the player about three seconds after he enters the map. There is a 25% chance to instead spawn a Pain Elemental. Since the script is dependent upon the player's location there is the potential for failure due to coordinate obstruction. In an attempt to account for this, the script analyzes the return value of Spawn to determine success or failure, and upon failure will wait a tic and attempt the spawn again.

```
script 1 ENTER
{
    str class = "Cacodemon";

    if (!random(0, 3))
        class = "PainElemental";

    delay(104);

    do {
        delay(1);
        int x = GetActorX(0);
        int y = GetActorY(0);
        int z = GetActorZ(0) + 100.0;
        int angle = GetActorAngle(0) >> 8;

        } until (Spawn(class, x, y, z, 0, angle));
    }
```

In this example Health Bonuses will "rain" down on the activating player n number of times, where n is the amount passed to the script argument. The effect is achieved by spawning the bonuses above the player's head and at randomized x and y coordinates. This example spawns in a square pattern.

```
script 1 (int n)
{
    if (PlayerNumber() > -1 && n > 0)
    {
```

```
print(s:"It's raining meds!");
```

```
while (n > 0)
```

```
{  
    int x = GetActorX(0) + random(-64.0, 64.0);  
    int y = GetActorY(0) + random(-64.0, 64.0);  
    int z = GetActorCeilingZ(0) - 8.0;
```

```
    if (Spawn("HealthBonus", x, y, z))  
        n--;
```

```
    delay(1);  
}
```

```
}  
}
```

This example uses some trigonometric functions to spawn the bonuses in a circular pattern.

```
script 2 (int n)
```

```
{  
    if (PlayerNumber() > -1 && n > 0)
```

```
{  
    print(s:"It's raining meds!");
```

```
while (n > 0)
```

```
{  
    int angle = random(0, 1.0);  
    int radius = random(0, 64.0);  
    int x = GetActorX(0) + FixedMul(cos(angle), radius);  
    int y = GetActorY(0) + FixedMul(sin(angle), radius);  
    int z = GetActorCeilingZ(0) - 8.0;
```

```
    if (Spawn("HealthBonus", x, y, z))  
        n--;
```

```
    delay(1);  
}
```

```
}  
}
```

SpawnDecal

int SpawnDecal (int tid, str decalname [, int flags [, fixed angle [, int zoffset [, int distance]]]])

Usage

Creates a decal on a wall by tracing a line from the actor with the specified TID until hitting said wall, on which the decal is then created. If tid is 0, the tracing is done from the activator of the script.

Parameters

tid: The TID of the actor.

decalname: The name of the decal to create, as defined in DECALDEF.

flags: These can be combined by using the | character:

SDF_ABSANGLE — If present, angle is an absolute angle. Otherwise, it is relative to the origin actor's angle.

SDF_PERMANENT — If present, the decal will stay indefinitely, and will not be removed if the maximum amount of decals present at once in a level is reached (this is controlled by the cl_maxdecals console variable).

SDF_FIXED_ZOFF — If present, zoffset is treated as a fixed-point value rather than an integer.

SDF_FIXED_DISTANCE — If present, distance is treated as a fixed-point value rather than an integer.

Default is 0.

angle: The direction in which to search for a wall. This is a fixed point angle. Default is 0.

zoffset: The offset from the middle of the origin actor for the Z height of the decal. Y-flipped decal will have reverse offset. Default is 0.

distance: The maximum distance to search for a wall. Default is 64.

Return value

The return value of the function is the number of decals spawned.

SpawnForced

int SpawnForced (str classname, fixed x, fixed y, fixed z [, int tid [, int angle]])

Usage

This is an alternative to Spawn which takes the same parameters. It forces the actor to spawn, even in conditions where the spawning would normally fail (for example, not enough room to place the actor). As such, it should be used carefully.

The coordinates are specified in fixed point (the same format that is used by GetActorX/Y/Z and SetActorPosition). For details, read the article on fixed point.

To get a list of the things you can spawn in the game, visit the [Classes](#) page.

The return value is the number of things spawned.

Examples

See Spawn for examples. SpawnForced uses the same arguments in the same way.

SpawnProjectile

`void SpawnProjectile (int tid, string type, int angle, int speed, int vspeed, int gravity, int newtid);`

Usage

SpawnProjectile is similar to the Thing_Projectile, Thing_ProjectileGravity, and Thing_Projectile2 specials, but instead of using a spawn ID you pass an actor's class name.

Examples

This script will fire an imp's fireball from the script activator's position, using the activator's angle, with a speed of 20.

script 1 (void)

```
{
    SpawnProjectile(0, "DoomImpBall", GetActorAngle(0) >> 8, 20, 0, 0, 0);
}
```

This script will make actors with a TID of 10 fire a Cacodemon's fireball every 10 tics, with an angle of 128 and a speed of 40.

script 2 OPEN

```
{
    SpawnProjectile(10, "CacodemonBall", 128, 40, 0, 0, 0);
    Delay(10);
    Restart;
}
```

This script will fire a random projectile at a random angle from a thing with TID 1 as specified by the projectile array at a horizontal speed of 20.

```
str projectile[3] = {"DoomImpBall", "CacodemonBall", "BaronBall"};
```

script 3 (void)

```
{
    SpawnProjectile(1, projectile[ random(0, 2) ], random(0, 255), 20, 0, 0, 0);
}
```

SpawnSpot

int SpawnSpot (str classname, int spottid [, int tid [, int angle]])

Usage

SpawnSpot requires an actor (e.g. a MapSpot) at the location where the actor will be spawned. To spawn something using exact coordinates, use Spawn. The spawn may fail if something blocks the spawn location. To force the actor to be spawned regardless of this, use SpawnSpotForced.

If there are multiple spot actors with the same tid, SpawnSpot will try to spawn an actor at each one of them.

Note that this function does not make teleport fog appear when the actor(s) spawn. Since the teleport effect is simply another actor, you can manually do this by spawning a TeleportFog actor at the same time.

classname: the actor to spawn. To get a list of the things you can spawn in the game, visit the [Classes](#) pages.

spottid: the thing ID of the actor to spawn at.

tid: the thing ID to give the spawned thing.

angle: byte angle.

The return value is the number of things spawned.

Examples

This will spawn an imp from Doom at the map spot with tid of 64:

```
SpawnSpot ("DoomImp", 64);
```

This will spawn a shotgun at the same mapspot with teleport fog and give it a tid of 200:

```
SpawnSpot ("Shotgun", 64, 200);
```

```
SpawnSpot ("TeleportFog", 64);
```


SpawnSpotFacing

int SpawnSpotFacing (str classname, int spottid [, int tid])

Usage

Like SpawnSpot, this will spawn an actor of the given type at the given mapspot. Like Thing_SpawnFacing, the thing will assume the angle of the mapspot it is spawned to.

The return value is the number of things spawned.

Examples

This script will spawn an imp at a mapspot with a tid of 1, facing the same direction as said mapspot and giving it a new tid of 2.

script 1 (void)

```
{  
    SpawnSpotFacing("DoomImp", 1, 2);  
}
```

SpawnSpotFacingForced

int SpawnSpotFacingForced (str classname, int spottid, int tid)

Usage

This is an alternative to SpawnSpotFacing which takes the same parameters. It forces the actor to spawn, even in conditions where the spawning would normally fail (for example, not enough room to place the actor). As such, it should be used carefully.

classname: the actor to spawn. To get a list of the things you can spawn in the game, visit the [Classes](#) pages.

spottid: the thing id of the MapSpot

tid: the thing id to give the spawned thing

The return value is the number of things spawned.

Examples

See SpawnSpotFacing for examples. SpawnSpotFacingForced uses the same arguments in the same way.

SpawnSpotForced

int SpawnSpotForced (str classname, int spottid, int tid, int angle)

Usage

This is an alternative to SpawnSpot which takes the same parameters. It forces the actor(s) to spawn, even in conditions where the spawning would normally fail (for example, not enough room to place the actor). As such, it should be used carefully.

classname: the actor to spawn. To get a list of the things you can spawn in the game, visit the [Classes](#) pages.

spottid: the thing id of the actor the thing is to be spawned at.

tid: the thing id to give the spawned thing

angle: byte angle.

The return value is the number of things spawned.

Examples

See SpawnSpot for examples. SpawnSpotForced uses the same arguments in the same way.

SwapActorTeleFog

int SwapActorTeleFog (int tid)

Swaps the TeleFogSourceType and TeleFogDestType fields of the corresponding actors with the given tid.

If tid is 0, the calling actor is considered the activator. This function does nothing to those who have the same source and destination fogs.

Returns the number of how many actors were successfully modified, not including those unaffected from same types.

Thing_Damage2

```
int Thing_Damage2 (int tid, int amount, str type);
```

Usage

This is exactly the same as Thing_Damage, except the damage type is specified by name.

The return value is the number of things damaged.

Parameters

tid: TID of the thing you want to damage.

amount: The amount of damage the thing will receive.

type: The damagetype to use. See Damage types.

Thing_Projectile2

void Thing_Projectile2 (int tid, int type, int angle, int speed, int vspeed, int gravity, int newtid)

Usage

Thing_Projectile2 is the same as Thing_Projectile and Thing_ProjectileGravity, except you can give the new projectile a TID. Note that this function requires the actor being spawned to have a SpawnID. To spawn an actor without one, use the SpawnProjectile function instead.

UnMorphActor

int UnMorphActor (int tid[, bool force])

Usage

This function and its complement MorphActor give the ACS coder direct access to the engine's morph subsystem, instead of having to strategically place DECORATE items and suchlike.

Parameters

tid: The actor(s) to unmorph. The activator is used if this parameter is zero.

force: Forces the unmorph to succeed. This parameter exposes an internal engine code that bypasses the "do not unmorph in a wall" thing spawn check; it should be used with caution. For full details and additional notes, please refer to the MorphProjectile class.

Notes

This function does honor the MRF_WHENINVULNERABLE flag when used on a player in the original morphing, provided that player is also the activator of the function. In other words, you cannot force a player to be unmorphed if you are not them; otherwise, it could be used as an exploit if the morphing was originally given by a powerup.

The return value is the number of actors successfully unmorphed. This also means that for TID = 0, it is also a "boolean" (0=failed, 1=succeeded).

Warp

bool Warp (int tid, fixed xofs, fixed yofs, fixed zofs, fixed angle, int flags [, str success_state [, bool exact [, fixed heightoffset [, fixed radiusoffset [, fixed pitch]]]])

Usage

Teleports the calling actor to the actor with the matching tid (reference actor).

Parameters

tid: The tid of the actor to teleport to.

xofs: Specifies how far forward/backward in front/behind of the reference actor the calling actor will appear. Positive numbers result in the actor spawning in front, and behind for negative.

yofs: Specifies how far to the side the calling actor will appear to the reference actor. Positive numbers are further to the reference actor's right, while negative numbers spawn them more to the left.

zofs: Specifies how high off the ground to appear. Positive numbers mean a higher altitude, negative means under the reference actor and possibly into the ground.

angle: Specifies the offset to add after the calling actor sets its angle to the reference actor's angle.

flags: Flags which can be used to alter the behavior and appearance of the warp. More than one flag can be combined by using the pipe | character between the constant names:

Flags that affect the behavior:

WARPF_ABSOLUTEOFFSET — Do not apply the angle to the xy-offset.

WARPF_ABSOLUTEANGLE — Use the angle parameter as an absolute angle (not an offset).

WARPF_ABSOLUTEPOSITION — Treat x, y and z offset parameters as absolute coordinates for the calling actor's position, instead of being relative to the reference actor's. This flag overrides

WARPF_ABSOLUTEOFFSET, but can still add the z offset of floorz when used with WARPF_TOFLOOR.

WARPF_USECALLERANGLE — Use the calling actor's angle instead of the reference actor's. Note: The angle will add itself to the caller's angle and cause it to orbit around the reference. The greater the angle, the faster it will orbit.

WARPF_NOCHECKPOSITION — Blindly accept any resultant position.

WARPF_STOP — Reduce caller velocity to 0 when the move is completed.

WARPF_TOFLOOR — Set caller z relative to floor z where teleported to, instead of relative to reference actor.

WARPF_TESTONLY — Does not warp the actor, but still allows it to jump to the success state (see below) if it were to warp. This is useful for checking things and causing chains to occur, such as with inventory items.

WARPF_BOB — Gets the float bob offsets of the reference actor and offsets it to follow with it.

WARPF_MOVEPTR — The reference actor does the warping instead of the calling actor, and all the flags apply. The calling actor is still responsible for performing a state jump and determining success, however.

WARPF_USEPTR — If set, the first parameter changes to accept an actor pointer instead of a tid.

WARPF_COPYVELOCITY — Copies the exact velocity of the actor, which is relative only to the reference actor's own, regardless of what angle the warped actor receives.

WARPF_COPYPITCH — Gains the same pitch as the reference actor, and then adds pitch after it, if any.

Flags to customize appearance of the warp:

WARPF_INTERPOLATE — Keep old interpolation data (make it appear as if actor moved from its previous location).

WARPF_WARPINTERPOLATION — Modify interpolation data with the vector PlayerNewPosition - PlayerOldPosition.

WARPF_COPYINTERPOLATION — Copies the actor's interpolation data for itself, allowing the actor to mimic the reference actor closely.

success_state: An optional state to jump to in the event of success. Default is "".

exact: Specifies whether or not partial state name matches are accepted for the success state. If set to true, the calling actor must have a state with a name that exactly matches the specified success state's name. If set to false, however, partial name match is enough (see SetActorState for further information and examples). Default is false.

heightoffset: A multiplier for the reference actor's height. This allows for further adjustment to the calling actor's z-offset by way of adding the result of multiplying the reference actor's height by the passed value to the z-offset. Default is 0.

radiusoffset: A multiplier for the reference actor's radius. Similar to heightoffset but using radius instead.

pitch: The amount of pitch to add to the offset, negative values aim the actor higher while positive will lower it. Unlike angle, this is not copied unless `WARPF_COPYPITCH` is specified for compatibility purposes.
NOTE: If the actor being orbited stops existing, and the success state is defined, it will ultimately fail to jump. Even if success state is left undefined, though, it will fail to move at all. The success state can be useful for getting rid of orbiting/external actors when the main actor is no longer being used.

Return value

The function returns false if the reference actor does not exist or the warping is unsuccessful, otherwise it returns true.

HudMessage

```
void HudMessage (text; int type, int id, int color, fixed x, fixed y, fixed holdTime [, fixed alpha]);
void HudMessage (text; HUDMSG_FADEOUT, int id, int color, fixed x, fixed y, fixed holdTime, fixed fadetime
[, fixed alpha]);
void HudMessage (text; HUDMSG_TYPEON, int id, int color, fixed x, fixed y, fixed holdTime, fixed typetime,
fixed fadetime [, fixed alpha]);
void HudMessage (text; HUDMSG_FADEINOUT, int id, int color, fixed x, fixed y, fixed holdTime, fixed inTime,
fixed outTime [, fixed alpha]);
```

Usage
Hudmessage works like Print, except it offers more flexibility. The 'text' parameter is in the Print specification, as used with Print or PrintBold. A semicolon separates it from the rest of the function's parameters, because a print specification can build a big string from littler comma-separated parts.

You can also use HudMessage together with SetFont to print images instead of text.

Parameters























Type
The type of message to create. Currently, there are 4 different ways to display a message: HUDMSG_PLAIN, HUDMSG_FADEOUT, HUDMSG_TYPEON and HUDMSG_FADEINOUT. Messages types other than HUDMSG_PLAIN also take additional parameters, listed below. Additionally you can use use various flags, listed below.

Id
Id is an identifier for the message. If a message with a non-zero id is displayed, then another with that same id is displayed, the first message will be removed before displaying the second. This is usually used to remove hudmessages manually or "move" them around (using a script). Messages with lower ids will overlap those with higher ids. Negative Ids (-1, -2, -3, etc) will work as well. Acceptable values are -2147483648 to 2147483647.

Color
The color of the message text. There are a number of predefined colors, listed below.

Color name	Color	Color name	Color	Color name	Color	Notes
CR_BLACK		CR_DARKRED		CR_ORANGE		

CR_YELLOW is meant to replicate the yellow font in Heretic and Hexen, where letters have a gray border.
CR_GREY and CR_DARKGREY can be used instead of CR_GRAY and CR_DARKGRAY.
To use the font graphics' original colors, use CR_UNTRANSLATED.

 CR_BLUE	 CR_FIRE	 CR_PURPLE
 CR_BRICK	 CR_GOLD	 CR_RED
 CR_BROWN	 CR_GRAY	 CR_SAPPHIRE
 CR_CREAM	 CR_GREEN	 CR_TAN
 CR_CYAN	 CR_ICE	 CR_TEAL
 CR_DARKBROWN	 CR_LIGHTBLUE	 CR_WHITE
 CR_DARKGRAY	 CR_OLIVE	 CR_YELLOW
 CR_DARKGREEN		

You can also specify the color as a string (including custom colors defined in the text colors lump); if you do so you must also add HUDMSG_COLORSTRING to the type with a binary OR ('| '). For example, if you define a color called 'NeonRainbowSmoothie', you could use it like this:

```
hudmessage(s:"test"; HUDMSG_PLAIN | HUDMSG_COLORSTRING, 0, "NeonRainbowSmoothie", 0.5, 0.5, 1.0);
```

x and y
The position of the hudmessage to be rendered on the screen in fixed point coordinates.
You can think of each message being inside a box. The difference between positive and negative values for

these parameters is that positive values position the box the message is in on the screen, and negative values position the left/top edge of the box the message is in.

[0.0, 1.0]: Position between left and right edge valid box locations

[-1.0, 0.0): Position between left and right edge of screen

(1.0, 2.0]: Same as [0.0,1.0], but center each line inside box

[-2.0, -1.0): Same as [-1.0,0.0), but center each line inside box

Valid values for y are:

[0.0, 1.0]: Position between top and bottom of valid box locations

[-1.0, 0.0): Position between top and bottom edge of screen

Note: The coordinate meanings are changed completely after using the SetHudSize function. Please see the Coordinate Behavior section for instructions on how to position text messages and images after the hud size has been set.

HoldTime

How many seconds the message stays on screen in fixed point. Use a floating point number with a decimal to obey actual seconds. If using HUDMSG_PLAIN, a HoldTime of 0 will cause the message to stay forever, or until the same ID is re-used. To get the shortest time possible for the message to stay on the screen (1 tic), use the formula $1.0 / 35 + 1$ (or simply 1873 as an integer).

Alpha

Optionally, an alpha value can be specified for the message. This is a fixed point value ranging from 0 (totally invisible) to 1.0 (totally opaque), defaulting to 1.0 if omitted. Additionally, additive blending is possible by combining one of the four message types using the pipe character "|" with HUDMSG_ADDBLEND. Note that to effectively use an alpha value, you need to explicitly add the HUDMSG_ALPHA flag. This is to preserve compatibility with mods predating the addition of the alpha feature but which mistakenly specified additional, then unused, parameters.

Additional type parameters

Note that you cannot specify an infinite (0) HoldTime when using any of these message types:

HUDMSG_FADEOUT

..., fixed fadetime);

Fadetime is the time, in seconds, that the message takes to fade out after its holdtime is up.

HUDMSG_TYPEON

..., fixed typetime, fixed fadetime);

Typetime is the time, in seconds, that it takes each character of the message to appear on the screen. After every character has been "typed," the message waits for holdtime seconds and then fades out for fadetime seconds.

HUDMSG_FADEINOUT

..., fixed inTime, fixed outTime);

inTime is the time, in seconds, that the message takes to fade in. The message then stays for the duration of its holdTime, then fades out using the duration specified in outTime.

Type flags

The following can be used in conjunction with the other types and flags by using the pipe character "|".

HUDMSG_ADDBLEND

The message is rendered with additive blending. This can be combined with an alpha value.

HUDMSG_LOG

The message is logged to the console.

HUDMSG_ALPHA

The message uses the alpha parameter.

HUDMSG_NOWRAP

Wrapping is disabled for this message.

Visibility flags

The following can be used in conjunction with one of the four message types above by using the pipe character "|" to determine whether the message is shown depending on the currently active view:

HUDMSG_NOTWITH3DVIEW

This message does not appear when the 3D view is active.

HUDMSG_NOTWITHFULLMAP

This message does not appear when the fullscreen automap is active.

HUDMSG_NOTWITHOVERLAYMAP

This message does not appear when the overlay automap is active.

These flags may be combined, so for example: HUDMSG_NOTWITHFULLMAP |

HUDMSG_NOTWITHOVERLAYMAP would prevent the message from appearing if any form of automap is active.

HUD layers

The following can be used in conjunction with one of the four message types above by using the pipe character "|" to determine where the message is rendered when it comes to the HUD layers:

HUDMSG_LAYER_OVERHUD

This is the default and standard behavior. The message appear on top of most HUD elements. This definition is only included for completeness' sake; you do not need to explicitly use it.

HUDMSG_LAYER_UNDERHUD

The message appears underneath other HUD elements, such as the status bar.

HUDMSG_LAYER_OVERMAP

The message appears on top of the fullscreen automap.

Combining one of the above with type flags is possible. For instance, HUDMSG_NOTWITHFULLMAP | HUDMSG_NOTWITHOVERLAYMAP | HUDMSG_LAYER_UNDERHUD would render the message under the status bar and only when the 3D view is active.

Note that these are not flags, thus can't be combined with each other, so for example HUDMSG_LAYER_UNDERHUD | HUDMSG_LAYER_OVERHUD is not valid.

Examples

Examples of x and y positions

(0.5,0.0) positions the message at the middle top of the screen.

(0.0,0.5) positions the message at the middle left of the screen.

(-0.25,0.0) positions the left edge of the message 1/4 of the way across the screen at the top.

(1.5,0.5) centers the box on the screen, and also centers each line inside the box.

Example scripts

This script gives a goal using the standard large font which is seen for the level names. The goal tells the player to destroy a certain number of a certain object. The amount and object type are decided by the parameters to this script; so giving parameters 5 and 0 would tell the player to destroy 5 generators. The text types on to the screen and fades out, and also stays on the console.

```
str goals[3] = {"generators", "altars", "teleporters"};
```

```
script 10 (int amount, int goalttype)
```

```
{
    SetFont("BIGFONT");
    HudMessage(s:"You must find ",
               d:amount, s:" ", l:goals[goalttype],
               s:"\nand destroy them.");
    HUDMSG_TYPEON | HUDMSG_LOG, 0, CR_TAN, 1.5, 0.8, 5.0,
    0.05, 2.0);
}
```

This demonstrates that the message to display can be built of smaller strings and also variables. \n is the new-line character. SetFont is used to change the font.

Another example is a damage counter, which stays on screen reporting the damage the player has until they are healed.

```
script 1 (void)
```

```

{
    int health = GetActorProperty(0, APROP_HEALTH);

    while(health < 100)
    {
        HudMessage(s:"You have ",
                   d:(100 - health),
                   s:" damage!");
        HUDMSG_PLAIN, 1, CR_RED, 0.1, 0.9, 1.0);

        Delay(1);

        health = GetActorProperty(0, APROP_HEALTH);
    }

    HudMessage(s:"You are in good health.";
               HUDMSG_PLAIN, 1, CR_RED, 0.1, 0.9, 5.0);
}

```

First this sets the player's health to a variable and loops whilst the health is less than 100 (i.e. whilst they are damaged). The current damage is displayed, before a frame's delay and then checking the health again. After the player is healed (health == 100) the while loop exits and a final message reports that the player is healed.

Note that the messages have an ID number of 1. If they were to lack an ID number, each health message would be written over the previous causing it to look very glitchy. The ID number prevents this from happening.

An example of a moving text:

```

script 1 (void)
{
    SetFont("BIGFONT");

    // Vary the x coordinate
    for(int x = 0.0; x < 0.5; x += 0.01)
    {
        // Show the message at our x coord
        HudMessage(s:"You got pommied\nby the appler";
                   0, 1, CR_TAN, x, 0.4, 2.0);
        // Slight delay
        Delay(1);
    }

    // Display the message once again at the final x coordinate
    HudMessage(s:"You got pommied\nby the appler";
               0, 1, CR_TAN, x, 0.4, 2.0);
}

```

This uses a for loop to change the value of x so that it scrolls across the screen. It starts at (0.0, 0.4) which means the text will be placed at the left hand edge, slightly above the center line. It finishes at (0.5, 0.4) which is just above the absolute center of the screen.

Displaying images

HudMessage can be used in conjunction with SetFont to display graphic lumps. Pass the name of a graphic lump as a parameter of SetFont, and use HudMessage with a capital 'A' to display the graphic. It is also worth noting that images displayed will scale if used in conjunction with SetHudSize. For instance, an image taken with dimensions 320 x 200 will be shrunk in half if SetHudSize is set to 640 x 400

The image lump name must be 8 characters or less

script 02 (void)

```
{  
    SetFont("PICTURE");  
    HudMessage(s:"A"; HUDMSG_PLAIN, 0, 0, 0.1, 0.8, 3.7);  
}
```

HudMessageBold

```
void HudMessageBold (text; int type, int id, int color, fixed x, fixed y, fixed holdTime [, fixed alpha]);  
void HudMessageBold (text; HUDMSG_FADEOUT, int id, int color, fixed x, fixed y, fixed holdTime, fixed fadetime [, fixed alpha]);  
void HudMessageBold (text; HUDMSG_TYPEON, int id, int color, fixed x, fixed y, fixed holdTime, fixed typetime, fixed fadetime [, fixed alpha]);  
void HudMessageBold (text; HUDMSG_FADEINOUT, int id, int color, fixed x, fixed y, fixed holdTime, fixed inTime, fixed outTime [, fixed alpha]);
```

Usage

HudMessageBold is to HudMessage as PrintBold is to Print. It prints a message in the same manner as HudMessage but for all players. See the HudMessage page for the usage of this function.

Examples

This command has a number of uses. One example would be printing a message to the player when a monster activates a script. Another is in multiplayer deathmatch or more specifically cooperative games. This following script reports to the player who has opened a door, and tells everyone in the game a similar message.

```
script 44 (int door)  
{  
    Door_Open(door, 10);  
  
    HudMessage(s:"You opened the hangar door!";  
        HUDMSG_PLAIN, 0, CR_RED, 0.5, 0.8, 5.0);  
    HudMessageBold(s:"Player ", d:PlayerNumber() + 1,  
        s:" opened the hangar door!";  
        HUDMSG_PLAIN, 0, CR_RED, 0.5, 0.9, 5.0);  
}
```

The one issue with this script is that the player opening the door gets both messages.

Log

`void Log(item(s));`

Usage

Log will print something in the log area of the screen (top left), as well as logging it to the console. It uses the same parameter format as the Print function.

Example

script 1 (void)

```
{
    AmbientSound("items/quaddamage", 127);
    Log(s:"You've been granted Quad Damage!");
    GiveInventory("PowerQuadDamage", 1);

    Delay(25 * 35);

    AmbientSound("items/quadwearingoff", 127);
    Log(s:"Quad Damage is running out...");

    Delay(5 * 35);

    AmbientSound("items/quadgone", 127);
    TakeInventory("PowerQuadDamage", 1);
}
```


Print

```
void Print(item(s));
```

Contents

- 1 Usage
- 1.1 Cast type
- 1.2 Escape sequence
- 2 Colors
- 3 Examples

Usage

Print will print something to the activating player's screen. This can be text or variables or any combination of the two. Note that Print is fairly inflexible (there is no way to modify the position on screen or the time the message stays up) so you may wish to consider using HudMessage instead.

Print will only display for the activator of the script, so if another player activates a script with a Print statement in it, then it will only be displayed for that other player and you will not see it. For printing to all players, see PrintBold. If a script is activated by something that is not a player at all then the message will simply not be displayed anywhere.

Note: Do not use Print as a debug tool. In many scripting and programming languages it is quite common to check that a complex code executes properly by MiniWikipediaLogoIcon.pngtracing its operation with print or printf statements; in ACS, you need to use Log or PrintBold or the trace will possibly not display at all, making it look like the script does not actually run.

Cast type

Item syntax

<cast type>:<expression>

The string can be built up of any number of these items, separated by commas. The cast type is used to differentiate what type of data is being displayed. The table below shows the available options.

cast type description

a prints a character array. (Ex.: a:arrayname)

a:(arrayname[, int starting_index[, int size_limit]]) prints the character array specified by arrayname starting from starting_index, printing only up to size_limit characters.

b binary number

c character

d decimal number (integer, same as i)

f fixed point number

i decimal number (integer, same as d)

k prints the key(s) (up to 2) that the player has bound to the specified command. (Ex.: k:"+use")

l localized string from LANGUAGE

n name, expression could be:

PRINTNAME_LEVELNAME - Prints the level name (e.g., "Entryway").

PRINTNAME_LEVEL - Prints the map lump name (e.g., "MAP01").

PRINTNAME_SKILL - Prints the skill name (e.g., "Hurt me plenty").

PRINTNAME_NEXTLEVEL - Prints the map lump name of the next map.

PRINTNAME_NEXTSECRET - Prints the map lump name of the next secret map.

Using 0 as expression prints the name of the activator (a tag name if the activator isn't a player, player name otherwise). Values greater than 0 will print the name of the n-th player (one-based).

s string

x hexadecimal number

Escape sequence

The other special characters to note are escaped characters. A new line is started with `\n`. A backslash character is placed with `\\`, except there is no graphic for backslash by default in Doom.



























Escape sequences are contained within a string. They begin with the backslash character: `'\'`.

Available escape sequences, as handled by the `strbin` function in `cmdlib.cpp`, include:

sequence	description
<code>\OOO</code>	Inserts a character by the octal value of its ASCII code. OOO must be a valid three-digit octal number.
<code>\a</code>	Beep(Verification needed)
<code>\b</code>	Backspace (erases previous character)(Verification needed)
<code>\cCODE</code>	Beginning of a color escape sequence, see below for color codes.
<code>\f</code>	Clear string buffer(Verification needed)
<code>\n</code>	New line
<code>\r</code>	New line (Advisable to use <code>\n</code>)(Verification needed)
<code>\t</code>	Tabulation
<code>\v</code>	Vertical tabulation(Verification needed)
<code>\xXX</code>	Inserts a character by the hexadecimal value of its ASCII code. XX must be a valid two-digit hexadecimal number.
<code>\?</code>	Delete (erases next character)(Verification needed)
<code>\</code>	Ignore the "new line" separator. This can be used to split a long string on several lines in the script source code. Make sure there aren't any space or other invisible character after the backslash and before the end of the line, however.
<code>"</code>	Double quotations. This is the only way to insert double quotes properly into the string, since they otherwise terminate without the backslash.

Colors

You can add color to lines of text by using the `\cX` escape code. Replace X with the following letter to produce the desired color:

code	color	code	color
a		n	
b		o	
c		p	
d		q	
e		r	
f		s	
g		t	
h		u	
i		v	
j		w	
k		x	
l		y	
m		z	
-	Default color for Print messages *		
+	Default color for PrintBold messages !		

The color will revert back to the default message color at the end of every line, therefore a `\cX` escape must be on every line where color is needed. Using `\c-` also clears color, so that printed messages (obituaries, pickup messages, etc.) with player names within them will retain their individual colors.

For example,

```
print(s:"\cgRoses are red\n\chViolets are blue\n\cjSilver for me\n\cfGold for you");
```

will be displayed as `Printcolor.png`.

You can also use the syntax `\c[colorname]`, where `colorname` is a name of a color as defined in the

textcolors lump.

For example,

```
print(s:"\c[Green]This is green");  
Would print
```

This is green

But on some versions of the engine this variant will not print only one symbol after escape sequence to stdout (not in in-game console), e. g. previous example on the OS console will be shown as "GreenThis is green".

Examples

Some basic uses of print.

```
script 1 (void)
```

```
{  
    int x = 10;  
    Print(s:"This is a string"); //prints "This is a string"  
    Delay(35*5);  
  
    Print(d:x); //prints "10"  
    Delay(35*5);  
  
    Print(s:"This is x: ", d:x); //prints "This is x: 10"  
    Delay(35*5);  
  
    Print(s:"I need ", d:x, s:" shells"); //prints "I need 10 shells"  
}
```

Another example of a (potentially) useful debugging script which continually updates the player on their current game coordinates:

```
script 1 ENTER
```

```
{  
    While(TRUE)  
    {  
        Print(f:GetActorX(0), s:" : ",  
              f:GetActorY(0));  
        Delay(10);  
    }  
}
```

This example tells the user to press whatever key they have bound to +use:

```
script 1 (void)
```

```
{  
    Print(s:"Press ", k:"+use", s:" to use this ladder.");  
}
```

PrintBold

Note: due to an oversight, the engine executes Print when this function is called. The function still prints to all player screens, but its default color is different. To rectify that, the CorrectPrintBold MAPINFO property must be set to true.

```
void PrintBold(item(s));
```

Usage

This is exactly the same as Print, except all players will see the printed text on the screen instead of just the activator of the script. It also uses a different default color. For a detailed description of the syntax, refer to Print.

Examples

This command is useful if you want a monster triggered script to print something to the screen. One way a monster can trigger a script is by setting it to their special field, so that when they die (or are otherwise activated), the script is run. Another possibility is to have ACS_Execute (or one of its variants) in one of the state in a custom actor's DECORATE code. This script takes one parameter, which is the TID of the monster which has this script.

```
script 99 (int monsterid)
{
    Thing_SpawnNoFog(monsterid, T_REDSKULLKEY, 0, 0);
    PrintBold(s:"The boss has been defeated!");
}
```

To clarify: set the monster TID to something unique, and the special to 80 (ACS_Execute). Set the parameters to (99, 0, monster's TID, 0, 0). When the monster is killed, a message will be reported to all players and the monster will drop a red skull key.

SetFont

void SetFont (str fontlump);

Usage

Sets the current font (within only the script) to fontlump, which is a lump you can make using imagetool. In addition to anything defined in FONTDEFS, ZDoom has the following available fonts:

CONFONT

Fixed width, this is the same font used in the console

SMALLFONT

The standard small font (shown for between level text). Default.

SMALLFONT2

An alternate small font, found only in Strife. In other games, this is an alias to SMALLFONT.

BIGFONT

Larger version of the font (used for level names on intermission screens).

Once the font is set, the remaining Print, PrintBold, HudMessage and HudMessageBold commands will use that font. You may also check if the font exists by using CheckFont.

Examples

This example is a rather useful script that displays an objective for all the players, and each objective only appears once ever.

```
#define numobjs 4
str objs[4] =
{"Gain access to the warehouse",
"Restart the generator",
"Defeat the intruders and\nsecure the warehouse",
"Evacuate to the helipad"};

script 255 (int num)
{
    if (num >= 1 && num <= numobjs)
    {
        if (objs[num - 1] != "")
        {
            SetFont("BIGFONT");
            HudMessageBold(s:"New objective:\n", l:objs[num - 1];
                HUDMSG_TYPEON | HUDMSG_LOG, 0, CR_TAN, 1.5, 0.5, 5.0, 0.01, 1.0);
            objs[num - 1] = "";
        }
    }
}
```

The first block of code sets up the data for the objectives. This can be freely modified. In the script, the first "if" statement checks if the objective number is valid, and the second checks if the objective has not already been shown. Then the font is set to the large intermission font. Finally the objective is displayed and then removed from the list.

It is also interesting to note that in combination with HudMessage, SetFont can be used to display pictures. Simply place an image into your wad and use SetFont with the image's name. Then create a HudMessage with the letter "A" as the text.

```
script 2 (void)
{
    SetFont("PICTURE");
    HudMessage(s:"A"; HUDMSG_PLAIN, 0, CR_UNTRANSLATED, 0.1, 0.8, 3.7);
}
```

}

SetHudClipRect

`void SetHUDClipRect (int x, int y, int width, int height [, int wrapwidth [, bool aspectratio]]);`

Usage

Set the clipping rectangle for future HUD messages. If you do not specify wrapwidth, the HUD message will be laid out as normal, but pixels outside the rectangle will not be drawn. If you specify wrapwidth, then the message will be wrapped to that width. Use `SetHUDClipRect(0, 0, 0, 0, 0)` to reset everything back to normal. Coordinates for consecutive HUD messages are treated the same.

Parameters

x

The X coordinate at which the clipping rectangle starts.

y

The Y coordinate at which the clipping rectangle starts.

width

The width of the clipping rectangle.

height

The height of the clipping rectangle.

wrapwidth

Wraps text to this amount of pixels, starting at x. If set to 0, the default wrapping method will take place instead. Default is 0.

aspectratio

If set to true, it forces the clipping rectangle to assume a 4:3 aspect ratio if the user's current aspect ratio is 16:9 or 16:10. If set to false, however, the user's current aspect ratio is used, instead.

Default is true.

Examples

`script 1 ENTER`

```
{
    while(true)
    {
        SetHudClipRect(100, 100, 100, 100, 100);
        HudMessage(s:"cool hud message text goes here, and will be wrapped."; HUDMSG_PLAIN, 10, CR_WHITE, 100,
100, 0.1);
        SetHudClipRect(0, 0, 0, 0);
        Delay(1);
    }
}
```

SetHudSize

```
void SetHudSize(int width, int height, bool statusbar);
```

Usage

Causes text messages to be stretched or shrunk to match the size they would appear if the user's resolution was width by height.

Note that it only applies to the 4:3 area of the screen in the center; this command will not cause text to be distorted in non-4:3 resolutions. SetHudSize does not interpret the given values in any way to match it to other aspect ratios.

SetHudSize is designed to be used for hudmessages that are really graphics, using a font which contains images instead of letters. It can also be used to make text appear the same size at all resolutions, for example a title with a special font or something.

After using SetHudSize, HudMessage and HudMessageBold coordinates also behave differently.

Using SetHudSize with either the width or height set to 0 will reset the HudMessage behavior to the default.

Parameters

width: Width of simulated drawing area

height: Height of simulated drawing area

statusbar:

FALSE: height does not include status bar

TRUE: height includes status bar, and it can be drawn on

If statusbar is TRUE, then HudMessage will be stretched differently depending on whether the status bar is visible or not, because the height available to draw in is shorter when the status bar is visible. However, a value of FALSE will still let you draw on the status bar if the HudMessage extends below the height you specified.

Coordinate Behavior

After SetHudSize, you need to specify actual pixel coordinates and not numbers in the [0.0, 1.0] or similar ranges. However, they are still fixed point, so you need to keep the decimal point. If you used SetHudSize(320, 200, TRUE); and want to draw a hudmessage at the center of the screen, you should pass it the coordinates (160.0, 100.0) and not (0.5, 0.5).

The fractional part of the coordinates affect what part of the hudmessage you are positioning:

For x:

.0 = positions center of box

.1 = positions left edge of box

.2 = positions right edge of box

.4 = centers text inside box and aligns center

.5 = centers text and aligns left edge

.6 = centers text and aligns right edge

For y:

.0 = positions center of box

.1 = positions top edge of box

.2 = positions bottom edge of box

So if you used SetHudSize(320, 200, TRUE), then the coordinates (160.1, 100.1) will position the upper-left corner of the hud message. The coordinates (160.2, 100.2) will position its lower-right corner instead.

To make HudMessage behave as normal, use SetHudSize(0, 0, FALSE).

Examples

This script uses `SetHudSize` to make a boss health display. It does not use any external graphics, only the default font, so it is a little primitive. It uses a row of `']'` characters to build up the scale, and a `'>'` to mark the bosses health. It takes the TID of the monster which is the boss as its parameter.

```
script 1 (int monster)
{
    SetFont("BIGFONT");
    SetHudSize(320, 200, 0);

    int h, maxh = GetActorProperty(monster, APROP_HEALTH);

    while(ThingCount(T_NONE, monster) > 0)
    {
        h = 175 - ((GetActorProperty(monster, APROP_HEALTH) * 150)/maxh);

        HudMessage(s:"]\n]\n]\n]\n]\n]\n]\n]\n]\n]\n]\n]\n]";
            HUDMSG_PLAIN, 1, CR_RED, 300.0, 100.0, 5.0);
        HudMessage(s:">";
            HUDMSG_PLAIN, 2, CR_RED, 280.0, h<<16, 5.0);

        Delay(5);
    }

    HudMessage(s:">";
        HUDMSG_PLAIN, 2, CR_RED, 280.0, 175.0, 4.86);
}
```

The first two lines set up the HUD. Next, the variables are prepared, `h` storing the current health, and `maxh` storing the maximum health which the monster is assumed to have at the start of the script. The while loop keeps the display updated whilst the monster is still alive.

The line that calculates the value of `h` is a typical rule of three. The method that it uses is called rearranging a scale. The variable `h` starts at `maxh` and then, during the battle, will eventually drop to 0. However, we want to display the mark starting at 25 pixels from the top of the HUD and finishing at 175 pixels from the top (25 from the bottom). So, as the health goes from `maxh` to 0, `h` needs to go from 25 to 175.

The first step is done by taking percentages. If h is the current health of the monster, then (h / maxh) is the fraction between 25 and 175 the $>$ must be placed. Now, the number of pixels between 25 and 175 is $175 - 25 = 150$. So, $(150 * (h / \text{maxh}))$ is the actual amount of pixels between 25 and 175 to place the $>$ marker. Due to integer division, this must be refactored to $((150 * h) / \text{maxh})$. Finally, it starts at 25 and ends at 175, so the value must be subtracted from 175 to get the final formula.

The two HudMessages render the counter. The delay stops a runaway script and sets to update every fifth frame. Finally, after the loop, the '>' marker is drawn once more at 0% health just to give a good representation of the monster's death to the player.

SetHudWrapWidth

```
void SetHudWrapWidth (int wrapwidth);
```

Usage

Sets the wrapping width for future HUD messages without altering the clipping rectangle. If you set the wrapping width to 0, messages will wrap to the full width of the HUD, as normal.

Parameters

wrapwidth

(Need more info)

SetMugShotState

void SetMugShotState (str state) â€” ACS version

void A_SetMugshotState (String name) â€” Action function version

Usage

Used to set the state of the mug shot in SBARINFO status bars. The state you set will only be interrupted by damage or if the player picks up a weapon, provided the mugshot supports it.

Examples

The following example will make the player believe that they are invulnerable or make spectators think he/she is cheating.

```
script 1 (void)
```

```
{  
    SetMugShotState("God");  
}
```

The player grins when they pick up this berserk pack.

```
ACTOR GrinBerserk : Berserk
```

```
{  
    States  
    {  
        Pickup:  
            TNT1 A 0 A_SetMugshotState("Grin")  
            Goto Super::Pickup  
    }  
}
```

StrCpy

```
bool StrCpy (a:destination, string source[, int source_index]);
```

Usage

Copy a source string to a destination array as a series of characters. Optionally, the copy can start from a given index in the source string.

Return Value

True if the entire string (or substring) was successfully copied to the array; false if the copy ran out of room or if a negative source_index was given.

Example

This function returns true if the map lump is in the form "ExMx". It copies the string to a char array, then it compares individual characters at 2 positions, and skips the character in IsMap_LumpChar[1].

```
int IsMap_LumpChar[8];  
str IsMap_LumpStr;
```

```
function bool IsMap_EXMX (void)  
{  
    if(StrCpy(a:IsMap_LumpChar, IsMap_LumpStr))  
    {  
        int c_e = IsMap_LumpChar[0];  
        int c_m = IsMap_LumpChar[2];  
  
        return( (c_e == 'e' || c_e == 'E') && (c_m == 'm' || c_m == 'M') );  
    }  
    return FALSE;  
}
```

StrLeft

str StrLeft (str string, int length)

Creates a new string containing the length first characters of string. If string does not exist, an empty string is returned. If string is shorter than length characters, the entire string is returned.

Parameters

string: a string.

length: length of the new string.

StrMid

str StrMid (str string, int start, int length)

Creates a new string containing the length characters of string starting from the one at position start. If string does not exist or is shorter than start characters, an empty string is returned. If string is shorter than start + length characters, the entire substring beginning at start is returned.

Parameters

string: a string.

start: position at which the new string begins.

length: length of the new string.

StrParam

`int StrParam(item(s));`

Usage

StrParam will create a new string formatted based upon the same method for print or log. This string will only exist for 1 tic, so a delay will nullify it. As of revision r4295, strings generated by StrParam, GetCVarString and GetUserCVarString will last for as long as they need to, rather than for 1 tic.

This can be used to concatenate multiple ACS strings.

Return Value

The return value is the string table index of the new string.

Error.gif

Warning: This feature has at least one use case where the outcome is indeterminate. This feature can break demo and multiplayer sync if an indeterminate result is used to modify the playsim (anything that uses the random number generator, modify level geometry, spawn obstacles, monsters, or powerups, and so on). Usage of the feature in conjunction with non-playsim related features, such as displaying a HudMessage, is safe.

If string cast types l, k, or n are used, the resulting string may vary based on client settings. Typically this will not cause issues unless the result is combined with some other string operation and that is used to affect the playsim.

Example

```
str mapname = "hangar";
```

```
script 3 enter
```

```
{
// you can also assign it to a variable,
// but its contents are only valid until this tick ends if you are
// using a version prior to revision 4295
int keyObject = strparam(s:mapname, s:"key"); // would create "hangarkey"
if (CheckInventory(keyObject))
{
    TakeInventory(keyObject, 1);
}
else
{
    print(s:"You require ", s:keyObject);
    terminate; // after ending the script, you cannot expect the value in keyObject to point to a valid string anymore
               // on versions prior to r4295
}
ACS_Execute(120,0,10);
// ACS_Execute will not execute code during this tick.
// keyObject will not exist anymore when it starts running script 120.
// if you are using a version prior to r4295

delay(1); // now a tick will pass. any dynamic strings generated before this delay will be invalid once execution
resumes on
    // versions prior to r4295
/*
    no keyObject here on pre-r4295
*/
```


StrRight

str StrRight (str string, int length)

Creates a new string containing the length last characters of string. If string does not exist, an empty string is returned. If string is shorter than length characters, the entire string is returned.

Parameters

string: a string.

length: length of the new string.

ActivatorTID

int ActivatorTID (void)

Usage

Returns the TID of the actor that activated the script.

Return value

The TID of the actor that activated the script.

Examples

Can be used for a line that monsters can trigger, to see if a monster triggered it.

```
script 1 (void)
{
    if (ActivatorTID () == 999)
        Print (s:"You are not a zombie");
    else
        DamageThing (0); // kill it
}
```

```
script 10 ENTER
{
    Thing_ChangeTID (0, 999);
}
```

The ENTER script sets the player TIDs to 999 (as 0 usually implies the activator of the script). The script number 1 checks if the activator has a TID of 999, as in, is a player, and if so tells them so. Otherwise it kills them.

This could be helpful if you had scripted marines as your allies, and did not want them to be subjected to the same treatment as other monsters.

CanRaiseActor

```
bool CanRaiseActor (int tid);
```

Usage

Checks to see if the actor or actors with the specified tid are viable for resurrection. If tid is 0, the check is done on the activator of the script.

Parameters

tid: the tid of the actor or actors to check.

Return Value

The function returns true if the actor can be resurrected, otherwise it returns false. In case multiple actors are checked, the function only returns true if all of the actors can be resurrected. If at least one of them fails the check, the returned value is false.

CheckActorCeilingTexture

bool CheckActorCeilingTexture (int tid, str texture)

Usage

Returns true if the ceiling texture of the sector the calling actor is in matches the name in the string of the ACS function. If tid is 0, it uses the activator. For animated textures you only need to check for the base texture.

Parameters

tid: TID of the actor.

texture: Texture or flat to check for.

Return value

Returns true if the ceiling texture of the sector the calling actor is in matches the name in the string of the ACS function, false otherwise.

Examples

This script simply checks to see if the ceiling texture is using the "F_SKY1" texture and prints a message accordingly.

```
if (CheckActorCeilingTexture(0, "F_SKY1"))
    print (s:"You're outside!");
else
    print (s:"You're inside!");
```

CheckActorClass

bool CheckActorClass (int tid, str class)

Usage

Parameters

tid: TID of the actor. Use 0 to refer to the activator.

class: The name to which the actor's class name must be compared.

Return value

True if the class given is the same as that of the actor, false otherwise.

Examples

This example tells you if the owner of the tid passed to script 1 is a DoomImp or not.

```
script 1 (int tid)
{
    if (CheckActorClass(tid, "DoomImp"))
        print(s:"It's an Imp!");
    else
        print(s:"Not an Imp!");
}
```

You can set this script to an Actor Enters Sector thing to warn the players when a certain type of monster is approaching. Otherwise, you can specify a tid through direct execution. If a tid is not specified it is resolved as 0 or the tid of the activator (getting the activating actor in either case). The script cycles through the list of monster class names given in the monster_class array to find a match and prints a generic message for unlisted classes.

```
#define NUM_CLASSES    18

str monster_class[NUM_CLASSES] = {
    "Arachnotron", "Archvile", "BaronOfHell",
    "Cacodemon", "ChaingunGuy", "Cyberdemon",
    "Demon", "DoomImp", "Fatso",
    "HellKnight", "LostSoul", "PainElemental",
    "Revenant", "ShotgunGuy", "Spectre",
    "SpiderMastermind", "WolfensteinSS", "ZombieMan"
};
```

```
script 1 (int tid)
{
    int class_index = -1;

    if (!tid && ActivatorTID())
        tid = ActivatorTID();

    for (int i=0; i<NUM_CLASSES; i++)
        if (CheckActorClass(tid, monster_class[i]))
            class_index = i;

    if (class_index > -1)
        PrintBold(s:"Look out for the ", s:monster_class[class_index], s:"!");
    else
        PrintBold(s:"Look out for the... wait, what is that?");
}
```

This example is slightly more complicated (and fun) in that it prints a custom phrase based on the class of the monster.

```
#define MONST_CLASS_NAME 0
#define MONST_CLASS_MSG 1
#define NUM_CLASSES 18
```

```
str monster_msg[NUM_CLASSES][2] = {
    {"Arachnotron", "time to squash this spider!"},
    {"Archvile", "I haaaaaate Arch-Viles..."},
    {"BaronOfHell", "about 5 rockets oughta do the job."},
    {"Cacodemon", "there's something very... 'Manual of the Planes'-ish about it..."},
    {"ChaingunGuy", "even more annoying than the ShotgunGuy!"},
    {"Cyberdemon", "*ca-chunk* *ca-chunk* *ca-chunk* BLAM! AAAH! SPLAT!"},
    {"Demon", "watch as it runs to its death!"},
    {"DoomImp", "an Imp, a shell..."},
    {"Fatso", "LOLFATSO"},
    {"HellKnight", "the Baron's red headed stepchild."},
    {"LostSoul", "get the fly swatter!"},
    {"PainElemental", "he won't spit Lost Souls if he can't see you!"},
    {"Revenant", "time for a fist fight!"},
    {"ShotgunGuy", "easy to take out but annoying in numbers."},
    {"Spectre", "who does he think he's hiding from?"},
    {"SpiderMastermind", "did he say 'RURURU?'"},
    {"WolfensteinSS", "hey I didn't know he had a backside!"},
    {"ZombieMan", "anybody need clips?"}
};
```

```
script 1 (int tid)
{
    str prefix;
    int class_index = -1;

    if (!tid && ActivatorTID())
        tid = ActivatorTID();

    for (int i=0; i<NUM_CLASSES; i++)
        if (CheckActorClass(tid, monster_msg[i][MONST_CLASS_NAME]))
            class_index = i;

    switch (monster_msg[class_index][MONST_CLASS_NAME])
    {
        case "Arachnotron":
        case "Archvile":
            prefix = "It's an ";
            break;

        default:
            prefix = "It's a ";
            break;
    }

    if (class_index > -1)
        PrintBold(s:prefix, s:monster_msg[class_index][MONST_CLASS_NAME],
            s:", ", s:monster_msg[class_index][MONST_CLASS_MSG]);
    else
```

```
PrintBold(s:"Uh... what is it?");  
}
```

CheckActorFloorTexture

bool CheckActorFloorTexture (int tid, str texture)

Usage

Returns true if the floor texture of the sector the calling actor is in matches the name in the string of the ACS function. If tid is 0, it uses the activator. For animated textures you only need to check for the base texture.

Parameters

tid: TID of the actor.

texture: Texture or flat to check for.

Return value

Returns true if the floor texture of the sector the calling actor is in matches the name in the string of the ACS function, false otherwise.

Examples

This script simply checks to see if the floor texture is grass and prints a message accordingly.

```
if (CheckActorFloorTexture(0, "GRASS1"))
    Print (s:"You're on grass!");
else
    Print (s:"You're not on grass.!");
```


CheckActorProperty

bool CheckActorProperty (int tid, int property, int value)
bool CheckActorProperty (int tid, int property, float value)
bool CheckActorProperty (int tid, int property, str value)

Parameters

tid: TID of the actor. Use 0 to refer to the activator.

property: One of the properties listed below.

value: The value to which the property must be compared.

Actor Properties

APROP_ActiveSound stringSound played when the actor is walking around, as defined in SNDINFO.

APROP_Accuracy integer Accuracy of the actor.

APROP_Alpha fixed point Alpha of the actor. Range is [0.0, 1.0]

APROP_Ambush bool Whether the actor's AMBUSH flag is set or not.

APROP_AttackSound stringSound played when the actor attacks, as defined in SNDINFO.

APROP_AttackZOffset fixed point The attack z offset of the player.

APROP_ChaseGoal bool Walks to goal instead of target if a valid goal is set

APROP_Damage integer Actor's damage.

APROP_DamageFactor fixed point Generic damage factor for the actor. It is applied before any specific DamageFactor.

APROP_DamageMultiplier fixed point Generic damage multiplier for the actor. This is typically used on actors (e.g monsters) to strengthen or weaken the damage they deal from their attacks.

APROP_DamageType stringActor's damage type.

APROP_DeathSound stringSound played when the actor dies, as defined in SNDINFO.

APROP_Dormant bool Whether or not actor has the DORMANT flag.

APROP_Dropped bool Whether or not actor has the DROPPED flag. Dropped items are destroyed by closing doors and crushers while non-dropped items are not; and in games with the "Weapons Stay" option of DMFlags turned on, only weapons with the DROPPED flag set to 0 stay. By default, any item not placed originally on the map has the DROPPED flag set to 1.

APROP_Friction fixed point Actor's current friction factor.

APROP_Friendly bool Actor is friendly to the player and hostile to enemies. In addition to setting and clearing the FRIENDLY flag, when the property is set to true, the total kill count of the map is decreased by one for each affected actor, provided they are countable (only hostile monsters are countable), and is increased when the property is set to false.

APROP_FriendlySeeBlocks integer (New from 4.10.0)

The range in which friendly monsters or hostile monsters with SEEFRIENDLYMONSTERS on can see other non-player actors. This value is measured in increments of 128 map units, e.g a value of 32 equals a range of 4096 map units.

APROP_Frightened bool Monster runs away from player.

APROP_Gravity fixed point Current gravity factor of actor.

APROP_Health integer Actor's current health. Setting this property to 0 or less, kills the actor.

APROP_Height fixed point Actor's current height.

APROP_Invulnerable bool Actor will not lose any health.

APROP_JumpZ fixed point Player's jump height. The formula for jumping distance is $(\text{JumpZ} * 2) / 2 + \text{MaxStepHeight}$, and to get a specific JumpZ from a jumping height you want to achieve, use $\text{Sqrt}((\text{jump height} - \text{MaxStepHeight}) / 2) * 2$.

APROP_Mass integer Actor's mass.

APROP_MasterTID integer The TID of the actor linked to by the actor's master field.

APROP_MaxDropOffHeight fixed point Defines the maximum height of a step this actor can climb down when moving.

APROP_MaxStepHeight fixed point Defines the maximum height of a step this actor can climb up when moving.

APROP_MeleeRange fixed point Actor's melee range.

APROP_NameTag stringName of the actor. If the actor has not been explicitly named by the Tag property or in a script, its name is by default the same as its class name.

APROP_NoTrigger bool Whether or not actor has the NOTRIGGER flag.
 APROP_NoTarget bool Actor cannot be targeted by other monsters.
 APROP_PainSound stringSound played when the actor is injured, as defined in SNDINFO.
 APROP_Radius fixed point Actor's current radius.
 APROP_ReactionTime integer The time in tics a monster needs to attack back. However, the main use of this property is to serve as a counter for A_Countdown.
 APROP_RenderStyle integer How the actor is rendered:
 STYLE_None Do not draw
 STYLE_Normal Normal; just copy the image to the screen
 STYLE_Fuzzy Draw silhouette using "fuzz" effect
 STYLE_SoulTrans Draw translucent with amount in transsouls CVAR
 STYLE_OptFuzzy Draw as fuzzy or translucent, based on user preference
 STYLE_Stencil Draw as single color
 STYLE_AddStencil Draw as single additive color
 STYLE_AddShaded Treats 8-bit indexed images as an alpha map while applying additive translucency. Index 0 = fully transparent, index 255 = fully opaque.
 STYLE_Translucent Draw translucent
 STYLE_Add Draw additive
 STYLE_Shaded Treats 8-bit indexed images as an alpha map. Index 0 = fully transparent, index 255 = fully opaque.
 STYLE_TranslucentStencil Draw as single translucent color
 STYLE_Shadow Draw dark translucent stencil
 STYLE_Subtract Draw subtractive
 APROP_ScaleX fixed point Horizontal scaling of the actor's sprite. Does not affect collision box size.
 APROP_ScaleY fixed point Vertical scaling of the actor's sprite. Does not affect collision box size.
 APROP_Score integer A simple counter. Score items automatically increase it by their amount.
 APROP_SeeSound stringSound played when actor sees the player, as defined in SNDINFO.
 APROP_SoundClass stringSound class of the player.
 APROP_SpawnHealth integer The current max health of the actor. Only players may have their max health set this way. Note that for them the default value is 0, which is interpreted as "100 unless modified by DeHackEd". The Player.MaxHealth property can change the default value.
 APROP_Species stringSpecies the actor belongs to.
 APROP_Speed fixed point Actor's speed.
 For monsters, this is the distance they move every time A_Chase is called. For projectiles, this is the distance they move each tic. For players, this is multiplied by the player's class speed to determine the final speed the player will move for each tic that they have a movement key held down. Consequently, the standard APROP_Speed for a player is always 1.0, not what their actual speed is.

APROP_Stamina integer Stamina of the actor.
 APROP_StencilColor color Stencil color of the actor, as a hexadecimal value, e.g. 0xFFFFFFFF (white).
 APROP_TargetTID integer The TID of the actor linked to by the actor's target field.
 APROP_TracerTID integer The TID of the actor linked to by the actor's tracer field.
 APROP_ViewHeight fixed point The view height of the player.
 APROP_Waterlevel integer How "submerged" the actor is.
 0: Not submerged at all (e.g. standing on solid ground)
 1: Less than half submerged ("ankle deep")
 2: At least half submerged ("waist deep")
 3: Entirely submerged (completely underwater)
 APROP_WaterDepth (development version 6f4a29b only) fixed The exact depth at which the actor is submerged at in a water volume.
 Return value
 True if the value given is the same as that used by the actor, false otherwise. This is especially useful for string properties (such as sounds and species) which cannot be obtained by GetActorProperty (as of r4307, this is not the case, and this function can be used on string-based properties) as ACS does not handle dynamic strings (this is not the case starting with r4295).

Examples

This example checks the species of the thing with the specified tid and destroys it if it is a DoomImp.

```
script 1 (int tid)
{
  if (CheckActorProperty(tid, APROP_SPECIES, "DoomImp"))
  {
    if (ThingCountName("DoomImp", tid) > 1)
      print(s:"These Imps must die!");
    else
      print(s:"This Imp must die!");

    Thing_Destroy(tid, TRUE);
  }
}
```

CheckActorState

```
bool CheckActorState (int tid, str statename [, bool exact]);
```

Usage

Checks to see if the actor with the matching TID has the specified state. If tid is 0, the check is performed on the activator of the script.

Parameters

tid: The TID of the actor on which to perform the state check.

statename: The state to check for.

exact: Specifies whether or not partial state name matches are accepted (see SetActorState for more information and an example). Default is FALSE.

Return value

The function returns true if a matching state is found, otherwise it returns false.

CheckClass (ACS)

`bool CheckClass (str classname);`

Usage

Checks to see if the specified actor class is a valid class or not.

Parameters

`classname`: The name of the actor class to check.

Return value

The function returns true if the specified actor class is valid, otherwise it returns false.

Examples

The following checks to see if the actor class "CoolThing" exists and, if it does, it prints a message to confirm as much. If "CoolThing" does not exist, then a message is printed confirming that instead.

Script 100 OPEN

```
{
    if (CheckClass("CoolThing"))
    {
        Print(s:"The Cool Thing exists.");
    }
    else
    {
        Print(s:"The Cool Thing does not exist.");
    }
}
```

CheckFlag

`bool CheckFlag (int tid, str flag);`

Usage

Checks to see if the actor with the matching tid has the specified actor flag set. If tid is 0, the check is performed on the activator of the script.

Parameters

tid: The tid of the actor on which to perform the flag check.

flag: The actor flag to check. This should be a valid flag.

Return value

The function returns true if the actor has the specified flag set, otherwise it returns false.

Examples

Checks if the actor with a tid of 1 has the FLOAT flag.

```
script 1 (void)
{
    if(CheckFlag(1, "FLOAT") == TRUE)
    {
        Print(s:"This enemy can fly!");
    }
    else
    {
        Print(s:"This enemy is bound by the force of gravity.");
    }
}
```

CheckFont

`bool CheckFont (str fontname);`

Usage

Checks to see if the font passed to it is a valid font or not. Any type of font or graphic usable by SetFont will be considered valid here as well.

Parameters

fontname: The name of the font to check.

Return Value

Returns the true if the font is valid, otherwise it returns false.

CheckPlayerCamera

int CheckPlayerCamera (int player)

Usage

Returns the TID of the camera that the specified player is currently viewing. If the player doesn't exist (or has no camera somehow) or the camera is currently using any player as a viewpoint, then this returns -1.

Examples

This script deactivates all actors with a TID of 60 while the player is using a camera. This could be used to prevent enemies from attacking the player while he is doing so.

```
script 10 ENTER
{
    while (CheckPlayerCamera(PlayerNumber()) == -1) Delay(1);

    Thing_Deactivate(60);

    while (CheckPlayerCamera(PlayerNumber()) != -1) Delay(1);

    Thing_Activate(60);

    Restart;
}
```


CheckProximity

Jump to navigationJump to search

bool CheckProximity (int tid, str classname, float distance [, int count [, int flags [, int ptr]]]) - ACS

bool CheckProximity (class<Actor> classname, double distance[, int count[, int flags[, int ptr]]]) - ZScript

Usage

Performs a check to see if an actor of type classname is within distance of the actor specified by tid (ACS), or from the caller (ZScript), and checks to see the population within distance is greater or equal to count. These conditions can be modified with flags and will count any actor, as long as it is the same classname. Can be performed based upon the actor's pointer as well. This function is the ACS and ZScript version of A_CheckProximity and works in a similar manner.

Parameters

tid: The tid of the actor around which to do the distance check. Use 0 to refer to the activator. Only used in ACS.

classname: The name of the actor to check for.

distance: Determines how far this function should search for the actors. Must be greater than 0. It's a point value, so if you want to check within 48, you need to specify 48.0.

count: The minimum number of actors the function must find within distance in order to succeed. Default is 1.

flags: Can be combined using the '|' separator. Default is 0.

CPXF_ANCESTOR: Also matches classname against any base classes the actor has inherited.

CPXF_NOZ: Disables Z height checking -- the function will only care about the actor being in range for X and Y coordinates, regardless of how high or low an actor is.

CPXF_CHECKSIGHT: Restricts actor counting to only those which can be seen.

CPXF_SETTARGET: Sets the first actor matching classname as the calling actor's target.

CPXF_SETMASTER: Sets the first actor matching classname as the calling actor's master.

CPXF_SETTRACER: Sets the first actor matching classname as the calling actor's tracer.

Only one of the two following may be used due to mutual exclusion:

CPXF_COUNTDEAD: By default, the function only counts living actors. This causes the function to also accept dead ones. This only makes sense for monsters.

CPXF_DEADONLY: Inverses the function to only count dead monsters instead of living ones.

Only one of the two following may be used due to mutual exclusion:

CPXF_LESSEQUAL: The function will succeed if it finds the number of actors equivalent to count or less, instead of greater than or equal to.

CPXF_EXACT: The function will only succeed if the exact number is found defined by count.

The following flags rely on using one of the SET* flags above:

CPXF_FARTHEST: The actor gets the furthest classname actor within distance from the pointer's location.

CPXF_CLOSEST: The actor gets the closest classname actor within distance to the pointer's location.

CPXF_SETONPTR: The pointer exchange is set on the calling actor's pointer instead of itself.

Examples

This script checks every second if there is at least one imp around the player and informs them if there is.

Script "Shotgunguy_Checker" Enter

```
{
    if (CheckProximity(0, "DoomImp", 256.0, 1))
    {
        Print(s:"There's an imp somewhere around!");
    }
    else
    {
        Print(s:"There are no imps around.");
    }
}
```

```
}
```

```
Delay(35);
```

```
restart;
```

```
}
```

This script uses CheckProximity to respawn ammunition, but only if the player has already picked it up. It checks one time to see if there is a cell pack at least 4 map units away from a thing (e.g. a map spot) with a TID of 10. If there is not, it then spawns one along with a teleport fog.

```
Script "Spawn Cellpack" (void)
```

```
{
```

```
    if (!CheckProximity(10, "CellPack", 4.0))
```

```
    {
```

```
        SpawnSpot("CellPack", 10);
```

```
        SpawnSpot("TeleportFog", 10);
```

```
    }
```

```
}
```

CheckSight

bool CheckSight (int source, int dest, int flags)

Usage

This function performs a sight check to see if an actor is (or multiple actors are) within the line of sight of another actor (or other actors). This check is inclusive and returns positive if any actors with either tid have a line of sight between each other. 0 can be used in place of either tid and means the script activator.

Parameters

source

TID of the actor(s) doing the searching.

dest

TID of the actor(s) to search for.

flags

Flags to modify the search behavior. Currently supported flags are:

CSF_NOFAKEFLOORS: Ignores fake floors created by Transfer_Heights.

CSF_NOBLOCKALL: Ignores lines marked "Block Everything". Monsters can see through these lines if there's a chance that shooting them will make them unblock (like scripted breakable glass).

Return value

True if any of the specified actors have a line of sight, or false otherwise.

ClassifyActor

int ClassifyActor (int tid)

Usage

Returns information about the specified actor.

Parameters

tid: TID of the actor. Use 0 to refer to the script activator.

Return Value

The function usually returns a bitfield where each bit in the result value corresponds to a value defined in zcommon.acs. The exception is when no actor with the specified TID is found, in which case the return value is 0.

Currently defined values include:

ACTOR_NONE (0): Only returned if tid was non-zero, and there were no actors found with that TID.

ACTOR_WORLD (1): Only returned if tid was zero, but the activator is the world rather than an actor.

ACTOR_PLAYER (2): The actor is a player.

ACTOR_BOT (4): The actor is a bot.

ACTOR_VOODOODOLL (8): The actor is a "voodoo doll" . (An extra copy of a player present in the map that does not have any AI but passes damage taken to the corresponding player when injured.)

ACTOR_MONSTER (16): The actor is an enemy.

ACTOR_ALIVE (32): The actor is currently alive.

ACTOR_DEAD (64): The actor is currently dead.

ACTOR_MISSILE (128): The actor is a missile in flight.

ACTOR_GENERIC (256): The actor is neither a missile nor an enemy. (The actor may be a decoration, invisible marker such as a teleport destination, etc.)

Since the return value is a bitfield, the individual bits can be checked by using a bitwise-AND (&) comparison, as shown in the following examples.

Examples

This script does nothing if the activator is a monster.

```
script 1 (void)
```

```
{
    if (!(ClassifyActor(0) & ACTOR_MONSTER))
        Print(s:"Congratulations, you are not a monster!");
}
```

This script automatically exits when the actor it affects disappears or loses its TID. Note that it uses a direct comparison to check for ACTOR_NONE rather than a bitwise-AND.

```
script 7 (int target)
```

```
{
    while (ClassifyActor(target) != ACTOR_NONE)
        Delay(70);

    ThingSound(target, "misc/tracker", 127);
}
```

GameSkill

int GameSkill (void)

Usage

Returns the skill level of the current game.

Return value

The skill level of the current game. For script readability there are skill levels defined in zdefs.acs as follows:

SKILL_VERY_EASY = 0

"I'm Too Young to Die" in Doom.

SKILL_EASY = 1

"Hey, Not Too Rough" in Doom.

SKILL_NORMAL = 2

"Hurt Me Plenty" in Doom.

SKILL_HARD = 3

"Ultra-Violence" in Doom.

SKILL_VERY_HARD = 4

"Nightmare!" in Doom.

These are ordered 0 through 4 so all operators will work. For example, skills which are \leq SKILL_NORMAL are very easy, easy and normal.

Examples

This script takes the TID of a boss monster and applies different effects depending on the skill. Easier skills get a quarter health boss. Harder skills get a half health boss. Nightmare gets a full strength boss which is invisible. Although this example is somewhat unbalanced, this command can help to finely tune skill levels.

```
script 1 (int boss)
```

```
{
    if (GameSkill () <= SKILL_EASY)
        SetActorProperty (boss, APROP_HEALTH, 1000);
    else if (GameSkill () < SKILL_VERY_HARD)
        SetActorProperty (boss, APROP_HEALTH, 2000);
    else
        SetActorProperty (boss, APROP_RENDERSTYLE, STYLE_FUZZY);
}
```

This example increases the amount of Imps that get spawned at the MapSpot with the matching TID, starting with the base count provided, and adding the factor amount for every skill level beyond SKILL_VERY_EASY. The return value from SpawnSpotFacing is analyzed to determine the success of the spawn attempt.

```
script 1 (int tid, int base, int factor)
```

```
{
    int count = base + GameSkill() * factor;
    while (count)
    {
        if (SpawnSpotFacing("DoomImp", tid))
        {
            SpawnSpot("TeleportFog", tid);
            count--;
        }
        delay(35);
    }
}
```

GameType

int GameType (void)

Usage

Returns the game type currently being played.

Return value

The game type currently being played. For readability there are definitions defined in zdefs.acs as follows:

GAME_SINGLE_PLAYER = 0

Solo play.

GAME_NET_COOPERATIVE = 1

Cooperative net game.

GAME_NET_DEATHMATCH = 2

Deathmatch net game.

GAME_TITLE_MAP = 3

Title map

Examples

With this command it is possible to do things such as give an explanation about the map to the player.

```
script 1 ENTER
```

```
{
    if (GameType() != GAME_NET_DEATHMATCH)
        Print (s:"This is a deathmatch only map!");
    else
        Print (s:"BobDM1\nBy Bob");
}
```

Another use could be to unlock extra areas in GAME_NET_COOPERATIVE mode like in the example below, where all doors with a sector tag of 666 are opened, the level music is changed, and a message is printed if playing in co-op.

```
Script "OpenCooperativeAreas" OPEN
```

```
{
    If (GameType() == GAME_NET_COOPERATIVE)
    {
        Door_Open (666,16,0);
        SetMusic ("COOPMUSIC");
        PrintBold (s:"New areas have been unlocked in CO-OP mode!");
    }
}
```

GetActorAngle

fixed GetActorAngle (int tid)

Usage

Returns the actor's angle. If tid is 0, the function uses the activator.

Parameters

tid: TID of the actor.

Return value

The actor's angle as a fixed point angle.

Examples

This script will thrust an actor in the direction the player is facing when executed. This could be used to simulating pushing an actor away from you. The `>> 8` function is used to convert fixed point angles to byte angles (see Definitions for more information).

```
script 10 ENTER
```

```
{
    ThrustThing(GetActorAngle(0) >> 8, 50, 1, 0);
}
```

This script prints reversed actor's angle.

```
script 15 (int monsterid)
```

```
{
    //this will obtain current actor's angle and convert to byte angle.
    int angle = GetActorAngle(monsterid) >> 8;
```

```
    /*Looking at Byte Angle table we know that we can't turn an actor
    180 degrees with negative value*/
```

```
    //This function is easiest way to do it.
```

```
    if (angle < 128)
        angle = angle + 128;
    else angle = angle - 128;
```

```
    /*E.g. if actor is facing north, it's byte angle is 64, we want it's reverted angle.
    So 64 < 128 , 64 + 128 = 192 which is south.*/
```

```
    //prints reversed angle.
    Print(d:angle);
```

```
}
```

GetActorCeilingZ

fixed GetActorCeilingZ (int tid)

Usage

This returns the lowest ceiling point above the actor, as an absolute value.

Parameters

tid: TID of the actor.

Return value

The lowest ceiling point above the actor, as a fixed point value world coordinate.

Examples

This script reports the height of the player off the ceiling:

```
script 124 ENTER
```

```
{  
    while (TRUE)  
    {  
        Print (f:GetActorCeilingZ (0) - GetActorZ (0));  
        Delay (1);  
    }  
}
```


GetActorClass

str GetActorClass (int tid);

Usage

Retrieves the class name of an actor with the specified tid. If tid is 0, it retrieves the class name of the activator of the script.

Parameters

tid: The tid of the actor to retrieve its class name.

Return Value

Returns the class name of the actor as a string.

Examples

This is the same example from this function, only much simpler.

```
script 1 (int tid)
{
    PrintBold(s:"Look out for the ", s:GetActorClass(tid), s:"!");
}
```

GetActorFloorTerrain

str GetActorFloorTerrain (int tid)

Usage

Retrieves the floor's terrain name of the sector in which the actor with the matching TID is. If tid is 0, the function is run on the activator of the script.

Return value

The function returns the floor's terrain name as a string. If no actor with a matching TID is found, the return value is an empty string.

GetActorFloorTexture

str GetActorFloorTexture (int tid)

Usage

Retrieves the floor's texture name of the sector in which the actor with the matching TID is. If tid is 0, the function is run on the activator of the script.

Return value

The function returns the floor's texture name as a string. If no actor with a matching TID is found, the return value is an empty string.

GetActorFloorZ

fixed GetActorFloorZ (int tid)

Usage

This returns the highest floor point underneath the actor, as an absolute value.

The actor must be in the blockmap for this to be updated after spawn.

Parameters

tid: TID of the actor.

Return value

The highest floor point underneath the actor, as a fixed point value world coordinate.

Examples

This script reports the height of the player off the ground:

```
script 124 ENTER
{
    while (TRUE)
    {
        Print (f:GetActorZ (0) - GetActorFloorZ (0));
        Delay (1);
    }
}
```

GetActorLightLevel

int GetActorLightLevel (int tid)

Usage

Returns the light level of the sector the actor is currently in. If tid is 0, it returns the activator's light level. Note that this does not take into account point-based lights in GZDoom's OpenGL mode; it only returns the light level of the actual sector.

Examples

This script checks the light value of the sector the calling actor is in. If it is below 40, the actor will become fuzzy.

```
script 1 (int boss)
{
    if (GetActorLightLevel(0) < 40)
        SetActorProperty (boss, APROP_RENDERSTYLE, STYLE_FUZZY);
}
```

GetActorPitch

fixed GetActorPitch (int tid)

Usage

Returns the actor's pitch.

Parameters

tid: TID of the actor. Passing 0 will get the activator's pitch.

Return value

The actor's pitch is a fixed point angle. Due to the limits of freelook in the software renderer, this value is bounded by -0.0888977 (approximately -32°) and 0.155548 (approximately 56°), more precisely -5825 and 10194 as ints, but in GL freelook will go from -0.25 (-90°) to 0.25 ($+90^{\circ}$). Note that looking up produces a negative value and looking down produces a positive value. 0 is looking straight ahead.

Examples

This script will alter the trajectory of a DoomImpBall fired from a thing with a TID of 1 based on the pitch of the activator.

script 1 (void)

```
{
    int speed, vspeed;
    speed = cos(GetActorPitch(0)) * 64 >> 16;
    vspeed = -sin(GetActorPitch(0)) * 64 >> 16;
    SpawnProjectile(1, "DoomImpBall", 0, speed, vspeed, 1, 0);
}
```

GetActorPowerupTics

int GetActorPowerupTics (int tid, str powerup)

Usage

Retrieves the remaining duration in tics of the specified active powerup for the actor with the matching TID. If tid is 0, the function is run on the activator of the script.

Parameters

tid: The TID of the actor to perform the check on.

powerup: The class name of the powerup to get its remaining duration.

Return value

The function returns the number of tics remaining until the powerup's effect wears off. If the specified powerup class does not exist or it does not inherit from Powerup, the return value is 0. The function also returns 0 if there are no actors with a matching TID or the powerup does not currently exist in the actor's inventory, i.e. it is not currently active.

GetActorProperty

int GetActorProperty (int tid, int property)

Parameters

tid: TID of the actor. Use 0 to refer to the activator.

property: One of the properties listed below.

Actor Properties

APROP_Accuracy integer Accuracy of the actor.

APROP_Alpha fixed point Alpha of the actor. Range is [0.0, 1.0]

APROP_Ambush bool Whether the actor's AMBUSH flag is set or not.

APROP_AttackZOffset fixed point The attack z offset of the player.

APROP_ChaseGoal bool Walks to goal instead of target if a valid goal is set

APROP_Damage integer Actor's damage.

APROP_DamageFactor fixed point Generic damage factor for the actor. It is applied before any specific DamageFactor.

APROP_DamageMultiplier fixed point Generic damage multiplier for the actor. This is typically used on actors (e.g monsters) to strengthen or weaken the damage they deal from their attacks.

APROP_DamageType string Actor's damage type.

APROP_Dormant bool Whether or not actor has the DORMANT flag.

APROP_Dropped bool Whether or not actor has the DROPPED flag. Dropped items are destroyed by closing doors and crushers while non-dropped items are not; and in games with the "Weapons Stay" option of DMFlags turned on, only weapons with the DROPPED flag set to 0 stay. By default, any item not placed originally on the map has the DROPPED flag set to 1.

APROP_Friction fixed point Actor's current friction factor.

APROP_Friendly bool Actor is friendly to the player and hostile to enemies. In addition to setting and clearing the FRIENDLY flag, when the property is set to true, the total kill count of the map is decreased by one for each affected actor, provided they are countable (only hostile monsters are countable), and is increased when the property is set to false.

APROP_FriendlySeeBlocks integer (New from 4.10.0)

The range in which friendly monsters or hostile monsters with SEEFRIENDLYMONSTERS on can see other non-player actors. This value is measured in increments of 128 map units, e.g a value of 32 equals a range of 4096 map units.

APROP_Frightened bool Monster runs away from player.

APROP_Gravity fixed point Current gravity factor of actor.

APROP_Health integer Actor's current health. Setting this property to 0 or less, kills the actor.

APROP_Height fixed point Actor's current height.

APROP_Invulnerable bool Actor will not lose any health.

APROP_JumpZ fixed point Player's jump height. The formula for jumping distance is $(\text{JumpZ} * 2) / 2 + \text{MaxStepHeight}$, and to get a specific JumpZ from a jumping height you want to achieve, use $\text{Sqrt}((\text{jump height} - \text{MaxStepHeight}) / 2) * 2$.

APROP_Mass integer Actor's mass.

APROP_MasterTID integer The TID of the actor linked to by the actor's master field.

APROP_MaxDropOffHeight fixed point Defines the maximum height of a step this actor can climb down when moving.

APROP_MaxStepHeight fixed point Defines the maximum height of a step this actor can climb up when moving.

APROP_MeleeRange fixed point Actor's melee range.

APROP_NoTrigger bool Whether or not actor has the NOTRIGGER flag.

APROP_NoTarget bool Actor cannot be targeted by other monsters.

APROP_Radius fixed point Actor's current radius.

APROP_ReactionTime integer The time in tics a monster needs to attack back. However, the main use of this property is to serve as a counter for A_Countdown.

APROP_RenderStyle integer How the actor is rendered:

STYLE_None Do not draw

STYLE_Normal Normal; just copy the image to the screen
 STYLE_Fuzzy Draw silhouette using "fuzz" effect
 STYLE_SoulTrans Draw translucent with amount in transsouls CVAR
 STYLE_OptFuzzy Draw as fuzzy or translucent, based on user preference
 STYLE_Stencil Draw as single color
 STYLE_AddStencil Draw as single additive color
 STYLE_AddShaded Treats 8-bit indexed images as an alpha map while applying additive translucency.
 Index 0 = fully transparent, index 255 = fully opaque.
 STYLE_Translucent Draw translucent
 STYLE_Add Draw additive
 STYLE_Shaded Treats 8-bit indexed images as an alpha map. Index 0 = fully transparent, index 255 = fully opaque.
 STYLE_TranslucentStencil Draw as single translucent color
 STYLE_Shadow Draw dark translucent stencil
 STYLE_Subtract Draw subtractive
 APROP_ScaleX fixed point Horizontal scaling of the actor's sprite. Does not affect collision box size.
 APROP_ScaleY fixed point Vertical scaling of the actor's sprite. Does not affect collision box size.
 APROP_Score integer A simple counter. Score items automatically increase it by their amount.

APROP_SpawnHealth integer The current max health of the actor. Only players may have their max health set this way. Note that for them the default value is 0, which is interpreted as "100 unless modified by DeHackEd". The Player.MaxHealth property can change the default value.
 APROP_Speed fixed point Actor's speed.
 For monsters, this is the distance they move every time A_Chase is called. For projectiles, this is the distance they move each tic. For players, this is multiplied by the player's class speed to determine the final speed the player will move for each tic that they have a movement key held down. Consequently, the standard APROP_Speed for a player is always 1.0, not what their actual speed is.

APROP_Stamina integer Stamina of the actor.
 APROP_StencilColor color Stencil color of the actor, as a hexadecimal value, e.g. 0xFFFFFFFF (white).
 APROP_TargetTID integer The TID of the actor linked to by the actor's target field.
 APROP_TracerTID integer The TID of the actor linked to by the actor's tracer field.
 APROP_ViewHeight fixed point The view height of the player.
 APROP_Waterlevel integer How "submerged" the actor is.
 0: Not submerged at all (e.g. standing on solid ground)
 1: Less than half submerged ("ankle deep")
 2: At least half submerged ("waist deep")
 3: Entirely submerged (completely underwater)
 APROP_WaterDepth (development version 6f4a29b only) fixed The exact depth at which the actor is submerged at in a water volume.

Return value

The value of the specified property of the actor. Since ACS does not handle dynamic strings (this is not the case starting with r4295), this function cannot be used with certain properties and CheckActorProperty can be used instead (note that as of r4307, this is not the case, and this function can be used on string-based properties). It's important to note that when using APROP_Health, it returns the exact value of the current health, even if it's a negative value.

Examples

This script checks for a few different properties and prints helpful messages about them.

```

script 1 (int tid)
{
    if (GetActorProperty (tid, APROP_HEALTH) <= 25)
        print (s:"Thing ", d:tid, " has less than 26 health!!");

    if (GetActorProperty (12, APROP_RENDERSTYLE) == STYLE_OPTFUZZY)
        print (s:"Thing 12 is probably a spectre!");
  
```

}

GetActorRoll

fixed GetActorRoll (int tid)

Usage

Returns the roll angle of the actor with the matching tid. If tid is 0, it gets the roll angle of the activator of the script. In conjunction with SetActorRoll or ChangeActorRoll functions, an actor's roll angle can be adjusted relatively.

Parameters

tid: the tid of the actor.

Return value

The actor's roll angle as a fixed point angle.

GetActorVelX

fixed GetActorVelX (int tid)

Usage

This returns the velocity of the actor along the X axis. Positive values means eastward movement; negative values are westward.

Parameters

tid: TID of the actor.

Return value

The X velocity of the actor, as a fixed point value.

Examples

This example prints the angle that the player is moving in based on x and y velocity.

```
script 1 enter
{
    int angle;
    while (TRUE)
    {
        angle = VectorAngle(GetActorVelX(0), GetActorVelY(0));
        print(f:angle);
        delay(1);
    }
}
```

This example prints the current speed of the player, using the FixedSqrt function.

```
script 1 enter
{
    int x, y, z, speed;
    while (TRUE)
    {
        x = GetActorVelX(0);
        y = GetActorVelY(0);
        z = GetActorVelZ(0);
        speed = FixedMul(x, x) + FixedMul(y, y) + FixedMul(z, z);
        print(f:FixedSqrt(speed));
        delay(1);
    }
}
```

GetActorVelY

fixed GetActorVelY (int tid)

Usage

This returns the velocity of the actor along the Y axis. Positive values means northward movement; negative values are southward.

Parameters

tid: TID of the actor.

Return value

The Y velocity of the actor, as a fixed point value.

Examples

This example prints the angle that the player is moving in based on x and y velocity.

```
script 1 ENTER
{
  int angle;
  while (TRUE)
  {
    angle = VectorAngle(GetActorVelX(0), GetActorVelY(0));
    print(f:angle);
    delay(1);
  }
}
```

This example prints the current speed of the player, using the FixedSqrt function.

```
script 1 ENTER
{
  int x, y, z, speed;
  while (TRUE)
  {
    x = GetActorVelX(0);
    y = GetActorVelY(0);
    z = GetActorVelZ(0);
    speed = FixedMul(x, x) + FixedMul(y, y) + FixedMul(z, z);
    print(f:FixedSqrt(speed));
    delay(1);
  }
}
```

GetActorVelZ

fixed GetActorVelZ (int tid)

Usage

This returns the velocity of the actor along the Z axis. Positive values means upward movement; negative values are downward.

Parameters

tid: TID of the actor.

Return value

The Z velocity of the actor, as a fixed point value.

Examples

This example prints the current speed of the player, using the FixedSqrt function.

script 1 enter

```
{
  int x, y, z, speed;
  while (TRUE)
  {
    x = GetActorVelX(0);
    y = GetActorVelY(0);
    z = GetActorVelZ(0);
    speed = FixedMul(x, x) + FixedMul(y, y) + FixedMul(z, z);
    print(f:FixedSqrt(speed));
    delay(1);
  }
}
```

GetActorViewHeight

fixed GetActorViewHeight (int tid)

Usage

This returns the view height of the actor, as a fixed point value. For a player, this corresponds to the Player.ViewHeight property, modified by crouching if needs be. For other actors, it corresponds to the CameraHeight if one is defined, or defaults to half their height otherwise.

Parameters

tid: TID of the actor.

Return value

The view height of the actor, as a fixed point value.

Examples

This script will train a camera with a tid of 1 on the player and adjust its pitch to stay with the player even if crouched. Unfortunately it appears that as of version 2.4.1 this function does not possess the capacity to accept a tid of 0 and retrieve the activator's data.

```
script 1 (void)
```

```
{  
  int x, y, z, dist, angle, pitch;  
  ChangeCamera(1, 0, FALSE);  
  //To overcome GetActorViewHeight limitation...  
  if (!ActivatorTID())  
    Thing_ChangeTID(0, 1000 + PlayerNumber());
```

```
  while (TRUE)
```

```
  {  
    x = GetActorX(0) - GetActorX(1);  
    y = GetActorY(0) - GetActorY(1);  
    z = GetActorZ(0) + GetActorViewHeight(ActivatorTID()) - GetActorZ(1);  
    angle = VectorAngle(x, y);
```

```
    if ((angle + 0.125) % 0.5 > 0.25)
```

```
      dist = FixedDiv(y, sin(angle));
```

```
    else
```

```
      dist = FixedDiv(x, cos(angle));
```

```
    pitch = -VectorAngle(dist, z);
```

```
    SetActorAngle(1, angle);
```

```
    SetActorPitch(1, pitch);
```

```
    delay(1);
```

```
  }
```

```
}
```

GetActorX

fixed GetActorX (int tid)

Usage

This returns the X coordinate of the actor. If tid is 0, the function uses the activator.

Parameters

tid: TID of the actor.

Return value

The X coordinate of the actor, as a fixed point value world coordinate.

Examples

This is a semi-useful debug script

```
script 123 ENTER
```

```
{
    While (TRUE)
    {
        Print (f:GetActorX (0), s:", ", f:GetActorY (0));
        Delay (1);
    }
}
```

It creates a display of the player's in-game coordinates. (Though using the "idmpos" console command would give you that information on-screen nonetheless.)

This script uses Spawn to create a ring of imps around the player. It is called "Imp Surprise". The two parameters are the number of imps and their distance from the player. There are a couple of things to notice for this script: It requires quite a lot of open space to function correctly, and the space must be flat. A good place to use it would be in a large open room with a rather good pickup in the middle, assigning this script as the pickup's special.

```
script 1 (int count, int dist)
```

```
{
    int basex = GetActorX (0);
    int basey = GetActorY (0);
    int angle, n;

    for (n = 0; n < count; n++)
    {
        angle = 1.0 * n / count;

        Spawn(
            "DoomImp",
            basex + dist * cos (angle),
            basey + dist * sin (angle),
            GetActorZ (0), 0,
            (angle + 0.5) >> 8
        );
    }
}
```

This uses a little maths to place the imps in a circle around the player. It uses the player's X and Y coordinates as the center of the circle. It finds the angle from which each imp must be spawned using percentages, and then spawns it at that point by splitting up the angle in to X and Y offsets using sin and cos. It also sets the angle of each imp to be facing inwards.

The line of code `angle = 1.0 * n / count` uses a small and noteworthy trick. As the angle is measured as a fixed point angle, 1.0 is a full circle. Each successive n must be a fraction of the way around this circle, and the maximum is count. However, the line would not work if it were formed like this:

```
angle = n / count * 1.0;
```

This is because count is bigger than n, and due to integer division, `n / count` will always equal 0. This avoids the issues with integer division.

The line `(angle + 0.5) >> 8` reverses the angle of creation of the monster and turns it in to a byte angle. For example, if the monster is created at 45 degrees from the player, then it should be facing 225 degrees (towards the player). In fixed point angles, it is created at 0.125 and should be facing 0.625, as it has turned 180 degrees (0.5 of a circle). The `>> 8` converts from a fixed point angle to a byte angle.

GetActorY

fixed GetActorY (int tid)

Usage

This returns the Y coordinate of the actor. If tid is 0, the function uses the activator.

Parameters

tid: TID of the actor.

Return value

The Y coordinate of the actor, as a fixed point value world coordinate.

Examples

This is a semi-useful debug script

```
script 123 ENTER
```

```
{  
    while (TRUE)  
    {  
        Print (f:GetActorX (0), s:", ", f:GetActorY (0));  
        Delay (1);  
    }  
}
```

It creates a display of the player's in-game coordinates.

GetActorZ

Jump to navigationJump to search
fixed GetActorZ (int tid)

Usage

This returns the Z coordinate of the actor. If tid is 0, the function uses the activator. It will return the absolute height of the actor, not the height relative to the floor. To calculate the height of the floor, subtract the result of GetActorFloorZ from this value.

Parameters

tid: TID of the actor.

Return value

The Z coordinate of the actor as a fixed point value world coordinate.

Examples

This script rains health potions on the player from above for a while.

```
script 1 (int count)
```

```
{  
    while (count-- > 0)  
    {  
        Delay (random (5, 15));  
        Spawn ("HealthBonus",  
            GetActorX (0),  
            GetActorY (0),  
            GetActorZ (0) + 256.0, 0, 0);  
    }  
}
```

Be careful as Spawn will not work when the spawning position is not valid.

GetAirSupply

int GetAirSupply (int playernum)

Usage

Gives the amount of tics remaining in a player's air supply. This function targets only players since monsters do not have an air supply â€” drowning is not implemented for them.

Parameters

playernum: Player number. This information can be obtained through PlayerNumber.

Return value

The duration in tics before the player will start to drown. A negative value tells the player is already drowning. Note that a negative value of 0 is possible by coincidence if the script is run at just the right tic, but is also used to mean there are no player for the given number.

Examples

This example prints the number of seconds the player has until his or her air supply runs out. It then prints a message warning the player to return for air. Set this script to be executed by Eyes Go Below Surface and Eyes Go Above Surface actors placed in your 'deep water' sectors.

```
int print_air[8];
```

```
script 1 (void)
```

```
{
    print_air[PlayerNumber()]++;
    print_air[PlayerNumber()] %= 2;
    while (print_air[PlayerNumber()])
    {
        if (GetAirSupply(PlayerNumber()) > 0)
            print(i:GetAirSupply(PlayerNumber()) / 35);
        else
            print(s:"You'd better go up for air!");

        delay(1);
    }
}
```

GetAmmoCapacity (ACS)

int GetAmmoCapacity (str classname)

Usage

Returns the maximum amount of the specified ammo type one can carry. The return value is Inventory.MaxAmount or, if the player has picked up a backpack, Ammo.BackpackMaxAmount. All types derived from Inventory but not from Ammo will return 0.

Default values for standard ammo with and without backpack:

16/32: PhosphorusGrenadeRounds
20/40: PhoenixRodAmmo
25/50: PoisonBolts
30/60: HEGrenadeRounds
50/100: CrossbowAmmo, ElectricBolts, RocketAmmo, Shell
100/200: GoldWandAmmo, MiniMissiles
150/300: MaceAmmo
200/400: BlasterAmmo, Clip, SkullRodAmmo
250/500: ClipOfBullets
300/600: Cell
400/800: EnergyPod
200/200: Mana1, Mana2

Examples

An example of backpacks that can be stacked, as in, each backpack increases the max ammo a little:

```
bool playerbackpack[8] = {FALSE};
```

```
script 1 (void)
```

```
{
    if (playerbackpack[PlayerNumber()])
    {
        if (GetAmmoCapacity("Clip") < 800)
        {
            SetAmmoCapacity("Clip", GetAmmoCapacity("Clip") + 100);
            SetAmmoCapacity("Shell", GetAmmoCapacity("Shell") + 25);
            SetAmmoCapacity("RocketAmmo", GetAmmoCapacity("RocketAmmo") + 25);
            SetAmmoCapacity("Cell", GetAmmoCapacity("Cell") + 75);
        }
    }
    else
        playerbackpack[PlayerNumber()] = TRUE;
}
```

```
script 998 ENTER
```

```
{
    if (GetAmmoCapacity("Clip") > 200)
        playerbackpack[PlayerNumber()] = TRUE;
}
```

```
script 999 RESPAWN
```

```
{
    playerbackpack[PlayerNumber()] = FALSE;
}
```

As the backpack will force a change in the amount of ammo the player has the first time it is picked up, this script records the whether each player has picked up a backpack already. There are eight potential players

and by default each has their variable set to FALSE. There are two exceptions, which are handled by scripts 998 and 999. 998 checks if the player has a backpack when entering the level by testing their capacity for bullets. 999 resets the player's variable should they die and therefore lose the backpack.

The actual stacking script is number 1, and should be set as the thing special of every backpack. That is, the backpack should have special number 80 (ACS_Execute) with parameters (1, 0, 0, 0, 0). The script checks if the player has already picked up a backpack previously to this, and if so, starts incrementing each type of ammo a little. It does this by first checking if the ammos have reached the absolute maximum amount. If not, it adds a little bit on to each. Note that it only checks if "Clip" has reached the maximum amount, but as each value is incremented at the same time, they will all hit their limits at the same time as the "Clip" type. If the player has not picked up a backpack before, they have now, so their variable is set to TRUE.

GetArmorInfo

str GetArmorInfo (int infotype)

int GetArmorInfo (int infotype)

fixed GetArmorInfo (int infotype)

Usage

Retrieves the value of the specified infotype for the currently equipped armor. This function works on players only.

Parameters

infotype: the armor information to retrieve its value:

ARMORINFO_CLASSNAME

The class name of the armor. If the player has no armor, the class name will be None.

ARMORINFO_SAVEAMOUNT

The save amount of the armor if the equipped armor is of the BasicArmorPickup type, the max save amount if it is of the BasicArmorBonus type. Do note, however, that if the player is already equipped with some sort of armor, this value could change upon picking up BasicArmorBonus-derived items whose max save amount is higher than whatever value is currently stored for this info type; the higher of the two values will be the one used, and thus returned.

ARMORINFO_ACTUALSAVEAMOUNT

The save amount of the armor if the equipped armor is of the BasicArmorPickup type, the max save amount if it is of the BasicArmorBonus type. Unlike SAVEAMOUNT above, picking up BasicArmorBonus-derived items on top of the current armor does not alter the value returned for this info type in any way.

ARMORINFO_SAVEPERCENT

The save percent of the armor.

ARMORINFO_MAXABSORB

The max absorb of the armor.

ARMORINFO_MAXFULLABSORB

The max full absorb of the armor.

Return value

Returns the stored value of the specified infotype. For CLASSNAME, the value is returned as a string; for SAVEAMOUNT, ACTUALSAVEAMOUNT, MAXABSORB and MAXFULLABSORB, it is returned as an integer; and for SAVEPERCENT, it is returned as a fixed point value.

GetArmorType

int GetArmorType (string armortype, int playernum)

Returns true if the player's armor type matches the first parameter.

Parameters

armortype: The class name of an armor type. This must be either "None", the name of a BasicArmorPickup or that of a BasicArmorBonus.

playernum: Player number. This information can be obtained through PlayerNumber.

This function targets only players. It concerns only BasicArmor and therefore does not cover HexenArmor.

Return value

The return value is the number of armor points if the player wears the designated armor, 0 otherwise.

Examples

This script constantly informs the first player which kind of Doom armor he is wearing.

```
#define ARMORTYPES4
```

```
str armor_types[ARMORTYPES] = {  
    "ArmorBonus",  
    "GreenArmor",  
    "BlueArmor",  
    "BlueArmorForMegasphere"  
};
```

```
str armor_messages[6] = {  
    "You are unarmored, but have found armor bonuses.",  
    "You are wearing security armor.",  
    "You are wearing combat armor.",  
    "You have a megasphere.",  
    "What the hell are you wearing?",  
    "You are unarmored."  
};
```

```
function void PrintArmorType (void)
```

```
{  
    bool found_armor;  
  
    if (CheckInventory("Armor")) // If player has at least 1 armor point, proceed to check the type...  
    {  
        for (int i; i<ARMORTYPES; i++)  
        {  
            if (GetArmorType(armor_types[i], PlayerNumber()))  
            {  
                found_armor = TRUE;  
                break;  
            }  
        }  
    }  
  
    if (found_armor)  
    {
```



```
    HudMessage(s:armor_messages[i]; HUDMSG_PLAIN, 1, CR_RED, 0.1, 0.9, 1.0);
}
else
{
    HudMessage(s:armor_messages[4]; HUDMSG_PLAIN, 1, CR_RED, 0.1, 0.9, 1.0);
}
}
else // ... otherwise, print the "unarmored" message
{
    HudMessage(s:armor_messages[5]; HUDMSG_PLAIN, 1, CR_RED, 0.1, 0.9, 1.0);
}
}
```

script 1 ENTER

```
{
while (TRUE)
{
    PrintArmorType();
    delay(35);
}
}
```

GetChar

int GetChar (str string, int index)

Gets the character in a string at index.

Parameters

string: a string.

index: index of the character in the string (starting at 0).

Examples

Here are some examples of how to use GetChar.

```
GetChar("abcde", 0); //this returns 'a'
```

```
GetChar("abcde", 2); //this returns 'c'
```

```
GetChar("abcde", 4); //this returns 'e'
```

This example prints a message on screen one character at a time in a randomized pattern. The text is displayed for one second and then removed in a similar fashion. Please note the use of the cast type c.

```
#define NUM_CHARS 6 //the length of str text
```

```
str text = "zdoom!"; //set NUM_CHARS to length
```

```
bool pos[NUM_CHARS];
```

```
script 1 (void)
```

```
{
    int i, j, x;
    SetHudSize(640, 480, 1);
    for (i=0; i<NUM_CHARS; i++)
    {
        do {
            j = random(0, NUM_CHARS - 1);
        } while (pos[j]);

        pos[j] = TRUE;
        x = 320.0 - NUM_CHARS * 10.0 / 2 + j * 10.0;
        HudMessage(c:GetChar(text, j); HUDMSG_PLAIN, j + 1, CR_RED, x, 240.0, 0);
        delay(5);
    }
}
```

```
delay(35);
for (i=0; i<NUM_CHARS; i++)
{
    do {
        j = random(0, NUM_CHARS - 1);
    } while (!pos[j]);

    pos[j] = FALSE;
    HudMessage(c:' '; 0, j + 1, 0, 0, 0, 0);
    delay(5);
}
}
```

Alternatively, you can define the pos boolean array size high, and use StrLen to set the number of loop iterations. This

method is more flexible provided you don't exceed the array size.

```
str text = "zomg it's zdoom!";  
bool pos[255];
```

```
script 1 (void)  
{  
    int i, j, x;  
    SetHudSize(640, 480, 1);  
    for (i=0; i<StrLen(text); i++)  
    {  
        do {  
            j = random(0, StrLen(text) - 1);  
        } while (pos[j]);  
  
        pos[j] = TRUE;  
        x = 320.0 - StrLen(text) * 10.0 / 2 + j * 10.0;  
        HudMessage(c:GetChar(text, j); HUDMSG_PLAIN, j + 1, CR_RED, x, 240.0, 0);  
  
        if (GetChar(text, j) != ' ') //don't delay for blank spaces  
            delay(5);  
    }  
  
    delay(35 + StrLen(text) * 2); //one second and 2 tics per character  
  
    for (i=0; i<StrLen(text); i++)  
    {  
        do {  
            j = random(0, StrLen(text) - 1);  
        } while (!pos[j]);  
  
        pos[j] = FALSE;  
        HudMessage(c:' '; 0, j + 1, 0, 0, 0, 0);  
  
        if (GetChar(text, j) != ' ') //don't delay for blank spaces  
            delay(5);  
    }  
}
```

GetCVar (ACS)

(This is for the ACS function. For the DECORATE function please [click here](#); for the ZScript function please [click here](#).)

Warning: This feature has at least one use case where the outcome is indeterminate. This feature can break demo and multiplayer sync if an indeterminate result is used to modify the playsim (anything that uses the random number generator, modify level geometry, spawn obstacles, monsters, or powerups, and so on). Usage of the feature in conjunction with non-playsim related features, such as displaying a HudMessage, is safe.

Using non-playsim CVARs such as screenblocks or invertmouse will cause indeterminate behavior. Using Server or User variables, either internal or defined via CVARINFO, is however perfectly safe.

```
int GetCVar (str cvar)
```

Usage

Returns the value of a particular cvar.

Parameters

cvar: name of a console variable to get the value from.

Return value

The value of the specified console variable. This is only useful for cvars that can be represented as integers. Also note that you can create your own console variables by using the CVARINFO lump.

If the console variable is a user CVAR, it will check the activator of the script. To check the user CVAR of a specific player, you can use GetUserCVar.

If no such console variable exists, it will return 0.

Examples

This script will spawn Imps only if sv_nomonsters is false:

```
if (!GetCVar ("sv_nomonsters"))
{
    SpawnSpot ("DoomImp", 66);
    SpawnSpot ("TeleportFog", 66);
}
```

One possible use for this is if you are making additions to the HUD, you can check if the player is using the status bar or not and act accordingly:

```
int screenblocks = GetCVar ("screenblocks");
```

```
if (screenblocks < 10)
{
    // The status bar is visible
}
else if (screenblocks == 10)
{
    // The fullscreen HUD is visible instead
}
else
{
    // No HUD of any sort is visible
}
```


GetCVarString (ACS)

Warning: This feature has at least one use case where the outcome is indeterminate. This feature can break demo and multiplayer sync if an indeterminate result is used to modify the playsim (anything that uses the random number generator, modify level geometry, spawn obstacles, monsters, or powerups, and so on). Usage of the feature in conjunction with non-playsim related features, such as displaying a HudMessage, is safe.

Using non-playsim CVARs such as screenblocks or invertmouse will cause indeterminate behavior. Using Server or User variables, either internal or defined via CVARINFO, is however perfectly safe.

string GetCVarString (str cvar)

Usage

Returns the value of a particular cvar.

Parameters

cvar: name of a console variable to get the value from.

Return value

(Verification needed)

The value of the specified console variable. For numerical values, see GetCVar instead. Also note that you can create your own console variables by using the CVARINFO lump.

If the console variable is a user CVAR, it will check the activator of the script. To check the user CVAR of a specific player, you can use GetUserCVarString.

If no such console variable exists, it will return an empty string.

Examples

script 42 (void)

```
{
    Print(s:"Your name is... ", s:GetCVarString("name"), s:"!");
    //Prints your name configured in the Player settings menu on the screen.
}
```

GetLevelInfo

int GetLevelInfo (int levelinfo)

Usage

This function provides you with properties of the current map. If you want to find out the level time, use Timer.

Parameters

levelinfo

The level information to get:

LEVELINFO_PAR_TIME

Par time of the map.

LEVELINFO_SUCK_TIME

Suck time of the map.

LEVELINFO_CLUSTERNUM

Number of the cluster to which the map belongs.

LEVELINFO_LEVELNUM

Number of the map. (See LevelNum map property)

LEVELINFO_TOTAL_SECRETS

Number of total secrets in the map.

LEVELINFO_FOUND_SECRETS

Number of revealed secrets.

LEVELINFO_TOTAL_ITEMS

Number of total countable items in the map.

LEVELINFO_FOUND_ITEMS

Number of picked up countable items.

LEVELINFO_TOTAL_MONSTERS

Number of total countable monsters in the map.

LEVELINFO_KILLED_MONSTERS

Number of killed monsters.

Return value

Returns value of the specified level property or 0 when the property is unknown.

Examples

This script reports on the player's progress on killing monsters. Similar to the kill count on the automap.

script 2 (void)

```
{
    int mtotal = GetLevelInfo (LEVELINFO_TOTAL_MONSTERS),
        mkilled = GetLevelInfo (LEVELINFO_KILLED_MONSTERS);

    if (mkilled == mtotal)
    {
        PrintBold (s:"You have killed all the monsters!");
    }
    else
    {
        PrintBold (s:"You have killed ", d:mkilled, s:" monsters!\n",
            d:mtotal-mkilled, s:" left to go!");
    }
}
```

GetLineActivation

int GetLineActivation (int lineid);

Usage

Retrieves the line activation flags of the line with the specified line ID.

Parameters

lineid: The ID of the line of which to get the activation flags.

Return value

The return value of the function is a bitfield where each bit in the result value corresponds to an activation flag value:

SPAC_None (0) — No flags.

SPAC_Cross (1) — Activated when crossed by player.

SPAC_Use (2) — Activated when used by player.

SPAC_MCross (4) — Activated when crossed by monster.

SPAC_Impact (8) — Activated when hit by projectile.

SPAC_Push (16) — Activated when bumped by player.

SPAC_PCross (32) — Activated when crossed by projectile.

SPAC_UseThrough (64) — Activated when used by player (with pass through).

SPAC_AnyCross (128) — Activated by anything crossing it which does not have the TELEPORT flag.

SPAC_MUse (256) — Activated by monsters using it.

SPAC_MPush (512) — Activated by monsters bumping into it.

SPAC_UseBack (1024) — The line can be used from the back side.

Since the return value is a bitfield, the individual bits can be checked by using a bitwise AND (&) comparison.

More than one flag can be checked by using the bitwise OR (|) comparison.

Examples

The following script will print a message on the screen if the tagged line can be activated by "using" it:

Script 1 (int id)

```
{
    If(GetLineActivation(id) & SPAC_USE)
    {
        Print(s:"You can open this door by 'using' it!");
    }
}
```

This script will print a message if the tagged line can be activated by "using" it or bumping into it:

Script 2 (int id)

```
{
    If(GetLineActivation(id) & (SPAC_USE | SPAC_PUSH))
    {
        Print(s:"You can open this door by 'using' or bumping into it!");
    }
}
```


GetLineRowOffset

int GetLineRowOffset (void)

Usage

This command returns the Y component of the alignment of the texture on the front side of the activating line. There is no similar command to find the X component.

Return value

The Y component of the alignment of the texture on the front side of the activating line.

Examples

Attaching this script to lines will show their offset.

```
script 1 (void)
{
    Print (d:GetLineRowOffset ());
}
```

GetPlayerInfo

int GetPlayerInfo (int playernumber, int playerinfo)

Usage

Retrieves player-related information. To get information for the player who activated the script, use the PlayerNumber function.

Parameters

playernumber

The player to get the information from.

playerinfo

One of:

PLAYERINFO_TEAM

Which team the player is on. No team is 255 always. Without use of the TEAMINFO lump the teams are: 0 for blue, 1 for red, 2 for green, 3 for gold, 4 for black, 5 for white, 6 for orange or 7 for purple.

PLAYERINFO_AIMDIST

How far the player autoaims.

PLAYERINFO_COLOR

The player's color, as 0xRRGGBB in hexadecimal.

PLAYERINFO_GENDER

The player's gender: 0 for male, 1 for female, and 2 for other.

PLAYERINFO_NEVERSWITCH

The player's neverswitchonpickup setting.

PLAYERINFO_MOVEBOB

The player's movebob setting.

PLAYERINFO_STILLBOB

The player's stillbob setting.

PLAYERINFO_FVIEWBOB (development version 0fdb740 only)

The player's fviewbob setting.

PLAYERINFO_PLAYERCLASS

A number representing the player's class. In Hexen, this is 0 for the fighter, 1 for the cleric, and 2 for the mage. Note that this is the player class the player has selected (playerclass cvar), not necessarily the one the player is currently playing with - to get the current player class number, use PlayerClass.

PLAYERINFO_FOV

The player's current FOV.

PLAYERINFO_DESIREDFOV

The player's fov setting.

Return value

Returns the value of the given property of the specified player. If you ask for information about a player who is not in the game, it will return -1. If you ask for an unknown information it will return 0.

Examples

This script opens a door only if the player is Female:

```
script 1 (void)
```

```
{
    if(GetPlayerInfo(PlayerNumber(), PLAYERINFO_GENDER) == 1)
        Door_Open(1, 20);
    else
        Print(s:"sorry dude, ladies only");
}
```

GetPlayerInput

int GetPlayerInput (int player, int input)

Usage

Returns information regarding the keyboard, mouse and joystick input of the specified player at this exact moment or the previous game tic. This can be used to create effects such as key pads or on-screen mouse cursors in your maps, or simply to act on player input directly instead of relying on other methods of determining the player's actions.

Note that this function does not read the player's input devices directly. Instead, it will only report information about which binds the player is using. For example, you cannot check to see if the player is pressing `W`, but you can check to see if they are using `+forward`. This design is intended to prevent abuse, as well as making the function more adaptable (since here the player can freely rebind `+forward` and still have the script work for them).

Parameters

player

The number of the player you want to get information on. Player 1 is 0, player 2 is 1, etc. Use -1 to specify the script activator instead.

input

The player input you wish to check. To use, specify one of the following flags. All flags come in two varieties; The `INPUT_*` series checks the player's raw input, e.g. the keys they are actually pressing. The `MODINPUT_*` series check the values after they have been processed by the game engine. These may be different if, for example, the player is fully or partially frozen (Movement inputs will be nulled) or is using a weapon such as the chainsaw which alters the player's input to include forward movement.

INPUT_BUTTONS

MODINPUT_BUTTONS

The movement commands that are currently active. See the table below for instructions on how to read individual controls.

INPUT_OLDBUTTONS

MODINPUT_OLDBUTTONS

The movement commands that were active during the previous tic. By comparing this value with `INPUT_BUTTONS`, you can determine the exact moment when the player presses or releases a key.

INPUT_PITCH

MODINPUT_PITCH

The pitch movement of the player. Looking up returns positive values, down returns negative. Maximum value is 32767.

INPUT_YAW

MODINPUT_YAW

The yaw motion of the player. Turning left returns positive values, right returns negative. Maximum value is 32767.

INPUT_ROLL

MODINPUT_ROLL

Not currently used.

INPUT_FORWARDMOVE

MODINPUT_FORWARDMOVE

The forward/backward movement of the player. Forward is positive, reverse is negative. Maximum positive value is 12800, same with negative except negative; walking speed is half that (6400). This is useful to compare to if the intent is to see if the player is walking or running and at specific player-intended speeds if a joystick axis is used.

INPUT_SIDEMOVE

MODINPUT_SIDEMOVE

The side-to-side (strafing) movement of the player. Left is negative, right is positive. Maximum positive value is 10240. When walking, this becomes 6144.

INPUT_UPMOVE

MODINPUT_UPMOVE

The up or down movement of the player (flying or swimming). Upwards is positive, down is negative. Note that these only return the input being received by ZDoom from the specified player, and do not correlate to the actual movement (if any) of the player. In other words, INPUT_SIDEMOVE may be used to measure the amount of force a player is putting on an analog axis (such as a joystick or mouse) bound to the strafe inputs. Therefore, if the player is stuck in a small room and cannot move any further in the specified direction, this function will still return the value that the player is trying to make the player move, not the actual player's end movement.

Return value

If input is one of the *BUTTONS flags, the return value is a bitmask representing the inputs that the specified player is currently using from the given category. Otherwise, the return value is a scalar value representing the amount of input being applied to the specified control. If the specified player is not in the game, the return value is always 0.

Reading buttons

Note: Currently, joystick or mouse axes bound to movement will not set the corresponding button for the purposes of reading by this function. (E.g. moving forward by pressing forward on an analog joystick will not set the BT_FORWARD key.) However, pressing a keyboard key does set the axis inputs. Therefore, it is recommended that mod authors read the axes above whenever possible, instead of relying on [MOD]INPUT_[OLD]BUTTONS to read directional movement.

To properly use GetPlayerInput for button presses, you will need to check the result value against a set of defined controls to determine which ones the player is using. You can do this by using the bitwise AND operator (&) in your code, as in this example:

```
int buttons = GetPlayerInput(-1, INPUT_BUTTONS);
```

```
if (buttons & BT_FORWARD)
{
    print(s:"You are pressing the move forward key.");
}
```

The bitwise OR operator (|) can be used to check for multiple buttons.

```
int buttons = GetPlayerInput(-1, INPUT_BUTTONS);
```

```
if (buttons & (BT_USE|BT_ATTACK))
{
    print(s:"You could be pressing the use key, or the attack key.");
}
```

The following buttons are supported:

ACS Definition	Corresponding action
BT_FORWARD	Walk forward
BT_BACK	Walk backward
BT_LEFT	Turn left
BT_RIGHT	Turn right
BT_MOVELEFT	Strafe left
BT_MOVERIGHT	Strafe right
BT_ATTACK	Fire primary
BT_ALTATTACK	Fire secondary
BT_USE	Use/Open
BT_JUMP	Jump
BT_CROUCH	Crouch
BT_TURN180	180-degree turn
BT_RELOAD	Reload weapon
BT_ZOOM	Zoom weapon
BT_SPEED	Run/walk modifier

BT_RUN Run/walk state
 BT_STRAFE Strafe modifier
 BT_LOOKUP Look up (Keyboard)
 BT_LOOKDOWN Look down (Keyboard)
 BT_MOVEUP Swim/fly upward
 BT_MOVEDOWN Swim/fly downward
 BT_SHOWSCORES Show multiplayer scoreboard
 BT_USER1 User-defined button 1
 BT_USER2 User-defined button 2
 BT_USER3 User-defined button 3
 BT_USER4 User-defined button 4

The four user-defined buttons (+user1, -user1, +user2, -user2 etc) can now be used for weapons, and have been included for usage with this function so that mod authors can implement up to four custom inputs which the player can bind in the controls menu.

BT_RUN and BT_SPEED

BT_RUN reflects the running/walking state, not merely whether the speed button is pressed or not like the case with BT_SPEED. This distinction becomes evident when autorun is involved.

Action	State	BT_SPEED	BT_RUN
Speed button not pressed; autorun disabled	Walking	Not set	Not set
Speed button pressed; autorun disabled	Running	Set	Set
Speed button not pressed; autorun enabled	Running	Not set	Set
Speed button pressed; autorun enabled	Walking	Set	Not set

Examples

This simple example prints a message if the player presses both the forward and back buttons at the same time, thereby canceling his forward or backward movement.

```
script 1 ENTER
{
  int buttons;

  while (TRUE)
  {
    buttons = GetPlayerInput(-1, INPUT_BUTTONS);

    if (buttons & BT_FORWARD && buttons & BT_BACK)
    {
      print(s:"Are you coming or going?");
    }

    delay(1);
  }
}
```

This example demonstrates how to make a simple three digit combination lock. In this case, there are three locks. The lock combination that is used is specified by passing a value between 0 (the first) and 2 (the third) in the first argument. When adding or subtracting locks, be sure to define the NUM_LOCKS constant and lock_code array accordingly. Also, be sure to adjust the number of cases in the switch statement at the end of the script as well.

```
#define NUM_LOCKS 3

int lock_code[NUM_LOCKS][3] = {{6, 9, 1}, {3, 2, 5}, {1, 7, 3}};
int digit_pick[3];
```

```

bool lock_picked[NUM_LOCKS];

script 1 (int this_lock)
{
    if (!lock_picked[this_lock])
    {
        int buttons, count, match, quit, wait;

        SetHudSize(640, 480, 1);
        HudMessage(s:"Forward and back change digits, use to pick, and fire to exit.";
            HUDMSG_PLAIN, 1, CR_WHITE, 0.1, 30.1, 0.0);
        SetPlayerProperty(0, ON, PROP_TOTALLYFROZEN);

        while (count < 3 && !quit)
        {
            buttons = GetPlayerInput(-1, INPUT_BUTTONS);

            if (buttons & BT_ATTACK)
            {
                quit = 1;
            }
            else if (buttons & BT_FORWARD)
            {
                digit_pick[count]++;
                digit_pick[count] %= 10;
            }
            else if (buttons & BT_BACK)
            {
                digit_pick[count] -= 9;
                digit_pick[count] %= 10;
            }
            else if (buttons & BT_USE)
            {
                count++;
            }
        }

        if (count < 3)
        {
            HudMessage(i:digit_pick[count];
                HUDMSG_PLAIN, 2+count, CR_RED, 305.0+count*10.0, 240.0, 0);
            do
            {
                delay(1);
                wait++;
            }
            while (GetPlayerInput(-1, INPUT_BUTTONS) == buttons && wait < 5);
            wait = 0;
        }
    }
}

if (!quit)
{
    for (int i=0; i<3; i++)
    {

```

```

    if (lock_code[this_lock][i] == digit_pick[i])
    {
        match++;
    }
    digit_pick[i] = 0;
}

if (match == 3)
{
    HudMessage(s:"Lock combination accepted."; HUDMSG_FADEOUT, 5, CR_GREEN, 320.0, 200.0, 3.0, 1.0);
    lock_picked[this_lock] = 1;

    switch (this_lock)
    {
    case 0:
        //action for lock 1
        break;
    case 1:
        //action for lock 2
        break;
    case 2:
        //action for lock 3
        break;
    }
}
else
{
    HudMessage(s:"Invalid lock combination."; HUDMSG_FADEOUT, 5, CR_RED, 320.0, 200.0, 3.0, 1.0);
}
}

HudMessage(s:""; 0, 1, 0, 0, 0, 0);
HudMessage(s:""; 0, 2, 0, 0, 0, 0);
HudMessage(s:""; 0, 3, 0, 0, 0, 0);
HudMessage(s:""; 0, 4, 0, 0, 0, 0);
SetPlayerProperty(0, OFF, PROP_TOTALLYFROZEN);
}
}

```

Important note

There have been recommendations to use '==' and '!=' to check for a single input bit being set. Please note that this advice is wrong and should not be followed! Always use '&' and '|' with the appropriate bits being checked.

GetPolyobjX

fixed GetPolyobjX (int po)

Usage

This returns the X coordinate of the polyobject.

Parameters

po: Number of the polyobject.

Return value

The X coordinate of the polyobject's start point, as a fixed point value world coordinate. A value of 0x7FFFFFFF (32767 when converted from integer to fixed point) means that the polyobject in question does not exist.

GetPolyobjY

fixed GetPolyobjY (int po)

Usage

This returns the Y coordinate of the polyobject.

Parameters

po: Number of the polyobject.

Return value

The Y coordinate of the polyobject's start point, as a fixed point value world coordinate. A value of 0x7FFFFFFF (32767 when converted from integer to fixed point) means that the polyobject in question does not exist.

GetScreenHeight

Warning: This feature has at least one use case where the outcome is indeterminate. This feature can break demo and multiplayer sync if an indeterminate result is used to modify the playsim (anything that uses the random number generator, modify level geometry, spawn obstacles, monsters, or powerups, and so on). Usage of the feature in conjunction with non-playsim related features, such as displaying a HudMessage, is safe.

GetScreenHeight only knows the resolution of the local session. Information about the resolution of other players in a game or demo does not exist.

int GetScreenHeight (void)

Usage

Returns the vertical resolution. For example, if you are playing in 800x600 it will return 600. This is mainly useful for formatting hudmessages which you want to put in the same position on all screens when using SetHudSize.

Return value

The vertical resolution of the screen.

Examples

This script places a "™ symbol at the left hand side of the screen, 100 pixels from the bottom.

script 1 (void)

```
{
    int h = GetScreenHeight();
    SetHudSize(GetScreenWidth(), h, 1);

    h = (h - 100) << 16;

    HudMessage(s:"@"; HUDMSG_PLAIN, 1,
        CR_BLUE, 0.1, h + 0.1, 10.0);
}
```

The application of this is that upon building a custom graphic for a HUD, you may want to position info/health/meters/etc. around the edge of the screen according to some positioning system, without necessarily resizing them. See also the command SetFont.

GetScreenWidth

Warning: This feature has at least one use case where the outcome is indeterminate. This feature can break demo and multiplayer sync if an indeterminate result is used to modify the playsim (anything that uses the random number generator, modify level geometry, spawn obstacles, monsters, or powerups, and so on). Usage of the feature in conjunction with non-playsim related features, such as displaying a HudMessage, is safe.

GetScreenWidth only knows the resolution of the local session. Information about the resolution of other players in a game or demo does not exist.

int GetScreenWidth (void)

Usage

Returns the horizontal resolution. For example, if you are playing in 800x600 it will return 800. This is mainly useful for formatting hudmessages which you want to put in the same position on all screens when using SetHudSize.

Return value

The horizontal resolution of the screen.

Examples

This script places an `~@`™ symbol at the left hand side of the screen, 100 pixels from the bottom.

script 1 (void)

```
{
    int h = GetScreenHeight();
    SetHudSize(GetScreenWidth(), h, 1);

    h = (h - 100) << 16;

    HudMessage(s:"~@"; HUDMSG_PLAIN, 1,
        CR_BLUE, 0.1, h + 0.1, 10.0);
}
```

The application of this is that upon building a custom graphic for a HUD, you may want to position info/health/meters/etc. around the edge of the screen according to some positioning system, without necessarily resizing them. See also the command SetFont.

GetSectorCeilingZ

`int GetSectorCeilingZ (int tag, int x, int y);`

Usage

This returns the sector ceiling height at coordinates [x, y].

Parameters

tag: tag of the sector.

x, y: coordinates of the point (not fixed point value!).

Return value

Returns the sector ceiling height at coordinates [x, y] as a fixed point value. When used on sectors which share tags, it will return the ceiling height of the sector with the lowest sector number and the matching tag. If the sector is flat (ceiling is not sloped), this will just return the ceiling height no matter where the coordinates are specified, so [0, 0] is as good as anywhere. If the sector is sloped and the coordinates are specified in an area outside of the the sector, a projected height (the height the ceiling would be if the sector were extended to that area) is returned. If tag is 0, the function returns the ceiling height of whatever sector is found at [x, y].

Examples

This example will print the ceiling height in the sector of the specified tag at the coordinates of the specified tid.

```
script 1 (int tag, int tid)
{
    int x, y, z;
    x = GetActorX(tid) >> 16;
    y = GetActorY(tid) >> 16;
    z = GetSectorCeilingZ(tag, x, y);
    print(f:z);
}
```

GetSectorFloorZ

fixed GetSectorFloorZ (int tag, int x, int y);

Usage

This returns the sector floor height at coordinates [x, y].

Parameters

tag: Tag of the sector.

x, y: Coordinates of the point (not fixed point value!).

Return value

Returns the sector floor height at coordinates [x, y] as a fixed point value. When used on sectors which share tags, it will return the floor height of the sector with the lowest sector number and the matching tag. If the sector is flat (floor is not sloped), this will just return the floor height no matter where the coordinates are specified, so [0, 0] is as good as anywhere. If the sector is sloped and the coordinates are specified in an area outside of the the sector, a projected height (the height the floor would be if the sector were extended to that area) is returned. If tag is 0, the function returns the floor height of whatever sector is found at [x, y].

Examples

This script should be run via an "Actor enters sector" special, or by assigning it to the lines around a platform. It makes the platform raise every time the player stands on it, and when it is too high from the surrounding floor (assumed to be 0.0) then it reports that the player cannot climb on to the platform another time.

```
script 1 (int sector)
```

```
{
```

```
    Floor_RaiseByValue (sector, 10, 8);
```

```
    TagWait (sector);
```

```
    if (GetSectorFloorZ (sector, 0, 0) > 24.0)
```

```
        Print (s:"This platform is too\nhigh to climb on!");
```

```
}
```

Another use of this command is creating lifts which run to a number of floors one after the other and then return to their start floor. This command can detect if they are at specific floors.

GetSectorLightLevel

int GetSectorLightLevel (int tag)

Usage

This returns the sector's light level.

Parameters

tag: Tag of the sector.

Return value

Returns the sector's light level as a value between 0 and 255. When used on sectors which share tags, it will return the light level of the sector with the lowest sector number and the matching tag.

Examples

script 1 (int sector)

```
{
    int l = GetSectorLightLevel (sector);
    Light_ChangeToValue (sector, 8);
    Delay (8);
    Light_ChangeToValue (sector, l);
}
```

GetSigilPieces

int GetSigilPieces (void)

Usage

Returns the number of Sigil pieces that are held by the player.

Examples

This script summons a robotic enemy at the given spot and assigns it the given TID. How strong it is depends on how many Sigil pieces are held.

```
str enemy[5] = { "Stalker", "Sentinel", "Reaver", "Crusader", "Inquisitor" };
```

```
script 1 (int spot, int mid)
{
    SpawnSpotFacing(enemy[GetSigilPieces() - 1], spot, mid);
}
```

GetUserArray

GetUserArray (int tid, str name, int pos)

Usage

Retrieves the value of one of the affected actor's user or native array-bound variables. Native arrays are arrays which are preceded by the keyword native when declared.

Parameters

tid: the TID of the affected actor. If 0, the script's activator is used.

name: the name of the array. Acceptable array types are int, double, bool, string and name.

pos: the position in the array, 0-indexed.

Return value

The value stored in the specified position of the array. double is returned as a fixed-point value, while name is returned as a string.

Examples

This function adds a rank to the specified 'skill' in the activator.

```
function void SetRanks (int skill, int amt)
{
    SetUserArray(0, "user_skills", skill, amt + GetUserArray(0, "user_skills", skill));
}
```


GetUserCVar

(This is for the ACS function. For the equivalent DECORATE function please [click here](#).)

int GetUserCVar (int playernumber, str cvar)

Usage

Returns the value of a particular user cvar of a specific player.

Parameters

playernumber: The number of the player (0 to 7).

cvar: name of a console variable to get the value from.

Return value

The value of the specified console variable. The console variable must also be a proper user variable, such as the ones found in Player settings. This is only useful for cvars that can be represented as integers. Also note that you can create your own user variables by using the CVARINFO lump.

If no such console variable exists, if the console variable is not a proper user variable, or if the player is not in game (or the number is not a valid player number) it will return 0.

GetUserCVarString (ACS)

string GetUserCVarString (int playernumber, str cvar)

Usage

Returns the value of a particular user cvar of a specific player.

Parameters

playernumber: The number of the player (0 to 7).

cvar: name of a console variable to get the value from.

Return value

(Verification needed)

The value of the specified console variable. The console variable must also be a proper user variable, such as the ones found in Player settings. For numerical values, see GetUserCVar instead. Also note that you can create your own user variables by using the CVARINFO lump.

If no such console variable exists, if the console variable is not a proper user variable, or if the player is not in game (or the number is not a valid player number) it will return an empty string.

GetUserVariable

GetUserVariable (int tid, str name)

Usage

Retrieves the value of one of the affected actor's user or native variables. Native variables are variables which are preceded by the keyword native when declared.

Parameters

tid: the TID of the affected actor. If 0, the script's activator is used.

name: the name of the variable. Acceptable variable types are int, double, bool, string and name.

Return value

The value stored in the variable. double is returned as a fixed-point value, while name is returned as a string.

Examples

This script increases the amount of XP that the activating enemy rewards upon death.

```
script "EnemyExp" (int amt)
{
    SetUserVariable(0, "user_rewardxp", amt + GetUserVariable(0, "user_rewardxp"));
}
```

GetWeapon

str GetWeapon (void)

Usage

Retrieves the class name of the weapon currently equipped by the script activator. As only the player can have weapons, the function returns meaningful results only if the activator is a player.

Return value

Returns the class name of the weapon currently equipped by the script activator, as a string, if the activator is a player. If not, the function returns "None".

Examples

This script gives the player a little bit of ammunition depending on the currently equipped weapon.

```
script "AmmoProvider" (void)
{
    str weapon = GetWeapon();

    if (StrIcmp(weapon, "Pistol") == 0 || StrIcmp(weapon, "Chaingun") == 0)
    {
        GiveInventory("Clip", 10);
        Log(s:"Received a clip.");
    }
    else if (StrIcmp(weapon, "Shotgun") == 0 || StrIcmp(weapon, "SuperShotgun") == 0)
    {
        GiveInventory("Shell", 4);
        Log(s:"Received 4 shotgun shells.");
    }
    else if (StrIcmp(weapon, "RocketLauncher") == 0)
    {
        GiveInventory("RocketAmmo", 1);
        Log(s:"Received a rocket.");
    }
    else if (StrIcmp(weapon, "PlasmaRifle") == 0 || StrIcmp(weapon, "BFG9000") == 0)
    {
        GiveInventory("Cell", 20);
        Log(s:"Received an energy cell.");
    }
    else Print(s:"A firearm needs to be equipped first to receive ammo for it");
}
```

IsPointerEqual

bool IsPointerEqual (int ptr_select1, int ptr_select2 [, int tid1 [, int tid2]]); “ ACS version
bool IsPointerEqual (int ptr_select1, int ptr_select2) “ DECORATE/ZScript version

Usage

Compares two pointers and see if they reference the same actor or not. tid1 determines the actor used to resolve ptr_select1, and tid2 determines the actor used to resolve ptr_select2. If tid1 and tid2 are equal, the same actor is used for resolving both pointers. Passing a tid of 0 for either tid* means that the check is done on the activator of the script to resolve the respective pointer.

The DECORATE version behaves the same, except that it does not accept tids, and instead, the check is done on the caller of the function to resolve the pointers.

Note that the DECORATE version of the function is to be used where an expression is expected. It is mostly useful when combined with A_JumpIf.

Parameters

ptr_select1: the first pointer to pass for comparison.

ptr_select2: the second pointer to pass for comparison.

tid1: the tid to determine the actor used to resolve the first pointer (ptr_select1). Default is 0, which refers to the activator.

tid2: the tid to determine the actor used to resolve the second pointer (ptr_select2). Default is 0, which refers to the activator.

Return value

The function returns true if both pointers refer to the same actor (equal), otherwise the returned value is false.

Examples

This imp reports two messages on death: one when it is killed by the player (player 1 in this case), and another when it is killed by anything otherwise. The function checks the imp's target at that moment and see if it was the player, jumping to the Player1Kill state if it was and printing the message.

ACTOR SomeImp : DoomImp

```
{
  States
  {
    Death:
      TNT1 A 0 A_JumpIf(IsPointerEqual(AAPTR_TARGET, AAPTR_PLAYER1) == TRUE, "Player1Kill")
      TNT1 A 0 A_PrintBold("Killed by something else")
      Goto Super::Death
    Player1Kill:
      TNT1 A 0 A_PrintBold("Killed by player 1")
      Goto Super::Death
  }
}
```

IsTIDUsed

bool IsTIDUsed (int tid)

Usage

Returns whether any actors using the given TID exist. This is more efficient than ThingCount(T_NONE, tid), because it only needs to check for one actor with the TID and not all of them. It also makes no distinction between dead and live things like ThingCount does.

Parameters

tid: the Thing ID to check.

Return value

TRUE (1) if any actor using that TID exists, alive or dead; FALSE (0) otherwise.

LineSide

int LineSide (void)

Usage

Returns an integer value based on the side of the line a script was activated from. For readability there are defines in `zdefs.acs` as follows:

```
LINE_FRONT = 0
```

```
LINE_BACK = 1
```

This allows you to carry out actions depending on which way the player crosses the line.

Examples

This script creates a one-way corridor. It uses a sequence of lines, all set to be repeatable and activated when walked over. They activate this script, script using the parameter which is the direction the player would be going in if they were walking from the front of the line over to the back.

```
script 1 (int angle)
```

```
{  
    if (LineSide() == LINE_BACK)  
        ThrustThing(angle, 10, 0, 0);  
}
```

If the player crosses the line normally, e.g. front to back, they will not be hindered. If they try to cross it in reverse, e.g. back to front, they will be thrust away in the correct direction.

PlayerClass

int PlayerClass (int playernumber)

Usage

Returns the number of the player's player class. The corresponding numbers are in the order of how they are entered into the MAPINFO or KEYCONF lumps (MAPINFO comes first). To see the numbers during runtime use the playerclasses console command to output all the classes into the console. If the returned number is -1 that means the activator was either a monster or out of player range.

While not too useful, the following constants are given for the hexen classes:

CLASS_FIGHTER - Fighter (0)

CLASS_CLERIC - Cleric (1)

CLASS_MAGE - Mage (2)

Examples

This example script (works for Hexen only) just prints the player class of the activator using the PlayerNumber function:

```
#include "zcommon.acs"
script 1 (void)
{
    switch(PlayerClass(PlayerNumber()))
    {
        case CLASS_FIGHTER:
            print(s:"You are a fighter!");
            break;
        case CLASS_CLERIC:
            print(s:"You are a cleric!");
            break;
        case CLASS_MAGE:
            print(s:"You are a mage!");
            break;
        case -1:
            //monster activated the script but we do not care about that
        default:
            //something happened but we are just gonna break
            break;
    }
}
```


PlayerCount

int PlayerCount (void)

Usage

Returns the number of players currently in the game. For single-player games, this will always be 1. For multi-player games, it can also return 1 when all but one of the players has quit the game.

Examples

A common issue is how to check if all players have entered a certain area. That is, if the number of players that have entered the area is equal to the result of PlayerCount.

Using "Actor enters sector" and "Actor leaves sector" things, it is possible to implement a primitive counter.

```
int count = 0;
```

```
// Use for Actor enters sector
```

```
script 10 (void)
```

```
{  
    count++;  
}
```

```
// Use for Actor leaves sector
```

```
script 11 (void)
```

```
{  
    count--;  
}
```

```
script 100 OPEN
```

```
{  
    while (count < PlayerCount())  
        Delay(35);  
  
    PrintBold(s:"All players ready!");  
  
    //etc.  
}
```

This counter will keep track of the number of players currently in the sector and when script 100 finds the counter to equal PlayerCount, a special action occurs.

Note that this does not account for such events as players dying in the sector, or players disconnecting from the game in the sector. It is possible to prevent these problems by using an array of boolean variables, one for each player, and checking the total against PlayerCount. DEATH and DISCONNECT scripts can reset the particular player's state.

Another possible use of this function is to perform certain actions depending on player count.

```
script 12 (void)
```

```
{  
    if (PlayerCount() >= 6)  
        Thing_Spawn(1, T_CYBERDEMON, 0, 0);  
}
```

This script 12 would spawn a Cyberdemon only in case there are 6 or more of active players on the server.

```
script 13 (void)
```

```
{  
    for (int i=0;i<PlayerCount();i++)  
        Thing_Spawn(i+2000, T_BARON, 0, 0);  
}
```

Script 13 would spawn as many Barons of Hell as number of active players on the server. Barons will be spawned in mapspots tagged 2000, 2001, 2002, etc so you have to create them on your map. Note that if the spawning location is blocked by heights of sectors or other objects, not all of Barons might be spawned. Consider tagging Barons with the last argument of Thing_Spawn function and using ThingCount to calculate how many of them were actually created to spawn the rest afterwards.

PlayerFrag

int PlayerFrag (void)

Usage

Returns the current number of frags for the player who activated the script. This is only really useful in multiplayer games. Remember that the number returned may be negative if the player has committed suicide.

For scripts not activated by a player, PlayerFrag will return 0.

Examples

In this example, if the player gets a frag the player's frag count and name are displayed. It does this by assigning a script as the thing special for each player. The script is executed when the player dies and the activator is credited as the thing that killed the player. Script 3 checks the PlayerNumber to make to only print in the event that the activator was a player. Note that this will also activate if the player suicides.

script 1 enter

```
{
    SetThingSpecial(0, 226, 3, 0);
}
```

script 2 respawn

```
{
    SetThingSpecial(0, 226, 3, 0);
}
```

script 3 (void)

```
{
    if (PlayerNumber() >= 0)
        printbold(n:0, s:" has ", i:PlayerFrag(), s:" frags");
}
```

PlayerInGame

bool PlayerInGame (int playernumber)

Usage

Returns true if the player [0..7] is in the game and false if not. If it returns false, that player is not counted by PlayerCount.

Examples

This example picks a player at random of the players that are in the game. It then cripples this player by reducing their speed for 30 seconds. The script then prints a message for whether the player quit, was killed, or survived the 30 seconds. If the player survived, their speed is restored.

```
int alive[8];
```

```
script 1 enter
```

```
{
    alive[PlayerNumber()] = TRUE;
    Thing_ChangeTID(0, 1000 + PlayerNumber());
}
```

```
script 2 respawn
```

```
{
    alive[PlayerNumber()] = TRUE;
    Thing_ChangeTID(0, 1000 + PlayerNumber());
}
```

```
script 3 death
```

```
{
    alive[PlayerNumber()] = FALSE;
}
```

```
script 4 open
```

```
{
    int p, tics;
    while (GameType() == GAME_NET_DEATHMATCH) //don't run otherwise
    {
        delay(35 * 60);

        do {
            p = random(0, 7);
        } while (!PlayerInGame(p) && !alive[p]);

        SetActorProperty(1000 + p, APROP_Speed, 0.5);
        printbold(s:"Everybody get ", n:p + 1, s:"!");

        tics = 0;
        do {
            tics++;
            delay(1);
        } while (tics <= 35 * 30 && alive[p] && PlayerInGame(p));
    }
}
```

```
if (tics <= 1050)
{
  if (!alive[p])
    printbold(s:"Well that takes care of that!");
  else
    printbold(s:"The quitter!");
}
else
{
  SetActorProperty(1000 + p, APROP_Speed, 1.0);
  printbold(n:p + 1, s:" dodged the bullet!");
}
}
}
```

PlayerIsBot

bool PlayerIsBot (int playernumber)

Usage

Returns TRUE if the player [0..7] is a bot and FALSE if not.

Examples

Giving a vital piece of puzzle or ability to a player could cause the game to become impossible if it is given to a bot rather than a human. This command can protect against such an issue. For example:

script 55 (void)

```
{
    int marine = Random(0, PlayerCount());

    while (PlayerIsBot(marine))
        // Pick another marine
        marine = Random(0, PlayerCount());

    SetActorProperty(1000 + marine, APROP_INVULNERABLE, TRUE);

    PrintBold(n:marine+1, s:" is totally invulnerable!");
}
```

script 1000 ENTER

```
{
    Thing_ChangeTID(0, 1000 + PlayerNumber());
}
```

Script 1000 just gives the players TIDs, in sequence 1000, 1001, 1002, etc. Script 55 is the important one here. First it sets marine to be the number of one of the players. However, if it picks a bot, it keeps picking a new number until it picks a non-bot player. Once it has a human player, it makes them totally invulnerable and reports to the other players that this is so.

PlayerNumber

int PlayerNumber (void)

Usage

Returns the player number for the player who activated the script, starting at 0. For scripts that were not activated by a player, PlayerNumber will return -1.

Examples

A useful application of this command is to give each player an individual TID. The code to do this is:

script 5 ENTER

```
{  
    Thing_ChangeTID(0, 1000 + PlayerNumber());  
}
```

This will assign players TIDs of 1000, 1001, 1002... so they can be accessed individually. They can also be accessed as a group using for loops. The following script gives all players maximum health based on the TIDs from script 5.

script 10 (void)

```
{  
    for (int n = 0; n < PlayerCount(); n++)  
        SetActorProperty(1000 + n, APROP_HEALTH, 200);  
}
```

This example will add to a map level variable if the activator of the script is a player. It takes advantage of the fact that PlayerNumber will return -1 for a non-player activator.

int teamkills;

script 15 (void)

```
{  
    if (PlayerNumber() >= 0)  
    {  
        teamkills++;  
    }  
}
```

SetResultValue

void SetResultValue (int value)

Usage

When a script is executed, it has an associated "result value". This is somewhat akin to a function's return value. Normally, a script's result value is 1, but SetResultValue can be used to change this to something else. However, you must use SetResultValue before calling any functions that make the script wait, such as Delay or TagWait. After the script starts waiting, its result value can no longer be changed.

The following are uses for a script's result value:

Pseudo-functions

If you execute a script with ACS_ExecuteWithResult, ACS_ExecuteWithResult will return the script's result value. This lets you use a script as if it was a function.

Switches

If you place the ACS_ExecuteWithResult special on a switch linedef, you can use SetResultValue to prevent the normal switch action from taking place (e.g. because the script did not perform any action). If the script's result value is set to 0, the switch texture will not be changed, nor will the switch sound be played.

DECORATE

You can use ACS_ExecuteWithResult in CustomInventory item's state chains to notify the item about the success or failure of the state chain.

Examples

This shows how SetResultValue is used with ACS_ExecuteWithResult to retrieve a return value from another script.

script 1 (void)

```
{
    Print(d:ACS_ExecuteWithResult(2, 0, 0, 0)); //prints 667
}
```

script 2 (void)

```
{
    SetResultValue(667);
}
```

As this command was added specifically so that many of Strife's unique specials could be implemented through ACS, strifehelp.acs is also a good place to see this command in action.

StrArg

```
int StrArg (str string);
```

Usage

Creates an integer representation of string which can be used as an argument to Action specials used on lines, actors, etc, most prominently ACS_NamedExecute.

Parameters

string: The string to convert.

Return Value

An implementation defined integer value representing string, which may not be used in contexts other than an action special.

StrCmp

```
int StrCmp (str string1, str string2[, int maxcomparenum]);  
int StrIcmp (str string1, str string2[, int maxcomparenum]);
```

Usage

Compares the two strings passed in arguments string1 and string2 character by character. If maxcomparenum is specified, it only compares up to maxcomparenum characters of each string.

StrIcmp is a case-insensitive version of StrCmp.

Parameters

string1: The string to compare string2 with.

string2: The string to compare string1 with.

maxcomparenum: The maximum number of characters of each string to compare.

Return Value

0 if the two strings or a portion of each string up to the maxcomparenum-th character are equal

a positive value if the first character that does not match has a greater value in string1

a negative value if the first character that does not match has a greater value in string2

Example

These examples are illustrative, you will need to put the function into right context and you will want to process the return value.

```
str s1 = "BFGBall";  
str s2 = "BFGExtra";
```

```
StrCmp(s1, "BFGBall"); //returns 0
```

```
StrCmp(s1, s2); //returns a positive value
```

```
StrCmp(s2, s1); //returns a negative value
```

```
StrCmp(s1, s2, 3); //returns 0
```

```
StrCmp(s1, "bfgball"); //returns a positive value
```

```
StrIcmp(s1, "bfgball"); //returns 0
```

ThingCount

int ThingCount (int type, int tid)

Usage

ThingCount counts all things specified on the map. You may specify a type of monster via spawn numbers, a specific TID, or both (monsters with a specific TID).

ThingCount will not count dead monsters, even though their things still exist and can be used, for instance, by SpawnSpot.

Note that for this function to count a specific type of actor, it will need to have a spawn number assigned. If you need to count actors without a SpawnID, use the alternative ThingCountName function instead.

If the type parameter is T_NONE, ThingCount will count all actors with the given TID, regardless of their type. Inversely, if the tid parameter is 0, it will count all actors of the given type, regardless of TID.

Examples

For example let's say you have a map with 10 enemies:

Imp - tid 5

Imp - tid 5

Imp - tid 0

Baron - tid 5

Baron - tid 5

Baron - tid 4

Baron - tid 0

Demon - tid 5

Demon - tid 4

Demon - tid 0

Here are some example values:

ThingCount(T_IMP, 0) = 3

ThingCount(T_BARON, 0) = 4

ThingCount(T_DEMON, 0) = 3

ThingCount(T_NONE, 5) = 5

ThingCount(T_NONE, 4) = 2

ThingCount(T_IMP, 5) = 2

ThingCount(T_DEMON, 4) = 1

ThingCount(T_NONE, 0) = 10

ThingCount(T_IMP, 4) = 0

This script simulates the behavior of E1M8. It waits for all the Barons with tag 10 to be defeated. Once they are, it lowers a sector tagged 15 to the lowest adjacent floor.

```
script 1 OPEN
```

```
{  
    While(ThingCount(T_BARON, 10) > 0)  
        Delay(35);
```

```
    Floor_LowerToLowest(15, 20);  
}
```

This script is open so it's run from the start of the map. Alternatively it could be modified to run upon crossing a line and made more general. The while loop just keeps executing the delay command whilst there are still Barons on the level.

The reason for this is there is no need for the script to check every single frame whether the barons are dead. It delays so that it checks only once per second, which might speed things up a very small amount. Once all the Barons die, the while loop stops and the floor gets lowered.

This script shows that ThingCount can be used effectively with Spawn and SpawnSpot, as both of these allow you to add a TID to the spawned thing. It takes the TID of a teleporter spot as its parameter.

```
script 1 (int teletid)
{
    SpawnSpot("Revenant", teletid, 100, 0);

    While(ThingCount(T_REVENANT, 100) > 0)
        Delay(35);

    Print(s:"You have beaten the revenant!");
}
```

First it spawns a Revenant with a (hopefully) unique tid of 100. Afterwards it delays whilst the revenant is still alive. Once the revenant is dead, it shows the player they have beaten it (as if they wouldn't know).

This script displays a counter of how many of a group monsters you must defeat before you have won. It takes a parameter, which is the TID of the group of monsters

```
script 5 (int armytid)
{
    Thing_Hate(armytid, 0, 0);

    While (ThingCount(0, armytid) > 0)
    {
        HudMessage(s:"Defeat ", d:ThingCount(0, armytid), s:" more monsters!";
            HUDMSG_PLAIN, 1, CR_RED, 0.5, 0.9, 2.0);

        Delay(35);
    }

    HudMessage(s:"Victory!";
        HUDMSG_PLAIN, 1, CR_GOLD, 0.5, 0.9, 2.0);
}
```

This demonstrates the usefulness of using a monster type of 0.

ThingCountName

int ThingCountName (str classname, int tid)

Usage

ThingCountName counts all things specified on the map. Monsters are specified by their names (see Classes), or by name and tid.

ThingCountName will not count dead monsters, even though their things still exist and can be used, for instance, by SpawnSpot.

Example

For example let's say you have a map with 10 enemies:

Imp - tid 5

Imp - tid 5

Imp - tid 0

Baron - tid 5

Baron - tid 5

Baron - tid 4

Baron - tid 0

Demon - tid 5

Demon - tid 4

Demon - tid 0

Here are some example values:

ThingCountName("DoomImp", 0) = 3

ThingCountName("BaronOfHell", 0) = 4

ThingCountName("Demon", 0) = 3

ThingCountName("DoomImp", 5) = 2

ThingCountName("Demon", 4) = 1

ThingCountName("DoomImp", 4) = 0

Notice that specifying a tid of 0 means that ThingCountName will just ignore the tid and count all instances of the specified thing, as opposed to only counting things with a tid of 0.

ThingCountNameSector

int ThingCountNameSector (str classname, int tid, int tag)

Usage

Counts the number of actors in a given sector matching the given name and TID.

Parameters

classname: The class name of the actors to count.

tid: The Thing ID of the actor(s) to count. Use 0 to count actors regardless of TID.

tag: The tag of the sector(s) to count actors in.

Return value

Returns the number of actors in the given sector(s) matching the given criteria.

Examples

This line counts the number of cacodemons with a tid of 62 in all sectors with a tag of 11:

```
int CacosPresent = ThingCountSector ("Cacodemon", 62, 11);
```

This line counts all actors with a class name of "MySuperImp" in sectors with tag 40:

```
int AllActors = ThingCountSector ("MySuperImp", 0, 40);
```

ThingCountSector

int ThingCountSector (int type, int tid, int tag)

Usage

Counts the number of actors in a given sector matching the given criteria. Only actors whose health is above 0 are counted.

Parameters

type: The type of actor to count. Use T_NONE to count actors of any type.

tid: The Thing ID of the actor(s) to count. Use 0 to count actors regardless of TID.

tag: The tag of the sector(s) to count actors in.

Return value

Returns the number of actors in the given sector(s) matching the given criteria.

Examples

This line counts the number of cacodemons with a tid of 62 in all sectors with a tag of 11:

```
int CacosPresent = ThingCountSector (T_CACODEMON, 62, 11);
```

This line counts every actor in sectors with tag 40:

```
int AllActors = ThingCountSector (T_NONE, 0, 40);
```

Timer

`int timer (void)`

Usage

Returns the number of ticks that have passed since a particular epoch. For maps that are not part of a hub, this will be the time since the level was started. For maps that are part of a hub, this will be the time since the user started a new game.

Examples

A full game timer can be added by adding a script like this to every level in a hub:

```
script 1 ENTER
{
    int t;
    while(TRUE)
    {
        t = Timer() / 35;

        HudMessage(d:t/60, s:":", d:(t%60)/10, d:t%10;
            HUDMSG_PLAIN, 1, CR_RED, 0.95, 0.95, 2.0);

        Delay(35);
    }
}
```

This will only work on hubs, which have to be defined using MAPINFO. It will display and keep track of the total time spent playing in the bottom right corner of the screen.

UniqueTID

`int UniqueTID ([int tid[, int limit]])`

Usage

Returns a new TID that is not currently used by any actors.

Parameters

tid: Starting value from which to check. It has two modes of operation:

If **tid** is non-zero, then it checks TIDs one-by-one starting at the given **tid** until it finds a free one.

If **tid** is zero, then it returns a completely random TID.

limit: Specifies the number of attempts to make to find a free TID. If **limit** is non-zero, then it will only check that many times for a free TID, so it might not find a free one. If no free TID is found, 0 is returned. If **limit** is zero, then the search is effectively unlimited.

Return value

0 if no free TID is found, a positive value corresponding to an unused TID otherwise.

Examples

This example is part of the code that can be used to see through the "eyes" of a missile(akin to Unreal Tournament's Redeemer) It is called in the Spawn state of the missile.

```
script "CameraRocket" (void)
{
    int MissileID = UniqueTID(); // Get a random TID, it is sure that it will not use an already used one.
    Thing_ChangeTID(0, MissileID); // Set that TID to the fired missile
    SetActivatorToTarget(MissileID); // Use the target to select the player who fired this missile
    ChangeCamera(MissileID, 0, 0); // Set the camera for the player
    SetPlayerProperty(0, ON, PROP_Frozen); // Don't let the player move
    Thing_ChangeTID(MissileID, 0); // Remove the TID from the missile, as it is no longer needed.
}
```

Note that the above function does not include turning of the missile based on the view angle or unfreezing the player when it explodes.

CheckActorInventory

```
int CheckActorInventory(int tid, str inventory_item);
```

Usage

Checks the given actor's inventory for the item specified by `inventory_item`. For a list of possible inventory items, see [Inventory](#).

This function does not treat tid 0 as the activator of the script. To check the script activator's inventory, see [CheckInventory](#).

The function will return the number of items of the given type the actor carries.

Examples

This example will give any player a shotgun who does not have one at the time script 2 is activated, regardless of how script 2 is activated.

```
script 1 enter
```

```
{
    Thing_ChangeTID(0, 1000 + PlayerNumber());
}
```

```
script 2 (void)
```

```
{
    for (int p = 0; p < 8; p++)
        if (PlayerInGame(p) && !CheckActorInventory(1000 + p, "Shotgun"))
            GiveActorInventory(1000 + p, "Shotgun", 1);
}
```

CheckInventory

```
int CheckInventory(str inventory_item);
```

Usage

Checks the inventory of the actor who activated the script for the item specified by `inventory_item`. For a list of possible inventory items, see [Inventory](#).

The function will return the number of items the actor carries. Keep in mind that for active powerups, you cannot check for the pickup item. You have to check for the internal powerup item.

Examples

An example of a script which for no good reason gives terrible advice to the player:

```
script 52 (void)
```

```
{
    if (CheckInventory("Shotgun") && CheckInventory("Shell") > 20)
        Print(s:"Use the shotgun to take out those 20 imps!");
    else
        Print(s:"Run away from the imps you loser!");
}
```

If the player has the shotgun and 20 shells, it tells them to use the shotgun on some imps. Otherwise it tells the player to run away.

CheckWeapon

bool CheckWeapon (str weapon)

Usage

If player's active weapon is the weapon with specified class name, this function returns true. If not, it returns false. The class name must be a type of weapon.

Examples

There are a number of useful applications of this command. In this example, a wall is made that can be "destroyed" by using the chainsaw upon it. It uses CheckWeapon to make sure it is being sawed and not shot or punched.

The first script just reports to the player that it can be sawed down, as they would be unlikely to work it out otherwise. This can be set to any line to activate once.

```
script 22 (void)
```

```
{  
    Print(s:"That wall looks like it\ncould be sawed down...");  
}
```

```
int sawed = 0;
```

```
script 23 (int sector)
```

```
{  
    if (CheckWeapon("Chainsaw") && sawed < 64)  
    {  
        Floor_LowerByValue(sector, 20, 4);  
        sawed += 4;  
    }  
}
```

The script which handles the sawing is number 23. It should be set to a line as a "Projectile hits" type of activation, and it should be set to be repeatable. The sectors which "destroy" upon sawing should be given a tag which is passed to this function. The variable sawed keeps track of how much sawing has been done. The limit here is 64. If the player is using the chainsaw and there is still some wall left to be sawed, the floor is lowered and the total is updated. This is repeated whilst the player chainsaws the wall until the limit is reached or until they get bored and wander away.

ClearActorInventory

```
void ClearActorInventory(int tid);
```

Usage

This function will clear the specified actor's inventory in a similar fashion to ClearInventory.

Note: This will not remove an inventory item or weapon that is flagged with the INVENTORY.UNDROPPABLE flag. An explicit call to TakeActorInventory is required to remove such items.

Examples

This could be a very nasty piece of code if implemented into a deathmatch level. Every minute a player will be picked at random and stripped of their inventory.

```
#define NUM_PLAYERS    8
#define PLAYER_BASE_TID 1000
#define CLEAR_INTERVAL 2100 // 1 minute

script 1 ENTER
{
    Thing_ChangeTID(0, PLAYER_BASE_TID + PlayerNumber());
}

script 2 OPEN
{
    int p;

    while (TRUE)
    {
        delay(CLEAR_INTERVAL);

        do {

            p = random(0, NUM_PLAYERS-1);

        } until (PlayerInGame(p));

        ClearActorInventory(PLAYER_BASE_TID + p);

        HudMessageBold(n:p+1, s:" is begging for a quick death!";
            HUDMSG_FADEOUT, 1, CR_RED, 0.5, 0.5, 3.0, 1.0);
    }
}
```

ClearInventory

```
void ClearInventory(void);
```

Usage

This will clear the player's inventory of weapons, ammo, armor and usable items.

Note: This will not remove an inventory item or weapon that is flagged with the INVENTORY.UNDROPPABLE flag. An explicit call to TakeInventory is required to remove such items.

If the function is run with no activator (for example an OPEN script or removing an existing activator with SetActivator), it will run for and affect all active players in the game.

For a list of things to give, see Inventory.

Examples

This example works cleanly in Doom.

```
script 50 (void)
{
    Print(s:"You hand your weapons over to security.");
    ClearInventory();
    GiveInventory("Fist", 1);
}
```

DropInventory

```
void DropInventory (int tid, str itemtodrop);
```

Usage

Drops to the ground the specified item from the inventory of the actor or actors with the matching tid. The item has to be present in the inventory in order for it to be dropped. With each call of the function, only one sample of the specified item is dropped.

If tid is 0, the item is dropped from the script activator's inventory.

Parameters

tid: the tid of the actor or actors from which the item is to be dropped.

itemtodrop: the item to be dropped.

Examples

This script will drop a shotgun from the activator of the script, provided that the activator has a shotgun in their inventory, of course.

Script 1 (void)

```
{  
    DropInventory(0, "Shotgun");  
}
```

DropItem

`int DropItem (int tid, string item [, int dropamount [, int chance]])`

Usage

Causes actors with the matching TID to drop the specified item. If tid is 0, the activator of the script is considered to be the dropper of the item. This works in a similar way to the DropItem actor property.

Parameters

tid: The TID of the actor which the item is dropped by.

item: The item to drop. This can be any valid actor class, not just inventory.

dropamount: The inventory amount the dropped item contains. This is only meaningful with actors inheriting from the Inventory class. If dropamount is greater than 0, the amount of inventory gained from picking up the item equals exactly to that number. Otherwise, the amount gotten is the same as the amount defined by the item itself, except for ammo items, which in this case, the amount is determined by the current skill level's DropAmmoFactor. Default is -1.

chance: The probability of the drop. The item is never dropped if this is -1 or less, while it is always dropped if this is 255 or greater. Default is 256.

Return value

Returns the total number of actors that attempted to drop the item, otherwise it returns 0. The success or failure of an item drop has no consequence on the returned value.

GetMaxInventory

int GetMaxInventory (int tid, string inventory);

Usage

Retrieves the maximum amount of the specified inventory item. If the item is present in the inventory of the actor on which to perform the check (henceforth called "reference actor"), the function retrieves the current maximum amount of that item. If, however, the item is absent from the inventory, the maximum amount of the item as defined by its actor class is retrieved, instead.

The function has a special handling for when "Health" is passed as inventory; if the reference actor is a player, it retrieves its current maximum health, otherwise it retrieves the spawn health if the actor is a non-player. This behaves exactly like GetActorProperty with APROP_SpawnHealth.

Parameters

tid: The tid of the reference actor. If this is 0, the check is performed on the activator of the script.

inventory: The inventory item to get its maximum amount.

Return value

The function returns 0 if the reference actor is non-existent, otherwise the value returned is as explained above.

Zandronum

Unfortunately, Zandronum 3.0 does not support GetMaxInventory, and will always return 0. Tip: You can use GetAmmoCapacity as a work-around.

Examples

Nuvolachalk.png Note: This article lists no examples. If you make use of this feature in your own project(s) or know of any basic examples that could be shared, please add them. This will make it easier to understand for future authors seeking assistance. Your contributions are greatly appreciated.

Using GetAmmoCapacity for Zandronum

Define and get the player's max level using GetAmmoCapacity.

DECORATE

ACTOR PlayerLevel : Inventory

```
{
    Inventory.Amount 1
    Inventory.MaxAmount 10
}
```

ACTOR PlayerMaxLevel : Ammo // For GetAmmoCapacity()

```
{
    Inventory.MaxAmount 10 // Same value as MaxAmount for PlayerLevel.
}
```

ACS

script 1 ENTER

```
{
    Print(s:"Max level is: ",d:GetAmmoCapacity("PlayerMaxLevel"));
}
```

GiveActorInventory

```
void GiveActorInventory(int tid, str inventory_item, int amount);
```

Usage

This function will give the amount of items to the specified actor.

This function does not treat tid 0 as the activator of the script. If 0 is given as tid then the items will be given to ALL players. To give items to the script's activator, use GiveInventory.

Examples

This example will give any player a shotgun who does not have one at the time script 2 is activated, regardless of how script 2 is activated.

script 1 enter

```
{
    Thing_ChangeTID(0, 1000 + PlayerNumber());
}
```

script 2 (void)

```
{
    for (int p = 0; p < 8; p++)
    {
        if (PlayerInGame(p) && !CheckActorInventory(1000 + p, "Shotgun"))
        {
            GiveActorInventory(1000 + p, "Shotgun", 1);
        }
    }
}
```

GiveInventory

```
void GiveInventory(str inventory_item, int amount);
```

Usage

This function will give the number of items specified to the activator. In the case of ammo, health and armor it will give the total number (so GiveInventory("Shell", 5) and GiveInventory("ShellBox", 5) will both give the player five shells.)

If the function is run with no activator (for example an OPEN script or removing an existing activator with SetActivator), it will run for and affect all active players in the game.

See the Inventory page for a list of items in ZDoom.

Examples

This script gives the player some equipment. It could be run some time after script 50 in ClearInventory, as an example.

```
script 51 (void)
```

```
{
    PrintBold(s:"Security issues the standard kit.");
    GiveInventory("Pistol", 1);
    GiveInventory("Clip", 50);
    GiveInventory("GreenArmor", 1);
}
```

This will result in the player gaining 70 bullets, as the pistol comes with 20. To set an exact number of ammo after giving a weapon, either use TakeInventory to remove all the player's ammo, or subtract the amount given with the weapon.

SetWeapon

bool SetWeapon(str weaponname);

Usage

Sets the player's current weapon to weaponname.

Returns TRUE if the weapon was set successfully, and FALSE if not.

Here is a listing of the weapons for each game:

Doom:

Fist

Chainsaw

Pistol

Shotgun

SuperShotgun (Doom 2/Final Doom only)

Chaingun

RocketLauncher

PlasmaRifle

BFG9000

Heretic:

Staff

Gauntlets

GoldWand

Crossbow

Blaster

SkullRod

PhoenixRod

Mace

Hexen:

Fighter

FWeapFist

FWeapAxe

FWeapHammer

FWeapQuietus

Cleric

CWeapMace

CWeapStaff

CWeapFlame

CWeapWraithverge

Mage

MWeapWand

MWeapFrost

MWeapLightning

MWeapBloodscourge

Strife:

PunchDagger

StrifeCrossbow

StrifeCrossbow2

AssaultGun
MiniMissileLauncher
StrifeGrenadeLauncher
StrifeGrenadeLauncher2
FlameThrower
Mauler
Mauler2

Note that in the case of Hexen, one class cannot use another classes weapons, so if you make new sprites for the Fighter's Axe and then play as the Cleric, you will be able to pick up the Axe, but not use it. Outside of playing a Hexen game though, any weapon can be used anywhere else.

Although the above list is complete, a full list of inventory can be seen at Inventory, along with details on how to make a textfile containing a list.

Examples

This is the worst script ever:

```
script 666 ENTER
```

```
{  
    while(TRUE)  
    {  
        if (Random(0, 1))  
            SetWeapon("Fist");  
        else  
            SetWeapon("RocketLauncher");  
  
        Delay(35*2);  
    }  
}
```

It randomly selects either the fist or rocket launcher every two seconds, making the player extremely frustrated and annoyed. Note that if the player does not have the weapon, there will be no result from this command. See CheckInventory to check for the weapon.

TakeActorInventory

`void TakeActorInventory(int tid, str inventory_item, int amount);`

Usage

This function will take the amount of items from the specified actor. In the case of ammo, health and armor it will take the total number (so `TakeActorInventory(3, "Shell", 5)` and `TakeActorInventory(3, "ShellBox", 5)` will both take five shells from actors with a TID of 3).

If 0 is given as TID then the specified items will be removed from ALL players.

Unlike `ClearActorInventory`, `TakeActorInventory` can remove items that are flagged as undroppable.

See the inventory page for a list of items in ZDoom.

Examples

In this example, when a player activates script 2, they will steal the most powerful weapon that can be found (in the order given in the weapon string array) from the first player that can be found with it at random. This results in the activating player gaining the weapon and the player found with said weapon losing it.

```
str weapon[7] = {
    "Chainsaw",
    "Shotgun",
    "SuperShotgun",
    "Chaingun",
    "RocketLauncher",
    "PlasmaRifle",
    "BFG9000"
};

bool player_checked[8];

script 1 enter
{
    Thing_ChangeTID(0, 1000 + PlayerNumber());
}

script 2 (void)
{
    int w, p, take, count;
    for (w=6; w>=0; w--)
    {
        while (count < PlayerCount() && !take)
        {
            count = 1;
            do {
                p = random(0, 7);
            } while (!PlayerInGame(p) || player_checked[p] || p == PlayerNumber());

            if (!CheckActorInventory(1000 + p, weapon[w]))
            {
                count++;
                player_checked[p] = 1;
            }
        }
    }
}
```

```
    }  
    else if(CheckActorInventory(1000 + p, weapon[w]))  
    {  
        TakeActorInventory(1000 + p, weapon[w], 1);  
        GiveInventory(weapon[w], 1);  
        take = 1;  
        w = -1;  
    }  
}  
  
for (p=0; p<8; p++)  
    player_checked[p] = 0;  
}
```

TakeInventory

```
void TakeInventory(str inventory_item, int amount);
```

Usage

This function will take the number of items specified from the activator. In the case of ammo, health and armor it will take the total number (so TakeInventory("Shells", 5) and TakeInventory("ShellBox", 5) will both take five shells from the player).

Unlike ClearInventory, TakeInventory can remove items that are flagged as undroppable.

If the function is run with no activator (for example an OPEN script or removing an existing activator with SetActivator), it will run for and affect all active players in the game.

See the Inventory page for a list of items in ZDoom.

Examples

This script randomly removes objects from the player. Sometimes it does not remove any object, though. It is due to how lucky the player is.

```
str weapons[8] = {"Pistol", "Shotgun", "SuperShotgun",  
"Chaingun", "RocketLauncher", "PlasmaRifle", "BFG9000",  
"Chainsaw"};
```

```
script 10 (void)  
{  
    int targ = random(0, 7);  
  
    if (CheckInventory(weapons[targ]))  
    {  
        TakeInventory(weapons[targ], 1);  
        Print(s:"You have dropped a weapon!\ncareless player...");  
    }  
}
```

Note that no inventory can be a negative amount.

UseActorInventory

int UseActorInventory (int tid, str classname)

Usage

Forces the actor(s) with the specified tid to use an item from their inventory, if they have one.

Parameters

tid: TID of the actor(s) that will attempt to use the item. If 0, this forces all players to use the item instead.

classname: The inventory item to be used.

Return value

The return value is the number of actors who used the item successfully.

UseInventory

int UseInventory (str classname)

Forces the activator to use the specified inventory item, if he has it. The return value is true if the item was used successfully, or false if not.

If the function is run with no activator (for example an OPEN script or removing an existing activator with SetActivator), it will run for and affect all active players in the game.

Examples

This example creates a hotkey based function that will use the most powerful healing item in the player's possession.

```
script "BestHealing" (void) net
{
  If (UseInventory("UltimateHealingPotion"))
  {
    terminate;
  }
  If (UseInventory("MajorHealingPotion"))
  {
    terminate;
  }
  If (UseInventory("HealingPotion"))
  {
    terminate;
  }
  If (UseInventory("MinorHealingPotion"))
  {
    terminate;
  }
}
```

ChangeCeiling

```
void ChangeCeiling(int tag, str flatname);
```

Usage

Changes the ceiling texture of all sectors with the specified tag to have flatname as their new ceiling texture. You may also use any texture, pname, sprite or internal graphic (such as TITLEPIC) in place of an actual flat. Note: Only graphics whose dimensions are powers of 2 (i.e.: 32, 64, etc.) will display correctly.

Examples

A simple script to change the texture of any ceilings with the sector tag 4.

Script 1 (void)

```
{  
    ChangeCeiling(4, "RROCK13");  
}
```

ChangeFloor

```
void ChangeFloor(int tag, str flatname);
```

Usage

Changes the floor texture of all sectors with the specified tag to flatname. You may also use any texture, pname, sprite or internal graphic (such as TITLEPIC) in place of an actual flat. Note: Only graphics whose dimensions are powers of 2 (i.e.: 32, 64, etc.) will display correctly.

Examples

This one is pretty straight-forward. In this example, pressing a switch turns off a waterfall, which in turn dries up a stream of water “ or in this case, the SFALL1 texture that is used as a flat.

Script 1 (void)

```
{  
    ChangeFloor(4, "RROCK13");  
}
```

You will need to have the sector that has got the flat you want to change tagged to a unique number “ in this example, the sector was tagged as number 4.

ChangeLevel

void ChangeLevel (str mapname, int position, int flags,[int skill])

Usage

Changes to a new map, places the player at the specified start spot, and optionally changes the skill level at the same time.

Parameters

mapname

The map (by lump name) to change to.

position

The player start number to use. This should match the number specified as the first argument of the player start thing. If there is only one player start, this parameter should be set to 0.

flags

The following flags are supported:

CHANGELEVEL_KEEPPACING: The player's facing angle will not be adjusted during the map change.

CHANGELEVEL_NOINTERMISSION: Do not display the intermission screen during the map change.

CHANGELEVEL_NOMONSTERS: Start the new map with nomonsters in effect.

CHANGELEVEL_PRERAISEWEAPON: The player's weapon will already be raised upon entering the new map.

CHANGELEVEL_RESETHHEALTH: The player's health is reset as if they had started a new game.

CHANGELEVEL_RESETHINVENTORY: The player's inventory is reset as if they had started a new game.

skill

Sets the skill level to start the new map at. This can be useful to create skill-choice maps such as that seen at the beginning of Quake 1.

To keep the skill level unchanged on the new map, use -1 for this parameter. Note that omitting this parameter will change the skill level to 0 (SKILL_VERY_EASY), which is probably not what you want.

Examples

When this script is activated, the player is sent to MAP01 (without intermission), his inventory is reset to default and the skill is set to normal (Hurt Me Plenty).

```
#include "zcommon.acs"
```

```
script 22 (VOID)
```

```
{  
    ChangeLevel ("MAP01", 1, CHANGELEVEL_RESETHINVENTORY|CHANGELEVEL_NOINTERMISSION,  
SKILL_NORMAL);  
}
```

This example will warp the player to map E3M1 at the default player start, will change the skill to Hard (Ultra-Violence) and reset the player's inventory. It will also use the player's current facing.

```
#include "zcommon.acs"
```

```
script 1 (VOID)
```

```
{  
    ChangeLevel ("E3M1", 0, CHANGELEVEL_RESETHINVENTORY|CHANGELEVEL_KEEPPACING,  
SKILL_HARD);  
}
```

ChangeSky

```
void ChangeSky (str sky1, str sky2);
```

Useful in conjunction with SetSkyScrollSpeed.

Usage

Changes the sky texture to sky1 and the second sky texture to sky2. Both textures must be the same height if doublesky is enabled. You may also use any flat, pname, sprite or internal graphic (such as TITLEPIC) in place of an actual texture.

Examples

This simple script changes the Doom 2 default to the red sky from Doom 1.

```
script 1 OPEN
{
    ChangeSky("SKY3","SKY3");
}
```

This would work exactly the same in Doom 1, too. If you wanted to be wacky, you could set the sky to a flat after a few moments of play:

```
script 1 OPEN
{
    Delay(400);
    ChangeSky("FWATER1","FWATER1");
}
```

After 400 tics, we get a beautiful vista of pixelated water. Note that the second parameter is redundant here because the water will tile over whatever is chosen.

ClearLineSpecial

```
void ClearLineSpecial(void);
```

Usage

Clears the special of the line that activated the script. This is most useful for scripts that may require the player to have something and may be triggered more than once, but once they are fully executed (i.e. the player now has the required item), you no longer want it to be executed.

Examples

This script can be triggered an infinite number of times without the blue skull key, but once the player activates it and has that key the line will be cleared of the special and it can no longer be activated.

```
script 1 (void)
```

```
{
    if(CheckInventory("BlueSkull"))
    {
        ClearLineSpecial();
        //do other things here
    }
    else
        Print(s:"You need a blue skull key to activate this");
}
```

This only works for the activating line, if you want to clear a different line that is not the one that triggers the script you should use SetLineSpecial.

QuakeEx

```
bool QuakeEx (int tid, fixed intensityX, fixed intensityY, fixed intensityZ, int duration, int damrad, int tremrad,
sound sfx [, int flags [, fixed mulwavex [, fixed mulwavey [, fixed mulwavez [, int falloff [, int highpoint [,
fixed rollintensity [, fixed rollwave]]]]]]]]];
```

Note: As of (development version aed72f5 only) the intensity parameters are decimal values instead of integers.

Usage

This is an extended version of Radius_Quake2 with the ability to create an earthquake effect to manipulate the player's view along a specific axis, either in direct map axis or along the camera based upon the flags. It also allows adjusting the Z axis (quaking up and down). It also follows the same functionality, and creates an earthquake around the calling actor. In addition, it can also manipulate the ins and outs of an earthquake.

Parameters

tid: Thing ID of map thing(s) for quake foci (if 0, the activator is used as the sole focus)

intensityX/Y/Z: Strength of earthquake, ranging from 0 to 9, along a particular axis.

duration: Duration in tics

damrad: Radius of damage in map units, things that are not on the ground are unaffected.

tremrad: Radius of tremor in map units

sound: Accompanying sound effect for the tremor.

flags: Can be combined with the pipe "|" character. All flags are compatible with each other:

QF_RELATIVE — Adjust the X and Y intensities to quake along the camera view's front and side respectively.

QF_SCALEDOWN — Scales the intensity over the duration, going from full at the start of the quake to 0 upon finishing. Can be combined with QF_SCALEUP.

QF_SCALEUP — Scales the intensity over the duration, going from 0 at the start of the quake to full upon finishing. Can be combined with QF_SCALEDOWN.

QF_WAVE — Changes the quake from a randomly generated one to a sine wave, and are further controlled by mulwavex/y/z. Intensity is known as 'amplitude' in this form.

QF_MAX — Requires QF_SCALEDOWN and QF_SCALEUP. Fully scaled quakes will gradually scale from 0 to half intensity, and back to 0. This changes 0 to start from the defined intensity instead.

QF_FULLINTENSITY — Requires QF_SCALEDOWN and QF_SCALEUP. Fully scaled quakes will only scale in to the half intensity from their origins. This changes half to full intensity instead.

QF_3D — Makes the screen shake of the earthquake also fall off based on how far away the player is on the Z axis from the source of the earthquake.

QF_GROUNDONLY: The screen shake of the earthquake will stop when the player or their camera is off the ground, similar to real world earthquakes. (New from 4.10.0)

QF_AFFECTACTORS: The damrad property will also be able to harm and throw around non-player actors, instead of only affecting players. Non-players can be made immune to the thrusting with DONTTHRUST. (New from 4.10.0)

QF_SHAKEONLY: The earthquakes' damrad property will only be used to throw the player and other actors around, without actually harming them. (New from 4.10.0)

mulwavex/y/z: Only used with QF_WAVE. Specifies the number of waves per second the wave quake goes through while active. Default is 1. These are float values, so precision can be achieved.

falloff: Determines how far away the quake will start to reduce its amplitude based on distance. Takes the same arguments at tremrad in map units. Anything inside this will experience the full effect of the quake. Default is 0, which is no falloff.

highpoint: Only used with QF_SCALE<DOWN/UP>. Determines how far into the quake in tics for the quake to reach the peak of its shaking (or lack thereof if QF_MAX is included). Default is 0, or directly half way.

rollintensity: The camera roll is affected in a similar way to intensityX/Y/Z if specified. Unlike normal intensity, this is not capped. If QF_WAVE is used, can also be negative to allow randomization. This feature does not rely upon the standard intensities and can be used separately (placing 0 in intensity and mulwave properties), but the flags affect it exactly the same.

rollwave: Similar to mulwavex/y/z, but for rolling.

Regular random quakes can stack with wave quakes, but to achieve this effect, one call to QuakeEx with

QF_WAVE specified must happen with another call to the same function without QF_WAVE. This allows for cameras to shake along the wave, as the wave takes priority over where the jittering regular quakes positions. Wave quakes also will pause when the game is paused in any manner. Normally, regular quakes persist through an opened console for example, but wave quakes will always halt when a menu or console pause occurs. Similarly, rolling quakes follow the same behavior.

Contrarily to the Radius_Quake action special, the radii for damage and tremor are given directly in map units, not in "tiles" of 64x64 map units. This must be kept in mind when updating a script from Radius_Quake to QuakeEx.

Return value

Returns false if either the activator is null or there are no actors with the specified tid. Otherwise, it returns true.

Radius_Quake2

void Radius_Quake2 (int tid, fixed intensity, int duration, int damrad, int tremrad, str sound)

Note: As of (development version aed72f5 only) the intensity parameter is a decimal value instead of an integer.

tid: Thing ID of map thing(s) for quake foci (if 0, the activator is used as the sole focus)

intensity: Strength of earthquake, ranging from 0 to 9

duration: Duration in tics

damrad: Radius of damage in map units

tremrad: Radius of tremor in map units

sound: the sound effect to play to accompany the tremor

Usage

Creates an earthquake at the specified foci (map spots).

Contrarily to the Radius_Quake action special, the radii for damage and tremor are given directly in map units, not in "tiles" of 64x64 map units. This must be kept in mind when updating a script from Radius_Quake to Radius_Quake2. Another difference with Radius_Quake is that the tid is the first parameter, rather than the last.

ReplaceTextures

`void ReplaceTextures (str oldtexturename, str newtexturename [, int flags]);`

Usage

Replaces all occurrences of oldtexturename with newtexturename.

Parameters

oldtexturename: the texture being replaced.

newtexturename: the new texture to be applied.

flags: the following flags are supported for this function and can be combined using the logical or operator | :

NOT_BOTTOM: Do not change any bottom textures on walls.

NOT_MIDDLE: Do not change any middle textures on walls.

NOT_TOP: Do not change any top textures on walls.

NOT_FLOOR: Do not change any floor textures.

NOT_CEILING: Do not change any ceiling textures.

Examples

The following script changes the generic floor into blood after 60 tics:

```
script 1 OPEN
```

```
{  
    Delay(60);  
    ReplaceTextures("FLOOR0_1","BLOOD1");  
}
```

If used in a correct way, this feature can be used to scare the player (eg. if the walls turn into a mess of gore after pressing a certain switch).

SectorDamage

`void SectorDamage (int tag, int amount, str type, str protection_item, int flags)`

Usage

Does the damage only when the function is called, and damages everything in the tagged sector.

Parameters

tag: The tag of the affected sector or sectors.

amount: The amount of damage to deal.

type: The damage type; this can either be a built-in damage type or any further custom-defined one.

protection_item: The item that when present in an actor's inventory, will protect said actor from the damage.

Passing an empty string "" as the parameter can be used to not specify an item.

flags: The following flags are supported for this function and can be combined using the bitwise OR operator | :

DAMAGE_PLAYERS — Players in the sector are damaged.

DAMAGE_NONPLAYERS — Shootable non-players in the sector are damaged.

DAMAGE_IN_AIR — Damage actors in the air as well as those on the ground or in the water.

DAMAGE_SUBCLASSES_PROTECT — If an actor is carrying an item derived from the specified protection item, they will be immune to damage. Otherwise, they are only offered protection if they have that exact kind of item.

DAMAGE_NO_ARMOR — Damage ignores armor.

At a minimum, you must specify **DAMAGE_PLAYERS** and/or **DAMAGE_NONPLAYERS**, or nothing will actually receive damage.

Examples

This example kills non-player actors who enter the sector where this script is run. Assign it to run from an Actor Enters Sector thing placed in the sector with the matching tag.

```
script 1 (int tag)
{
    if (PlayerNumber() < 0)
    {
        PrintBold(s:"Kill the non-players!!!");
        Sector_SetFade(tag, 255, 0, 0);

        while (GetActorProperty(0, APROP_HEALTH) > 0)
        {
            SectorDamage(tag, 100, "Fire", "", DAMAGE_NONPLAYERS | DAMAGE_IN_AIR);
            delay(5);
        }

        Sector_SetFade(tag, 0, 0, 0);
    }
}
```

SetAirControl

void SetAirControl (fixed amount);

Usage

Sets how well the player can move while in the air.

Air control is multiplied (using FixedMul) to the default "movement factor" when the player is not on the ground in an area that has gravity, and is not underwater. This gives the player very limited mobility while in the air. The "movement factor" is calculated based on the type of floor the player is on, such as icy or muddy.

Air control is also multiplied to the default bob factor so that the player does not bob up and down as much when in the air.

(Need more info)

Parameters

amount: How much control the player should have while in the air. Amount is a fixed-point value. The default is 0.00390625.

Examples

Gives the player near full air control.

Script 1 OPEN

```
{  
    SetAirControl(1.0);  
}
```

SetCameraToTexture

`void SetCameraToTexture(int cameratid, str texturename, int fov);`

Usage

Binds the named texture to the specified camera (which does not have to be an actual camera thing). From this point on, whatever the camera sees in the specified FOV will be drawn onto the specified texture every frame. Apparently, there is no way to "unbind" the texture, but it can be bound to a different camera.

Camera textures are only recalculated when they are viewed. Therefore, you may make liberal use of camera textures, as long as there are not several on-screen at once.

Note that the texture used must be defined as a "cameratexture" in the ANIMDEFS lump.

Examples

Sample ANIMDEFS entry:

`cameratexture TCAMTEX1 128 64 fit 80 56`

This will make available a texture called "TCAMTEX1", 80 pixels wide and 56 high (but with a slightly higher, horizontally-compressed resolution), that can be applied to any wall or set of walls in a map. Next, a camera thing is put in an interesting spot in the map and is given a TID (7 in this example). Afterwards, paste the following in the map's ACS script:

```
script 1 OPEN {
    SetCameraToTexture( 7, "TCAMTEX1", 90 );
}
```

Now, all surfaces with TCAMTEX1 on them will be video screens showing what the camera with TID 7 sees. One could re-assign the texture to a different camera with a trigger somewhere in the level:

```
script 2 (int camera_tid, int fov) {
    SetCameraToTexture( camera_tid, "TCAMTEX1", fov );
}
```

Triggered by walk-over lines, this would allow to re-use the same camera texture over and over again at different points in the level, possibly saving a bit of memory and lines in ANIMDEFS. Another use is to have a script that periodically changes the screen's view:

```
script 3 OPEN {
    int inter_delay = 70; // 2 seconds between switching
    while( TRUE ) {
        SetCameraToTexture( 7, "TCAMTEX1", 90 );
        Delay( inter_delay );
        SetCameraToTexture( 12, "TCAMTEX1", 90 );
        Delay( inter_delay );
        SetCameraToTexture( 15, "TCAMTEX1", 40 ); // this one's zoomed in!
        Delay( inter_delay );
    }
}
```

SetCeilingTrigger

void SetCeilingTrigger (int tag, int height, int special [, int arg1 [, int arg2 [, int arg3 [, int arg4 [, int arg5]]]])

Usage

When the ceiling specified by tag moves the specified height, special(arg1, arg2, arg3, arg4, arg5) will be activated.

Examples

SetCeilingTrigger (12, 128, acs_execute, 18, 0);

Will execute script 18 (on the current map) when the sector ceiling with tag 12 moves 128 units up (from 128 to 256 or -128 to 0 etc)

SetCeilingTrigger (25, -64, teleport, 15);

Will teleport the script activator to spot 15 when the sector ceiling with tag 25 moves down 64 units (from 0 to -64, 128 to 64 etc)

SetCVar (ACS)

[Jump to navigation](#)[Jump to search](#)

```
bool SetCVar (str cvar, int value);
```

Usage

Sets the value of a particular console variable. Only mod-created console variables defined through CVARINFO can be changed with this function, any built-in engine CVars cannot be.

Parameters

cvar: name of a console variable.

value: value to apply to said console variable.

Return value

Returns FALSE if cvar is invalid, or it is not writable. Returns TRUE if the cvar is set successfully.

SetCVarString

```
bool SetCVarString (str cvar, str value);
```

Usage

Sets the value of a particular console variable. Only mod-defined console variables through CVARINFO can be changed by using this function. Engine's built-in ones cannot be.

Parameters

cvar: name of a console variable.

value: value to apply to said console variable.

Return value

Returns FALSE if cvar is invalid, or it is not writable. Returns TRUE if the cvar is set successfully.

SetFloorTrigger

`void SetFloorTrigger (int tag, int height, int special [, int arg1 [, int arg2 [, int arg3 [, int arg4 [, int arg5]]]])`

Usage

When the floor specified by tag moves the specified height, special(arg1, arg2, arg3, arg4, arg5) is executed.

Examples

`SetFloorTrigger (12, 128, acs_execute, 18, 0);`

Will execute script 18 (on the current map) when the sector floor with tag 12 moves 128 units up (from 128 to 256 or -128 to 0 etc)

`SetFloorTrigger (25, -64, teleport, 15);`

Will teleport the script activator to spot 15 when the sector floor with tag 25 moves down 64 units (from 0 to -64, 128 to 64 etc)

SetFogDensity

void SetFogDensity (int tag, int density)

Warning: This feature is GZDoom specific, and is not compatible with ZDoom!
To see all of GZDoom's specific features, see [GZDoom features](#).

Usage

Sets the density of the fog of the tagged sector. This overrides the default calculation from the light level.

Parameters

tag: the tag of the sector whose fog density is to be set.

density: the density amount of the fog. The higher this value is the more dense the fog is. This ranges from 0 to 510.

SetGravity

`void SetGravity (fixed amount);`

Usage

This function sets the global gravity of an entire level. The values used here are the same as those used by the `sv_gravity` cvar.

Parameters

`amount`: the gravity value to apply, where 800.0 is normal, 400.0 is half, and 1600.0 is double, etc.

Examples

This changes the gravity to 400.0 upon opening of the level:

```
script 2 OPEN
{
    SetGravity(400.0);
}
```

SetLineActivation

void SetLineActivation (int lineid, int activation [, int repeat])

Usage

Sets the line activation flags of the line with the specified ID.

Parameters

lineid: The ID of the line of which to set the activation flags.

activation: The activation flags to set. Multiple flags can be set by using the bitwise OR operator (|) between the constant names:

SPAC_None — No flags.

SPAC_Cross — Activated when crossed by player.

SPAC_Use — Activated when used by player.

SPAC_MCross — Activated when crossed by monster.

SPAC_Impact — Activated when hit by projectile.

SPAC_Push — Activated when bumped by player.

SPAC_PCross — Activated crossed by projectile.

SPAC_UseThrough — Activated when used by player (with pass through).

SPAC_AnyCross — Activated by anything crossing it which does not have the TELEPORT.

SPAC_MUse — Activated by monsters using it.

SPAC_MPush — Activated by monsters bumping into it.

SPAC_UseBack — The line can be used from the back side.

repeat: Whether the line's assigned action special can be activated multiple times or not.

ValueResult

Greater than 0 Can be activated multiple times

Equal to 0 Can only be activated once

Less than 0 No change

Default is -1.

Examples

The following line sets the tagged line so it can be activated by bumping into it:

```
SetLineActivation(1, SPAC_PUSH);
```

Note that the above line also clears all other activation flags, if any, before setting the new flag. If adding a flag without clearing the ones which are already set is what is desired, then using GetLineActivation is needed:

```
SetLineActivation(1, GetLineActivation(1) | SPAC_PUSH);
```

This line clears all activation flags of the tagged line, if any:

```
SetLineActivation(1, SPAC_NONE);
```

SetLineBlocking

Warning: The feature described on this page has been deprecated, and will no longer be supported or updated by the GZDoom developers. While some functionality may be retained for the purposes of backwards-compatibility, authors are strongly discouraged from utilizing this feature in future projects and to instead use `Line_SetBlocking`. Compatibility with future GZDoom versions is not guaranteed.

```
void SetLineBlocking (int lineid, int setting);
```

Usage

Sets the line blocking for a line. This can be set to block nothing, the player and monsters, or everything (including projectiles and hitscans). For readability, here are some definitions defined in `zdefs.acs`:

`BLOCK_NOTHING` — block nothing (turn blocking off)

`BLOCK_CREATURES` — block enemies and the player (walking things)

`BLOCK_EVERYTHING` — nothing can cross this line

`BLOCK_RAILING` — emulates Strife's railings

`BLOCK_PLAYERS` — block only players (but not monsters)

For compatibility with Hexen, you may also use `ON` in place of `BLOCK_CREATURES` and `OFF` in place of `BLOCK_NOTHING`.

Examples

```
script 1 (int lineid)
{
    SetLineBlocking(lineid, BLOCK_EVERYTHING);
    if(CheckInventory("BlueSkull"))
    {
        SetLineBlocking(12, OFF);
    }
}
```

SetLineMonsterBlocking

Warning: The feature described on this page has been deprecated, and will no longer be supported or updated by the GZDoom developers. While some functionality may be retained for the purposes of backwards-compatibility, authors are strongly discouraged from utilizing this feature in future projects and to instead use `Line_SetBlocking`. Compatibility with future GZDoom versions is not guaranteed.

```
void SetLineMonsterBlocking(int lineid, int setting);
```

Usage

Sets blocking for monsters only.

Parameters

lineid: id of the line to set.

setting: this has two settings, 0 for off and 1 for on. For readability, you should use ON or OFF which are defined as 1 and 0 (respectively) in `zdefs.acs`.

Examples

This example will set all lines with the id of 1 to block monsters until some catastrophic thing happens five minutes into the map which allows the monsters to penetrate those lines.

```
script 1 OPEN
```

```
{
    SetLineMonsterBlocking(1, ON);
    delay(35 * 60 * 5);
    SetLineMonsterBlocking(1, OFF);
    HudMessageBold(s:"ZOMG! The monsters can break through now!!!";
        HUDMSG_FADEOUT, 1, CR_RED, 0.5, 0.5, 3.0, 1.0);
}
```

SetLineSpecial

```
void SetLineSpecial (int lineid, int special [, int arg0 [, int arg1 [, int arg2 [, int arg3 [, int arg4]]]]);
```

Usage

SetLineSpecial will change the special on all lines with the line id number specified (assigned by Line_SetIdentification or directly in UDMF).

Using the ZDoom versions of ACC, you can with specify the new special by name instead of number, and you can leave out the arguments you are not interested in. In other words, it is perfectly acceptable to do something like this:

```
SetLineSpecial (1, ACS_Execute, 10);
```

instead of this:

```
SetLineSpecial (1, 80, 10, 0, 0, 0, 0);
```

The first version is more readable and less error-prone, so you should always opt for that version of the command instead of using special numbers.

Parameters

lineid: id of the line to set.

special: action special to set on the line.

arg0-arg4: arguments passed to the special when executed.

Examples

This example will be triggered by script 1 and will cause all lines with line id 9 to execute script 10.

script 1 (void)

```
{
    Print (s:"setting action");
    ActivatorSound("misc/chat", 127);

    SetLineSpecial (9, ACS_Execute, 10);
}
```

script 10 (void)

```
{
    ActivatorSound("misc/chat", 127);
}
```


SetLineTexture

```
void SetLineTexture(int lineid, int line_side, int sidedef_texture, str texturename);
```

Usage

SetLineTexture will change the specified texture on all lines with the line id number specified (assigned by Line_SetIdentification or directly in UDMF). For readability, line_side and sidedef_texture have some definitions in zdef.acs:

SIDE_FRONT — front of the linedef (front sidedef)

SIDE_BACK — back of the linedef (back sidedef)

TEXTURE_TOP — upper texture of sidedef

TEXTURE_MIDDLE — middle texture of sidedef

TEXTURE_BOTTOM — lower texture of sidedef

Parameters

lineid: the line id of the lines to change.

line_side: the side of the line to change.

sidedef_texture: which texture to change.

texturename: the texture that will be set on the line. Using "-" means to remove a texture.

Examples

```
script 1 (int line1, int line2, int line3)
```

```
{
    SetLineTexture(line1, SIDE_FRONT, TEXTURE_MIDDLE, "-"); //remove middle
    SetLineTexture(line1, SIDE_BACK, TEXTURE_MIDDLE, "-"); //floating texture

    SetLineTexture(line2, SIDE_BACK, TEXTURE_TOP, "BFALL1");
    SetLineTexture(line3, SIDE_BACK, TEXTURE_BOTTOM, "FOOTEX01");
}
```

SetSectorDamage

void SetSectorDamage (int tag, int amount [, string damagetype [, int interval [, int leaky]]])

Usage

Sets a sector's damage properties. This is a variant of the Sector_SetDamage special that can take actual damage types as a parameter. Damage settings from regular Doom sector types or UDMF map settings will be overridden by this.

Parameters

tag: Tag of the affected sector.

amount: Amount of damage to apply.

damagetype: Damage type to inflict. Default is "None".

interval: Time between two inflictions of damage in tics. Default is 32.

leaky: Probability of a radiation suit "leaking" damage. A value of 0 means it does not leak while a value of 256 means that the suit has no effect at all. Doom's 20% damage sector type uses a value of 5. Default is 0.

SetSectorGlow

void SetSectorGlow (int tag, bool plane, int red, int green, int blue, int height)

Warning: This feature is GZDoom specific, and is not compatible with ZDoom!
To see all of GZDoom's specific features, see GZDoom features.

Usage

Applies a glow effect on the floor or ceiling of the tagged sector.

Parameters

tag: the tag of the sector on which to apply the effect.

plane: if true, the glow is applied on the ceiling, otherwise (i.e. if false) it is applied on the floor.

red: the red compound of the color. If this is set to -1, the glow effect is removed (the other color compounds are ignored).

green: the green compound of the color.

blue: the blue compound of the color.

height: the height of the glow.

Examples

This script applies a red glow on the floor and a blue one on the ceiling.

```
script "Init" OPEN
{
    SetSectorGlow(1, false, 255, 0, 0, 64);
    SetSectorGlow(1, true, 0, 0, 255, 64);
}
```

SetSectorTerrain

void SetSectorTerrain (int tag, int whichplane, string terraintype)

Usage

Sets a sector's terrain type independently of its texture. If set, this will take precedence.

Parameters

tag: Tag of the affected sector.

whichplane: The plane to apply the terrain on. 0 for floor and 1 for ceiling. Ceiling terrain of the model sector is used for the top of 3D floors.

terraintype: The terrain type to apply.

SetSkyScrollSpeed

`void SetSkyScrollSpeed (int sky, fixed skyspeed);`

Usage

Changes the scrolling speed of a sky. This is useful in conjunction with `ChangeSky`.

Parameters

`sky`: either 1 or 2.

`skyspeed`: the desired scrolling speed.

Examples

`script 1 ENTER`

```
{  
  SetSkyScrollSpeed (1, 155); // Or 0.002365, since the speed is a fixed point value  
}
```

SetUserCVar (ACS)

```
bool SetUserCVar (int playernumber, str cvar, int value);
```

Usage

Sets the console variable of a particular player. Only mod-defined console variables through CVARINFO can be changed by using this function. Engine's built-in ones cannot be.

Parameters

player: player number of the player .

cvar: name of a console variable.

value: value to apply to said console variable.

Return value

Returns FALSE if cvar is invalid, it is not writable, or the player doesn't exist. Returns TRUE if the cvar is set successfully.

SetUserCVarString (ACS)

`bool SetUserCVarString (int playernumber, str cvar, str value);`

Usage

Sets the console variable of a particular player. Only mod-defined console variables through CVARINFO can be changed by using this function. Engine's built-in ones cannot be.

Parameters

player: player number of the player .

cvar: name of a console variable.

value: value to apply to said console variable.

Return value

Returns FALSE if cvar is invalid, it is not writable, or the player doesn't exist. Returns TRUE if the cvar is set successfully.

SpawnParticle

`void SpawnParticle (int color [, bool fullbright [, int lifetime [, int size [, fixed x [, fixed y [, fixed z [, fixed velx [, fixed vely [, fixed velz [, fixed accelx [, fixed accely [, fixed accelz [, int startalpha [, int fadeStep]]]]]]]]]]]]];`

Usage

Spawns a single particle at the x, y and z coordinates.

Parameters

color: The color of the particle. This is a hexadecimal value, e.g. 0xFF0000 (which is bright red).

fullbright: If true, it renders the particle fully bright. Default is false.

lifetime: The lifetime of the particle in tics. This ranges from 0 to 255. Default is 35.

size: The size of the particle, up to a maximum value of 65535. Default is 1.

x/y/z: The absolute map coordinates at which to spawn the particle. Default is 0.

velx/vely/velz: The velocity along the x/y/z axis. This is in absolute direction, not relative. Default is 0.

accelx/accely/accelz: Defines how much to accelerate this particle by over its lifespan. Default is 0.

startalpha: Specifies the alpha upon spawning. This ranges from 0 to 255. Default is 255.

fadeStep: The amount the particle fades each tic. This ranges from -1 to 255, with -1 indicating automatic mode (a complete fadeout over the length of lifetime). The particle is automatically removed early if it fades completely before lifetime expires. Default is -1.

Examples

This a simplified version of the script that spawns the yellow "portal particles" at the start of Map01 of Eviternity. The particles are spawned randomly across the width and height of a linedef and slowly fade out.

Script "Eviternity_PortalParticles" ENTER

```
{
  SpawnParticle (0xFFE74B, 1, Random(30,45), 7 , Random(-184.0,-72.0), -736.0, Random(8.0, 120.0), 0.0, 3.0, 0.0,
0.0, 0.0, 0.0, 1.0, -1);
  Delay(1);
  Restart;
}
```


Ceil

fixed Ceil (fixed value)

Usage

Rounds the specified value to the highest whole number.

Return value

The value after rounding, as a fixed-point number.

Custom function

Note: The following is a non-native implementation of the function, for versions of GZDoom older than 2.4.0.
Map a fixed point value to the highest whole number.

```
function int ceil (int x)
{
    return (x + 65535) & 0xFFFF0000;
}
```

Cos

fixed Cos (int angle)

Usage

Returns the value for the cosine of angle.

Parameters

angle: The fixed point angle value to compute the cosine for.

Return value

Returns the fixed point value of the cosine.

Example

This script spawns two Medikits on the left and right side of the activator when activated.

script 1 (void)

```
{
    int x = GetActorX (0);
    int y = GetActorY (0);
    int z = GetActorZ (0) + 32.0;
    int angle = GetActorAngle (0);

    Spawn ("Medikit", x + cos (angle + 0.25) * 32, y + sin (angle + 0.25) * 32, z);
    Spawn ("Medikit", x + cos (angle + 0.75) * 32, y + sin (angle + 0.75) * 32, z);
}
```

FixedDiv

fixed FixedDiv (int a, int b)

Usage

Returns the fixed point result of dividing fixed point a by fixed point number b.

Parameters

a: The fixed point dividend.

b: The fixed point divider.

Return value

The result of fixed point numbers dividing.

Examples

You cannot use normal division operator on fixed point numbers when you expect the result to be fixed point number too. See the results of this example:

script 1 (void)

```
{
    Print (f: FixedDiv (1.0, 0.5)); // 2
    Print (f: 1.0 / 0.5);           // 0.000030518
}
```

However, dividing a fixed point number a by an integer b yields a fixed point value still, and FixedDiv must not be used in this case:

script 2 (void)

```
{
    int z = 1.0 / 3;
    print(f:z);           /* 0.333 */
}
```

FixedMul

fixed FixedMul (int a, int b)

Usage

Returns the fixed point result of multiplying two fixed point numbers.

Parameters

a, b: The fixed point numbers to multiply.

Return value

The result of fixed point numbers multiplication.

Examples

You cannot use normal multiplication operator on fixed point numbers. See the results of this example:

script 1 (void)

```
{  
    Print (f: FixedMul (0.5, 0.5)); // 0.25  
    Print (f: 0.5 * 0.5);          // 16384  
}
```

However, multiplying a fixed point number a by an integer b yields a fixed point value still, and one must not use FixedMul in this case:

script 2 (void)

```
{  
    int z = 1.2 * 3;  
    print(f:z);          /* 3.6 or so */  
}
```

Floor (ACS function)

fixed Floor (fixed value)

Usage

Rounds the specified value to the lowest whole number.

Return value

The value after rounding, as a fixed-point number.

Random

`int Random (int min, int max)`

Usage

Returns a random integer between min and max (inclusive).

Parameters

min: The minimum value to return.

max: The maximum value to return.

Return value

A random integer between min and max (inclusive).

Examples

This script damages the activator with damage from 1 to 10 when activated:

```
script 1 (void)
```

```
{  
    DamageThing (Random (1, 10));  
}
```

You can also use fixed point numbers, this script causes the activator to face a random direction:

```
script 1 (void)
```

```
{  
    SetActorAngle (0, Random (0, 1.0));  
}
```

Round

fixed Round (fixed value)

Usage

Rounds the specified value to the nearest whole number.

Return value

The value after rounding, as a fixed-point number.

Custom function

Note: The following is a non-native implementation of the function, for versions of GZDoom older than 2.4.0.

This function implements rounding. When performing the bitshift operation on a fixed point number, what you are actually doing is simply cutting off the end of the number, the part that is less than 1. So $0.5 \gg 16$ will return 0. There may be cases where you want to round values of 0.5 or greater to the nearest whole number instead when converting it to an integer, for example placing an absolute location for a hudmessage.

If you need to round a number to the nearest whole but need the resulting number as a fixed point, you can just bitshift it again in the opposite direction using `round(number) << 16`.

```
// Returns integer value
```

```
function int round(int fixedNumber)
```

```
{  
    return (fixedNumber + 0.5) >> 16;  
}
```

Cut off decimal part version

```
// Returns fixed point value
```

```
function int floor(int fixedNumber)
```

```
{  
    return fixedNumber & 0xFFFF0000;  
}
```

Sin

fixed Sin (int angle)

Usage

Returns the value for the sine of angle.

Parameters

angle: The fixed point angle value to compute the sine for.

Return value

Returns the fixed point value of the sine.

Example

This script spawns two Medikits on the left and right side of the activator when activated.

script 1 (void)

```
{
    int x = GetActorX (0);
    int y = GetActorY (0);
    int z = GetActorZ (0) + 32.0;
    int angle = GetActorAngle (0);

    Spawn ("Medikit", x + cos (angle + 0.25) * 32, y + sin (angle + 0.25) * 32, z);
    Spawn ("Medikit", x + cos (angle + 0.75) * 32, y + sin (angle + 0.75) * 32, z);
}
```


Sqrt

int Sqrt (int number)

fixed FixedSqrt (fixed number)

Usage

Returns the square root of an integer or fixed point number.

Parameters

number: Value to take the square root of.

Return value

Either the rounded integer or fixed point square root of the number.

Manual calculation

In older versions of ZDoom, the function is not built into ACS so here are several that work fine. Do note that for the distance between to objects the distance function can be faster.

This version uses Newton's method, where the solution converges quadratically from the initial guess. This function has been compared with the following two, and is at least 5 to 10 times faster in normal use.

```
function int sqrt(int number)
{
    if(number <= 3) { return number > 0; }

    int oldAns = number >> 1,          // initial guess
        newAns = (oldAns + number / oldAns) >> 1; // first iteration

    // main iterative method
    while(newAns < oldAns)
    {
        oldAns = newAns;
        newAns = (oldAns + number / oldAns) >> 1;
    }

    return oldAns;
}
```

This is a simpler, but slower, version of a square root function. It rounds the root up or down.

```
function int sqrt (int x)
{
    int r;
    x = x + 1 >> 1;
    while (x > r)
        x -= r++;
    return r;
}
```

If you need a function that rounds down, this one will work. Based off of this algorithm.

```
function int isqrt (int n)
{
    int a;
    for (a=0;n>=(2*a)+1;n-=(2*a++)+1);
```

```

    return a;
}

```

This sample-based formulae processes fixed values and returns a fixed result which accuracy is defined by the samples integer, and results zero if the input number is negative.

```

function int sqrt(int number)
{
    int samples=15; //Samples for accuracy

    if (number == 1.0) return 1.0;
    if (number <= 0) return 0;
    int val = samples<<17 + samples<<19; //x*10 = x<<1 + x<<3
    for (int i=0; i<samples; i++)
        val = (val + FixedDiv(number, val)) >> 1;

    return val;
}

```

This is the same as above but is optimised for 15 samples

```

function int sqrt(int number)
{
    if (number == 1.0) return 1.0;
    if (number <= 0) return 0;
    int val = 150.0;
    for (int i=0; i<15; i++)
        val = (val + FixedDiv(number, val)) >> 1;

    return val;
}

```

StrLen

int StrLen (str string)

Usage

Returns the length of the string specified by string. Note that all strings in ACS are static, i.e. they are not created on the fly, so it is not really necessary to use this function unless you have a lot of strings stored in a library somewhere – like in Daedalus – and you need to have proper delays for a HudMessage. The existence of StrParam makes this function more useful, however.

Parameters

string: The string whose length we want to know.

Return value

Length (the count of the characters) of the given string.

Examples

This script prints a string and its length:

```
script 1 (void)
```

```
{  
    str thestring = "This is the string."  
  
    Print (s:thestring, i:StrLen (thestring));  
}
```

VectorAngle

fixed VectorAngle (int x, int y)

Usage

Returns the fixed point angle of the vector (x,y). Angles are measured from the east and moving counterclockwise.

This function is more commonly known as atan2. To get the value of atan(x), use VectorAngle(1.0, x)

Parameters

x, y: Coordinates of the end point of the vector.

Return value

Fixed point angle of the vector (x,y).

Examples

This script will print a little ^ at the bottom of the player's screen pointing at the actor with TID set to 1:

script 1 ENTER

```
{
    int vang, angle;
    while(TRUE)
    {
        vang = VectorAngle (GetActorX (1) - GetActorX (0), GetActorY (1) - GetActorY (0));
        angle = (vang - GetActorAngle (0) + 1.0) % 1.0;

        if (angle < 0.2 || angle > 0.8)
        {
            int sx = 320 - (320 * Sin (angle) / Cos (angle));

            SetHudSize (640, 480, 0);
            HudMessage (s:"^"; HUDMSG_PLAIN, 1, CR_RED, sx * 1.0, 480.2, 0);
        }
        else
        {
            HudMessage (s:""; HUDMSG_PLAIN, 1, 0, 0, 0, 0);
        }

        Delay (1);
    }
}
```

VectorLength

`int VectorLength (int x, int y)`

Usage

Returns the length of the vector (x,y).

Parameters

x, y: Coordinates of the end point of the vector.

Return value

Length of the vector (x,y).

Examples

Here is a quick example to use it with three dimensions:

```
function int VLength3d(int x, int y, int z)
{
    int len = VectorLength(x, y);
    len = VectorLength(z, len);
    return len;
}
```

Break

```
break;
```

Usage

break is used to exit from a (given type of) block of code early, and is most commonly used to break out of the current scope of a do, for, or while statement, or to break out of a switch block completely.

Examples

This example breaks out of the for loop if the matching player is not currently in the game.

```
for (int i = 0; i < 8; i++)
{
    if (!PlayerInGame (i))
        break;

    TeleportOther (1000 + i, 60 + i, 1);
}
```

This example breaks out of the while loop if the player has the radiation suit.

```
while (PlayerDrugged)
{
    delay (35);

    if (CheckInventory ("RadSuit"))
        break; // Radiation suit protects against drugged effect

    FadeTo (random (0, 2) * 128, random (0, 2) * 128, random (0, 2) * 128, 1.0, 1.0);
}
```

This example uses the break statement multiple times in a select block. It is important to remember to use break at the end of each case to avoid execution “falling through” to the next case block.

```
switch (GameSkill ())
{
    case SKILL_VERY_EASY:
        // Spawn one zombie. Pathetically easy.
        SpawnSpot ("ZombieMan", 60);
        break;

    case SKILL_EASY:
        // Spawn one imp. Still really easy.
        SpawnSpot ("DoomImp", 60);
        break;

    case SKILL_HARD:
        // Spawn a baron, in addition to threeimps.
        SpawnSpot ("BaronOfHell", 60);
        // break is intentionally not used here, to allow execution to continue through the next block.

    case SKILL_NORMAL:
```

```
// Spawn three imps.  
SpawnSpot ("DoomImp", 61);  
break;  
  
case SKILL_VERY_HARD:  
    // Spawn a CYBERDEMON!  
    SpawnSpot ("Cyberdemon", 60);  
    break;  
}
```

Continue

`continue;`

Usage

`continue` is used in a `for`, `do`, or `while` loop to end the current iteration, returning to the start of the loop.

Examples

This script ignores any players that autoaim, and logs who doesn't.

```
for (int i = 0; i < 8; i++)
{
    if (GetPlayerInfo (i, PLAYERINFO_AIMDIST) > 0 )
        continue;

    Log (n:i+1, s:"\c- does not autoaim");
}
```



```
restart
```

```
restart;
```

Usage

restart is a function in ACS to restart a running script. You will need to have at least one delay in the script or you will get the runaway error, unless it only executes a certain number of times.

Examples

This script will give the player a health bonus every 200 tics.

Script 1 ENTER

```
{  
  GiveInventory("HealthBonus",1);  
  Delay(200);  
  restart;  
}
```

ScriptCall

ScriptCall (str classname, str funcname, args)

Usage

Executes a script-defined function. If the first argument of the function to execute is of type Actor, the activator of the script is passed and is stored in that argument. This allows the activator to be accessed by said function.

Parameters

classname: the name of the actor class in which the function is defined.

funcname: the name of the function to execute. Only static functions are executable.

args: the arguments, if any, to pass to the specified function. The types of arguments the function could have are int, bool, double, string, name, color and sound. double is passed as a fixed-point value, color as an integer value, and name and sound as strings.

Return value

The value the executed function returns. The value's type is the same as the passed arguments' types, in addition to void.

Examples

This script gives the player a unique TID. This is achieved by executing FindUniqueTid, which is defined in Actor.

```
script "SetPlayerTID" ENTER
{
    // Only if the player does not already have a TID.
    if(ActivatorTID() == 0)
    {
        int ptid = ScriptCall("Actor", "FindUniqueTid", 0, 0);
        Thing_ChangeTID(0, ptid);
        Log(s:"TID changed to ", d:ptid); // Print a message.
    }
}
```

Note that ACS already has a function which can get a unique TID. The above example is for educational purposes.

If executed, this GivePresent function gives the activator of the script an item and logs a message. The item is specified by the present argument.

// In ZScript:

```
class ExampleActor
{
    static void GivePresent (Actor activator, string present)
    {
        // Only players get the item.
        if(activator && activator.player)
        {
            activator.A_GiveInventory(present);
            activator.A_Log("You received a wonderful present!", true);
        }
    }
}
```

// In ACS:

```
script "GetGift" (void)
{
```

```
ScriptCall("ExampleActor", "GivePresent", "Soulsphere");  
}
```

Suspend

suspend;

Usage

Suspend is a function used in ACS to suspend the script it is used within. You can also use ACS_Suspend to suspend other scripts.

Suspend is essentially the same as terminate, except that it leaves a marker in memory which instructs the script to pick up where it left off the next time it is run. After a script has been suspended, activating the script again by any means will resume the script from the point it was last suspended.

Examples

This script will update the texture on a wall every time the player uses the wall. It will toggle between three different possibilities. The script could easily be adapted to a computer terminal which allows the user to “page” between multiple displays or messages.

```
script 1 (void) {  
    SetLineTexture (60, SIDE_FRONT, TEXTURE_MIDDLE, "SCREEN2");  
    suspend;  
  
    SetLineTexture (60, SIDE_FRONT, TEXTURE_MIDDLE, "SCREEN3");  
    suspend;  
  
    SetLineTexture (60, SIDE_FRONT, TEXTURE_MIDDLE, "SCREEN1");  
}
```

The first activation of this script will change the texture to “SCREEN2” and suspend the script. The second activation will resume the script with the next line, which changes the texture to “SCREEN3” and suspends the script again. The third activation will change the line back to “SCREEN1” and the script will exit normally, so that the next time the script is run, it will start over from the beginning.

Terminate

```
terminate;
```

Usage

Terminate is a function used in ACS to end the current script early. You can also use ACS_Terminate to terminate other scripts.

Example

This script will terminate itself if the player has not collected all the keys yet.

```
int keys = 0; // See Scope
```

```
script 1 (void)
```

```
{
```

```
    keys++;
```

```
    if (keys < 3)
```

```
        terminate;
```

```
    // All keys collected, open the door.
```

```
    Door_Open (24, 16, 0);
```

```
}
```

Note that there is no need to add the terminate command to the very end of a script. (After the Door_Open line in this example) When the closing brace is reached, the script will terminate itself automatically.

ActivatorSound

Note: This function has been superseded by PlaySound, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

```
void ActivatorSound(str sound, int volume);
```

Usage

Plays a sound from whoever activated the script (be it a monster, the player or projectile).

Parameters

sound: The sound to be played, as defined in SNDINFO.

volume: An integer range from 0 to 127, with 127 being full volume and 0 being muted.

Examples

```
script 4 (void) {  
    ActivatorSound("misc/i_pkup",127);  
    print(s:"Sounds like you got something!");  
}
```

AmbientSound

This page is for the ACS function. You might be looking for the actor instead.

```
void AmbientSound(str sound, int volume);
```

Usage

Plays a world sound (all players can hear it at the same volume, regardless of how close to the activator they are).

Volume is an integer range from 0 to 127, with 127 being full volume and 0 being muted.

Examples

This script waits for all monsters tagged with tid 1 to die, and then plays a victory fanfare sound which is audible to all players, regardless of where they are.

```
script 1 (void)
```

```
{  
    // Wait for all tid 1 monsters to die  
    while( ThingCount(T_NONE, 1) > 0 )  
    {  
        delay(4);  
    }  
  
    // Play a victory noise for all players at full volume  
    AmbientSound("handel/messiah", 127);  
}
```

LocalAmbientSound

```
void LocalAmbientSound(str sound, int volume);
```

Usage

Plays a sound at world volume (can be heard at the same volume no matter where the player stands) that is only heard by the activator of the script. Volume is an integer ranging from 0 to 127, with 127 being full volume and 0 being muted.

Examples

Script 1 ENTER

```
{  
  LocalAmbientSound("QTalk", 127); //Plays a sound at full volume that is only heard by the activator  
  Print(s:"Welcome to Hell");  
}
```


LocalSetMusic

`void LocalSetMusic (str song [, int order [, int unused]])`

Usage

LocalSetMusic changes the music in the game, but unlike SetMusic, only affects the player who activated the script. If song refers to a tracker module (MOD), then order specifies the so-called order in the song to start playing at; otherwise, order is ignored. order is optional, so you do not need to specify it if you are not using a MOD.

Additionally, you may also specify "*" instead of the name of a song, and the level's default music as defined in MAPINFO will be played.

Note that the third parameter is currently defined but not used by ZDoom. It is recommended that you do not specify the third parameter at all (as in the examples below), as this parameter may be used in the future.

Examples

This command changes the currently-playing music to "D_DOOM" for the script activator:

```
LocalSetMusic("D_DOOM");
```

This command returns the music track to the default specified in MAPINFO:

```
LocalSetMusic("*");
```

This command changes the music to the 2nd order of the specified MOD:

```
LocalSetMusic("MYMOD", 2);
```

PlayActorSound

`void PlayActorSound (int tid, int sound, int channel, fixed volume, bool looping, fixed attenuation);`

Usage

Plays a sound at a thing. This is similar to `PlaySound` except it takes a sound identifier as the sound parameter instead of an explicit sound, and thus the sound that is eventually played depends on the actor's defined sounds. If `tid` is 0, the sound will be played by the activator of the script.

Parameters

`tid`: The tid of the actor to play the sound from.

`sound`: The sound identifiers to play the sound they reference:

`SOUND_See` — Plays the `SeeSound` of the actor.

`SOUND_Attack` — Plays the `AttackSound`.

`SOUND_Pain` — Plays the `PainSound`.

`SOUND_Death` — Plays the `DeathSound`.

`SOUND_Active` — Plays the `ActiveSound`.

`SOUND_Use` — Plays the `Inventory.UseSound`. (Verification needed)

`SOUND_Bounce` — Plays the `BounceSound`.

`SOUND_WallBounce` — Plays the `WallBounceSound`.

`SOUND_CrushPain` — Plays the `CrushPainSound`.

`SOUND_Howl` — Plays the `HowlSound`.

`SOUND_Push` — Plays the `PushSound`. (New from 4.10.0)

`channel`: The channel to play the sound on. Default is `CHAN_BODY`.

`volume`: The volume of the sound. This ranges from 0.0 (mute) to 1.0 (full volume). Default is 1.0.

`looping`: Whether the sound is to be looped or not. If `TRUE`, the sound loops, otherwise it does not. Default is `FALSE`.

`attenuation`: This is how quickly the sound fades with distance from its source. The higher the value the quicker the sound fades out. The following are the standard attenuation values:

`ATTN_NONE` — 0 (no attenuation; the sound can be heard regardless of the distance).

`ATTN_NORM` — 1.0 (normal attenuation; this is the default value).

`ATTN_IDLE` — 1.001

`ATTN_STATIC` — 3.0 (the sound diminish very rapidly with distance).

PlaySound

```
void PlaySound (int tid, str sound [, int channel [, fixed volume [, bool looping [, fixed attenuation [, bool local]]]])
```

Usage

Plays a sound at a thing. This is similar to A_PlaySound action function except it will not default to playing "weapons/pistol" sound. If tid is 0, the sound will be played by the activator of the script.

Parameters

tid: the TID of the actor to play the sound from.

sound: the sound to play, as defined in SNDINFO.

channel: the channel to play the sound on. Default is CHAN_BODY.

volume: the volume of the sound. This ranges from 0.0 (mute) to 1.0 (full volume). Default is 1.0.

looping: whether the sound should loop or not. If true, the sound loops, otherwise it does not. Default is false.

attenuation: how quickly the sound fades with distance from its source. The higher the value the quicker the sound fades out. The following are the standard attenuation values:

ATTN_NONE — 0 (no attenuation; the sound can be heard regardless of the distance).

ATTN_NORM — 1.0 (normal attenuation; this is the default value).

ATTN_IDLE — 1.001

ATTN_STATIC — 3.0 (the sound diminish very rapidly with distance).

local: If true, the sound is played if the player is either looking out the eyes of the actor from which the sound is coming, or said actor is the player and the player is looking out the eyes of a non-monster actor. Also, the sound is played with ATTN_NONE. Default is false.

SectorSound

```
void SectorSound(str sound, int volume);
```

Usage

Plays a sound in the sector that the line the script is attached to faces. This is a point sound, so anyone far away will not hear it as loudly. Volume is an integer range from 0 to 127, with 127 being full volume and 0 being muted.

Examples

This script generates a creaking sound when walking over a linedef into a sector.

```
script 1 (void)
```

```
{  
    SectorSound("world/creak1", 127);  
}
```

To make this all work, be sure you have your linedef action special set to 80 (ACS_Execute), the argument pointed to the correct script number, and activation trigger set to "Player Crosses Line".

SetMusic

```
void SetMusic (str song [, int order [, int unused]]);
```

Usage

SetMusic changes the music in the game.

Parameters

song:

The name of a music file to be played.

If song contains a /, then it is assumed to be a full path in a PK3 including music/ and the file extension. The file name may be longer than 8 characters.

If song is "*", then the default music as defined in MAPINFO will be played.

order:

If song refers to a tracker module (MOD), then order specifies the so-called order in the song to start playing at.

If song refers to a multi-track music file, then order is the track number belonging to the song you want to play.

If song is not a tracker module or a multi-track file, then order can be omitted.

unused: This parameter is currently defined but not used by ZDoom. It is recommended that you do not specify the third parameter at all.

Examples

```
Script 100 (int tid) // Boss battle
```

```
{
    SetMusic("BosFight", 0);

    ACS_Execute(666, 0, tid, 0, 0); // Starts a health tracker for the boss

    while (ThingCount(T_NONE, tid) > 0) delay(35);

    SetMusic("*", 0); // Restore the level's default music
}
```

SetMusicVolume

void SetMusicVolume (fixed volume);

Usage

Sets the sound volume of the music currently playing, as a multiplier.

Parameters

volume: The volume multiplier to set. This ranges from 0 to 2.0, with values higher than 1.0 only working if the global volume is low enough so that the end result is not more than 1.0.

Examples

This script fades the current music out, and when it is finished fading it, it changes the music and fades that music in.

```
int musicVolume = 1.0;
```

Script 1 (void)

```
{
    while(musicVolume > 0)
    {
        musicVolume -= 0.025;
        SetMusicVolume(musicVolume);
        Delay(1);
    }

    SetMusic("D_Shawn");

    while(musicVolume < 1.0)
    {
        musicVolume += 0.025;
        SetMusicVolume(musicVolume);
        Delay(1);
    }
}
```

SoundSequence

`void SoundSequence (str sndseq);`

Usage

Plays a sound sequence defined in SNDSEQ lump.

Examples

First off, you have to make sure you have got a sequence set up in the SNDSEQ lump, for example:

```
:Heartbeat
  volume 127
  playuntildone world/heart
  volume 64
  playuntildone world/heart
end
```

Just a simple sequence that plays a loop of a beating heart “ first time at full volume and the second time at half volume.

To have this play in your level, you need to call it in a script. A very, very simple and easy to use script.

Script 1 (void)

```
{
  SoundSequence ("Heartbeat");
}
```

The script can be executed by having a linedef with special 80 (ACS_Execute) and the script number as argument; the linedef itself be set to “œPlayer Crosses Lineœ” .

SoundSequenceOnActor

```
void SoundSequenceOnActor (int tid, str sndseq);
```

Usage

Plays a sound sequence defined in SNDSEQ lump.

Examples

First make sure that you have defined in a SNDINFO lump in your mod the sounds that will be used by your sequence, for example:

```
world/machinery1      A_AMB1
```

Here A_AMB1 is the name of a sound file included in the mod.

Then define a sound sequence in a SNDSEQ lump. For example:

```
:MachineryLoop1  
  playrepeat world/machinery1  
end
```

You can then easily use the sound sequence in your ACS script to make a sound come from a 'thing' on your map. For example, using a map spot with its tag set to 1, you could use this line:

```
SoundSequenceOnActor (1, "MachineryLoop1");
```


SoundSequenceOnPolyobj

```
void SoundSequenceOnPolyobj (int polynum, str sndseq);
```

Usage

Plays a sound sequence defined in SNDSEQ lump. Note that a polyobject can play only a single sequence at a time: if the same script uses SoundSequenceOnPolyobj followed by an instruction to move or rotate the polyobject, the sound sequence will be overridden by the one from the movement (even if it is silent or undefined). Instead, SoundSequenceOnPolyobj should be called after the movement instruction.

SoundSequenceOnSector

`void SoundSequenceOnSector (int tag, string seqname, int location);`

Usage

Plays a sound sequence defined in SNDSEQ lump. The location parameter determines from where exactly the sound comes. Possible values are the following constants:

SECSEQ_FLOOR: From the floor and below

SECSEQ_CEILING: From the ceiling and up

SECSEQ_FULLHEIGHT: At any height from the sector

SECSEQ_INTERIOR: Between the floor and ceiling

SoundVolume

`void SoundVolume (int tid, int channel, fixed volume)`

Usage

Changes the volume of the currently playing sound by the actor(s) with the specified TID. If tid is 0, the sound volume change is done to the sound being played by the activator of the script.

Parameters

tid: The tid of the actor to change its sound volume.

channel: The channel of the playing sound.

volume: The new sound volume value to set. This ranges from 0.0 (mute) to 1.0 (full volume).

StopSound

```
void StopSound (int tid, int channel);
```

Usage

Stops the sound currently playing on the specified channel for the actor with matching tid. This is similar to the A_StopSound action function except that the default channel is CHAN_BODY. If tid is 0, this stops the sound currently playing by the activator of the script.

Parameters

tid: The tid of the actor to stop its sound.

channel: The channel on which the sound is to be stopped. Default is CHAN_BODY.

ThingSound

Note: This function has been superseded by PlaySound, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

```
void ThingSound (int tid, str sound, int volume);
```

Usage

Plays a sound at a thing. This is a point sound, so anyone far away will not hear it as loudly.

Parameters

tid: TID of the actor.

sound: The sound to be played, as defined in SNDINFO.

volume: An integer range from 0 to 127, with 127 being full volume and 0 being muted.

Examples

This script shuts down some kind of generator in the level. The sound source is an object close to the generator so when the player hits the switch to turn it off, they hear the powering down sound coming from the generator.

```
script 1 (void)
{
    // Message to the player
    Print(s:"Generator shut down.  Rear hatch access granted.");

    // Turn off the generator lights
    Light_ChangeToValue(1, 144);
    Light_ChangeToValue(2, 96);

    // Play a shutdown sound from the generator
    ThingSound(1, "ambient/poweroff", 127);
}
```

GetLineUDMFFixed

fixed GetLineUDMFFixed (int lineid, string key)

Usage

Retrieves a value for a custom user key set for a line.

Parameters

lineid

The tag of the line. A value of 0 uses the activating line.

key

The name given to the user key.

Return value

The fixed point value that was set for this user key.

GetLineUDMFInt

int GetLineUDMFInt (int lineid, string key)

Usage

Retrieves a value for a custom user key set for a line.

Parameters

lineid

The tag of the line. A value of 0 uses the activating line.

key

The name given to the user key.

Return value

The int value that was set for this user key.

GetSectorUDMFFixed

fixed GetSectorUDMFFixed (int tag, string key)

Usage

Retrieves a value for a custom user key set for a sector.

Parameters

tag

The tag of the sector.

key

The name given to the user key.

Return value

The fixed point value that was set for this user key.

GetSectorUDMFInt

int GetSectorUDMFInt (int tag, string key)

Usage

Retrieves a value for a custom user key set for a sector.

Parameters

tag

The tag of the sector.

key

The name given to the user key.

Return value

The int value that was set for this user key.

GetSideUDMFFixed

fixed GetSideUDMFFixed (int lineid, bool side, string key)

Usage

Retrieves a value for a custom user key set for a line.

Parameters

lineid

The tag of the sidedef's line.

side

If 0, the front side of the line; if 1, the back side. The SIDE_FRONT and SIDE_BACK constants can also be used.

key

The name given to the user key.

Return value

The fixed point value that was set for this user key.

GetSideUDMFInt

int GetSideUDMFInt (int lineid, bool side, string key)

Usage

Retrieves a value for a custom user key set for a line.

Parameters

lineid

The tag of the sidedef's line.

side

If 0, the front side of the line; if 1, the back side.

key

The name given to the user key.

Return value

The int value that was set for this user key.

GetThingUDMFFixed

fixed GetThingUDMFFixed (int thingid, string key) (Planned feature)

Usage

Retrieves a value for a custom user key set for a thing. (Not implemented yet)

Parameters

thingid

The TID of the thing.

key

The name given to the user key.

Return value

The fixed point value that was set for this user key.

GetThingUDMFInt

int GetThingUDMFInt (int thingid, string key) (Planned feature)

Usage

Retrieves a value for a custom user key set for a thing. (Not implemented yet)

Parameters

thingid

The TID of the thing.

key

The name given to the user key.

Return value

The int value that was set for this user key.

ACS_ExecuteWait

```
void ACS_ExecuteWait (int script, int unused, int arg1, int arg2, int arg3);
```

Usage

Using ACS_ExecuteWait is exactly equivalent to the following two commands:

```
ACS_Execute (script, 0, arg1, arg2, arg3);
```

```
ScriptWait (script);
```

Note that where you would specify a map number with ACS_Execute, you must specify 0 here because you can only wait on scripts in the current map.

Parameters

script: The script number to execute.

unused: Not currently used. Should always be set to 0.

arg1: First argument passed to the script.

arg2: Second argument passed to the script.

arg3: Third argument passed to the script.

Examples

This function is rather uncommon, but does have its uses. A common use would be to wait on a scripted event. For example, if there were some sort of event or challenge that used a script at various points in the level, and at one point the player had to complete the event as part of another script, this command can be used rather than copying the challenge script twice. To elaborate, here is a script.

```
script 15 (int armytid, int sector)
{
    Print (s:"Defeat the monsters for a prize!");

    ACS_ExecuteWait (5, 0, armytid, 0, 0);

    Ceiling_RaiseByValue (sector, 10, 64);
}
```

This script tells the player to defeat the monsters for a prize. Then it executes script 5 which could be script 5 as used here. After the success of script 5, a ceiling is raised, supposedly to reveal a prize. As this script can be used elsewhere, script 15 is like a wrapper for script 5 which offers a prize.

ACS_NamedExecuteWait

void ACS_NamedExecuteWait (string script, int unused, int arg1, int arg2, int arg3)

Usage

Using ACS_NamedExecuteWait is exactly equivalent to the following two commands:

```
ACS_NamedExecute (script, 0, arg1, arg2, arg3);
```

```
NamedScriptWait (script);
```

Note that where you would specify a map number with ACS_NamedExecute, you must specify 0 here because you can only wait on scripts in the current map.

Parameters

script: The script name to execute.

unused: Not currently used. Should always be set to 0.

arg1: First argument passed to the script.

arg2: Second argument passed to the script.

arg3: Third argument passed to the script.

Examples

Like ACS_ExecuteWait there are often other solutions to achieve the desired effect, however, this method could save unnecessary code repetition.

```
script "WaitOnMonsters" (int tid)
{
    while (ThingCount(T_NONE, tid))
        delay(1);
}
```

```
script "MonsterChallengeA" (int tid, int tag, int speed)
{
    print(s:"Kill all the monsters to open the door.");
    ACS_NamedExecuteWait("WaitOnMonsters", 0, tid);
    Door_Open(tag, speed, TRUE);
}
```

```
script "MonsterChallengeB" (int tid, int tag, int speed)
{
    Print(s:"Kill all the monsters to lower the floor.");
    ACS_NamedExecuteWait("WaitOnMonsters", 0, tid);
    Floor_LowerToLowest(tag, speed);
}
```

Delay

`void Delay (int tics);`

Usage

Delays the script for the specified amount of time.

Parameters

tics: The amount of time to wait in tics.

Examples

Delay is a very common command. A useful application of it is to prevent infinite loops and the consequential "Runaway script x terminated" error. For example:

```
script 1 ENTER
{
    int health;
    while (TRUE)
    {
        health = GetActorProperty (0, APROP_HEALTH);
        Print (s:"You have ", d:health, s:" health!");
    }
}
```

This script will cause a runaway error because it will try to tell the player their health endlessly in a single tic without stopping. Adding a slight delay will result in the desired effect, a pointless health update that lasts forever, telling the player their own health.

```
script 1 ENTER
{
    int health;
    while (TRUE)
    {
        health = GetActorProperty (0, APROP_HEALTH);
        Print (s:"You have ", d:health, s:" health!");
        Delay (1); // Wait for next frame
    }
}
```

The other obvious use is to delay events in a script. For example, the behavior of the door that is used in the trap for the first key in E1M6 of Doom can be simulated with a script like this:

```
script 12 (int sector, int speed, int seconds)
{
    Door_Close (sector, speed);
    Delay (35*seconds);
    Door_Open (sector, speed);
}
```

Although this behavior can be achieved using `Door_CloseWaitOpen` anyway.

NamedScriptWait

[Jump to navigation](#)[Jump to search](#)

void NamedScriptWait (string script)

Usage

Delays the script it is contained within until the named script specified by script has completed execution. If the specified script is not running, this command will wait until it has run. For numbered scripts, use ScriptWait.

Parameters

script: The script name to wait for.

PolyWait

```
void PolyWait (int polyid);
```

Usage

PolyWait delays the script it is in until the polyobject with an id number specified (polyid) has finished its movement. For instance, if a polyobject door is specified, then the script will delay until it has been reset to its original location. This function is useful if you want something to happen exactly after a polyobject stops moving (or similar).

Parameters

polyid: The polyobject id to wait for.

Examples

The most straightforward use of this command would be after Polyobj_Move.

```
script 1 (void)
```

```
{  
    Polyobj_Move (0, 10, 128, 64);  
    PolyWait (0);  
    Print (s:"Entryway is open.");  
}
```

This script can be used in an adaptation of the example wad (729 bytes) at the polyobject page. Instead of using a line special on the doors, this script could be accessed via a switch or a thing special. It opens the door and reports to the player that this has happened.

ScriptWait

```
void ScriptWait (int script);
```

Usage

Delays the script it is contained within until the script specified by script has completed execution. If the specified script is not running, this command will wait until it has run. For named scripts, use NamedScriptWait.

Parameters

script: The script number to wait for.

Examples

The advantage of ScriptWait is that it can hold a once-only script. Say there was a script which is to be run only once to open a door (for example it activates when the player destroys a control panel which can only be done once), but the map requires the door to be unlocked previously to this. In the event that the door is still locked, ScriptWait can be used to hold the once-only script until the script that unlocks the door has started and finished.

An example implementation of this code follows. It is rather lengthy, but fairly straightforward.

```
bool locked = TRUE;
script 1 (int sector)
{
    if (locked)
    {
        Print (s:"Security access required!");
        ScriptWait (2);
    }

    Door_Open (sector, 20);
}

script 2 (int count)
{
    while (count > 0)
    {
        HudMessage (i:count--; HUDMSG_PLAIN, 1,
                     CR_RED, 0.05, 0.95, 1.0);
        Delay (1);
    }

    HudMessage (s:"Verified!"; HUDMSG_PLAIN, 1,
                CR_GOLD, 0.05, 0.95, 1.0);

    locked = FALSE;
}
```

The first script is the once-only script. If the door is locked, it tells the user and waits for the unlock script to run and finish. After that, or if the door was already unlocked, the door opens.

The second script takes a parameter, which is the amount of frames to count before unlocking. Note that count is displayed as count--, where the two minus signs are the decrement operator. After the count is up, the door is unlocked.

TagWait

Jump to navigationJump to search

```
void TagWait (int tag);
```

Usage

Delays the script TagWait is called from until the sector with the tag specified by tag has stopped moving (make sure you do not do this with a wagglng sector or one that is in perpetual motion, because the script will delay forever).

TagWait will always wait 1 tic even if the sector is not moving.

Parameters

tag: The sector tag to wait for.

Examples

This script causes a tagged door to open, and prints a message to all players when this is complete.

```
script 1 (int sector)
```

```
{
    PrintBold (s:"Opening the hangar doors...");
    Door_Open (sector, 5, 0);
    TagWait (sector);
    PrintBold (s:"Hangar doors now open!");
}
```

The first two lines start the door opening and tell the player(s) this is happening. Note that the door is opening at a speed of 5, which is quite slow (what you might expect for a large hangar door). The script then uses TagWait to wait an unspecified amount of time before the door really opens. Once it does, this is reported.

The advantage of TagWait here is that you can use the same script on many hangar doors despite a different in height and therefore opening time. Or, you can modify your own hangar door's size or the script's speed and the "opened" message will always arrive at the exact time.

Note that when using this with a lift, the script will wait until the lift has completely finished its moving sequence, that is, until the lift has successfully returned to its starting position.

ACS_Execute

80:ACS_Execute (script, map, s_arg1, s_arg2, s_arg3)

script: Script number to execute

map: Map which contains the script

s_arg1: First argument passed to the script

s_arg2: Second argument passed to the script

s_arg3: Third argument passed to the script

Usage

Executes the specified script. A map value of zero indicates that the script is on the current map. If the script is on a different map, then the execution of the script will be delayed until the player enters the map that contains it. Only one copy of a script can be running at a time when started with this special. Use ACS_ExecuteAlways to run multiple copies of the same script at the same time (note that those cannot be suspended or terminated, however).

If the specified script was previously executed but then suspended, then execution will begin at the point immediately after where it was suspended instead of starting over again at the beginning.

Examples

This special is one of the most commonly used when editing for ZDoom. Although it is almost always used as special on a line or thing, it does have some uses when called from another script.

This script is for a first level (say, MAP01), and it opens an ammo cache on the following level (MAP02) depending on the skill level and the player's health.

script 10 (void)

```
{
    int health = GetActorProperty(0, APROP_HEALTH);

    if (GameSkill() < SKILL_HARD && health <= 100)
    {
        Print(s:"Look for an ammo cache\n
              in the next area!");

        ACS_Execute(5, 2, 0, 0, 0);
    }
    else
        Print(s:"This switch appears to\n
              have no function...");
}
```

The first line stores the player's health in a variable called health. Note that a TID of 0 usually refers to the activator of the script, which more often than not is the player. The if statement checks that the player is playing on normal skill or easier, and has 100 or less health. These are the properties for the cache opening. If they fit the requirements, it tells them and sets the script to execute on the following map. If they do not, it leaves them with a mysterious and annoying message.

The following script opens the cache on the next level:

script 5 (void)

```
{
    Ceiling_RaiseByValue(17, 20, 96);
}
```

This script simply raises the ceiling of sector tagged 17 by 96 units at speed 20. This is a very specific script but as the previous script is also specific it does not matter.

The two maps in question need to be part of the same cluster for this to work. All Chex, Doom and Heretic episodes have their own clusters, and the Doom 2 and Final Doom maps are divided in clusters based on the intermission texts: MAP01–MAP06, MAP07–MAP11, MAP12–MAP20, MAP21–MAP30, MAP31, MAP32. Clusters can be defined with MAPINFO lumps.

ACS_ExecuteAlways

226:ACS_ExecuteAlways (script, map, s_arg1, s_arg2, s_arg3)

script: Script to execute

map: Map which contains the script

s_arg1: First argument passed to the script

s_arg2: Second argument passed to the script

s_arg3: Third argument passed to the script

Usage

Like ACS_Execute, this special starts a script. However, it will allow multiple instances of a script to run simultaneously. The downside is that any scripts started with this special cannot be suspended or terminated with ACS_Suspend or ACS_Terminate, respectively.

A common use for this special is in multiplayer games, when more than one player may need to run a script at the same time. A script executed with ACS_Execute by one player would not be triggered by another if they attempt to trigger it while it is still running. In fact, the script does not run at all. ACS_ExecuteAlways can be used to prevent this problem by being able to run multiple instances at once, one for each player.

Examples

This example shows how ACS_ExecuteAlways can be an advantage over ACS_Execute. The script regenerates health for a player while they remain within the sector it is activated from.

```
int InSector[8];
```

```
script 10 (void)
```

```
{
    InSector[PlayerNumber()] = TRUE;

    while (InSector[PlayerNumber()]) {
        GiveInventory("HealthBonus", 1);
        ThingSound(0, "special/regen", 127);
        delay(15);
    }
}
```

```
script 11 (void)
```

```
{
    InSector[PlayerNumber()] = FALSE;
}
```

Script 10 is called by an "Actor Enters Sector" thing using ACS_ExecuteAlways. It sets a flag variable to true and loops until the flag is false. Script 11 is called by an "Actor Leaves Sector" thing, and unsets the flag variable. The flag system is necessary because the script cannot be ended simply by using ACS_Terminate on it.

Because this script is using ACS_ExecuteAlways instead of ACS_Execute, it is possible for eight (or more) copies of the script to be active at once – one for each player in the game.

ACS_ExecuteWithResult

Jump to navigationJump to search

84:ACS_ExecuteWithResult (script, s_arg1, s_arg2, s_arg3, s_arg4)

Usage

This is like ACS_ExecuteAlways, except the script is always run on the current map, and the return value is whatever the script sets with SetResultValue.

Parameters

script: Script to execute

s_arg1: First argument passed to the script

s_arg2: Second argument passed to the script

s_arg3: Third argument passed to the script

s_arg4: Fourth argument passed to the script

Examples

This special has a slightly unusual behaviour as most scripts are not designed to functionally return a value. It is possible, however. The syntax fits together like this:

script 1 (void)

```
{
    Print(d:ACS_ExecuteWithResult(2, 0, 0, 0)); //prints 667
}
```

script 2 (void)

```
{
    SetResultValue(667);
}
```

The point of creating a script like this is slightly overlooked by the use of functions. Functions cannot use Delay or other waiting commands. ACS_ExecuteWithResult can use them, but the result has to be decided before they are used. This means that ACS_ExecuteWithResult is sort of like a hybrid between a normal script and a function.

The purpose of this special would be to have other time-dependent events happen after the result is determined.

Another notable use of ACS_ExecuteWithResult is with switch animations. If the result value of the executed script is FALSE, the switch animation and sound will not play when the switch is hit. If the result is TRUE, the switch will animate. One example might be a switch which only responds to a specific weapon. Normally, any line marked as projectile activated will play a switch animation when hit by any weapon. Setting up a script like the following allows for more pleasing behavior. The switch only animates when it is hit with the pistol:

script 1 (void)

```
{
    if( CheckWeapon("Pistol") ) {
        Print(s:"You shot me with the pistol.");
        SetResultValue( TRUE ); // Cause the switch to animate
    } else {
        SetResultValue( FALSE ); // The switch will not animate if you use another weapon
    }
}
```

ACS_ExecuteWithResult can also be used in DECORATE expressions:

actor ReloadingPistol : Weapon

```
{
```



```

states
{
Ready:
    PISG A 0 A_JumpIfInventory ("PistolClip", 0, 2)
    PISG A 0 A_JumpIf(1 == (ACS_ExecuteWithResult(999,0,0,0)), "Reload")
    PISG A 1 A_WeaponReady
    loop
}
}

```

Return values in ZScript

Similar to DECORATE, this function can also be used in ZScript and returns an integer. There are a few ways to convert this integer to different types making it less limiting than it may appear.

Booleans

Booleans are, internally, just an integer that can be a value of 0 or 1. In this case, setting the result value to TRUE or FALSE will handle it.

```

// In ACS
SetResultValue(TRUE);
// In ZScript
bool result = ACS_ExecuteWithResult(/*..*/);

```

Floating points

Floating point values in ACS are stored as 16-bit fixed point integers. What this means is that the actual number is multiplied by 216, reserving 16 bits for the whole portion and 16 bits for the fractional portion. Returning a floating point value from ACS will return it as a 16-bit fixed point integer, so to convert it, all that needs to be done is dividing it by 216.

```

// In ACS
SetResultValue(5.2);
// In ZScript
// Make sure the number you're dividing by is a floating point number to prevent
// integer division which will remove the fractional portion
double result = ACS_ExecuteWithResult(/*..*/) / 65536.0;

```

Strings/Sounds/Names

Strings are returned as their index in the global ACS string table. Using LookupString() they can be converted back to the string and then implicitly converted to Names and Sounds as well.

```

// In ACS
SetResultValue("MyResult");
// In ZScript
string result = level.LookupString(ACS_ExecuteWithResult(/*..*/));

```

Implicit conversions

```

Name result = level.LookupString(ACS_ExecuteWithResult(/*..*/));
Sound result = level.LookupString(ACS_ExecuteWithResult(/*..*/));
TextureIDs

```

While texture ids cannot be returned directly, a string holding the texture name can, which can then be converted.

```

// In ACS
SetResultValue("TEXNAME");
// In ZScript
TextureID result = TexMan.CheckForTexture(level.LookupString(ACS_ExecuteWithResult(/*..*/)));

```

Colors

In ZScript, a color is treated very similarly to an integer. Hex values in particular are useful for setting colors since each

ARGB channel refers to 8 bits in the integer:

```
0x 00 00 00 00
```

```
  alpha red green blue
```

```
// In ACS
```

```
SetResultValue(0xFFFF0000); // Returns fully opaque red
```

```
// In ZScript
```

```
Color result = ACS_ExecuteWithResult(*..*/);
```

Also, unlike the other ACS specials, ACS_ExecuteWithResult can pass a fourth script parameter, making it useful as a line special.

ACS_LockedExecute

83:ACS_LockedExecute (script, map, s_arg1, s_arg2, lock)

85:ACS_LockedExecuteDoor (script, map, s_arg1, s_arg2, lock)

script: Script to execute

map: Map which contains the script

s_arg1: First argument passed to the script

s_arg2: Second argument passed to the script

lock: Required key, if any (see key types)

Usage

Executes the specified script if the player has the right key. A map value of zero indicates that the script is on the current map. If the script is on a different map, then the execution of the script will be delayed until the player enters the map that contains it. Only one copy of a script can be running at a time when started with this special.

If the specified script was previously executed but then suspended, then execution will begin at the point immediately after where it was suspended instead of starting over again at the beginning.

The difference between these two specials is the message they print when the player does not have the necessary key(s). ACS_LockedExecute prints the "remote" message ("You need ... to activate this object" in Doom) and ACS_LockedExecuteDoor prints the "door" message ("You need ... to open this door" in Doom).

Examples

As the only thing which can be locked normally is a door, using Door_LockedRaise, this script type allows any object or event to be locked with a key. Using the script on its own is no different from checking the player's inventory in a way such as this:

script 1 (void)

```
{
    if (CheckInventory("RedCard") || CheckInventory("RedSkull"))
    {
        Print(s:"You use the red key.");
        // More code here
    }
    else
        Print(s:"You need the red key.");
}
```

The advantage to this command is that it can check by key types rather than by key names. For example, a script which activates an elevator could be called with ACS_LockedExecute to lock it in a simple and elegant way.

ACS_Suspend

81:ACS_Suspend (script, map)

script: Script to suspend

map: Map which contains the script

Usage

Suspends execution of a script until an ACS_Execute or ACS_LockedExecute special starts it. If the specified script is not currently running, then it will be immediately suspended the next time it is run.

Examples

The following rather long example displays the use of ACS_Suspend to hold a script for a short time. The first piece of code tells the player a bomb has been activated and displays that it will detonate after a set amount of time.

```
script 1 ENTER
```

```
{
    Thing_ChangeTID(0, 999);
}
```

```
script 17 (int time)
```

```
{
    Print(s:"A bomb has been triggered!");
    SetFont("BIGFONT");

    while (time > 0)
    {
        HudMessageBold(i:time--, s:" seconds remain!";
            HUDMSG_PLAIN, 1, CR_RED, 0.5, 0.95, 1.0);

        Delay(35);
    }

    Thing_Damage(999, 300, 0);
}
```

The first script sets every player's TID to 999. For the second block, the first two lines introduce the scenario for the player. The while loop runs the countdown. Finally, after the countdown, a Thing_Damage is executed which damages every player 300 points and thus hopefully killing them, to represent the bomb going off.

The following script uses ACS_Suspend:

```
script 50 (int hold)
```

```
{
    ACS_Suspend(17, 0);
    Print(s:"You bought yourself ", i:hold,
        s:" seconds!");
    Delay(35*hold);
    ACS_Execute(17, 0, 0, 0, 0);
}
```

This holds the countdown script for the specified amount of time. It uses ACS_Suspend to stop it, waits for the amount of time, and then uses ACS_Execute to resume the script. The reason this works is because ACS_Suspend and ACS_Execute always remember where in the script it has been suspended.

If you wanted to stop the script completely, you would use ACS_Terminate.

ACS_Terminate

82:ACS_Terminate (script, map)

script: Script to terminate

map: Map which contains the script

Usage

Terminates execution of the specified script. You may not terminate scripts that were executed using the ACS_ExecuteAlways special or ENTER scripts.

Examples

The following example builds on the bomb counter example at ACS_Suspend. It terminates the countdown and tells the player the bomb has been defused.

script 51 (void)

```
{  
    Print(s:"The bomb has been defused!");  
    ACS_Terminate(17, 0);  
}
```

This is the logical command to use, as it permanently prevents the bomb from going off and thus saves the player.

Autosave

15:Autosave (No parameters required)

Usage

Automatically saves the game to an autosave slot. A common usage is to split levels into pieces, so that if the player dies, he or she does not have to start from the very beginning.

Note: using this will notify the player, so that using it right before an event will let the player know something may happen.

Examples

Usually this is set on a line making sure not to set it to be repeatable. If you were to be generous, you could set up a safety script like this:

```
script 10 (void)
{
    if (GetActorProperty(0, APROP_HEALTH) > 10)
        Autosave();
}
```

This sort of script will prevent the game from saving when the player's health is too low, as it can be considered annoying to overwrite a previous save when the player is in a dire situation.

External links

Old reference for Autosave

Conversions from linedef types

The following Doom map format types can be converted as Autosave:

Type	Conversion	Trigger
ZDoom 348:W1 Autosave	Autosave()	Player Cross
ZDoom 349:S1 Autosave	Autosave()	Player Use

Ceiling_CrushAndRaise

42:Ceiling_CrushAndRaise (tag, speed, crush [, crushmode])

tag: Tag of affected sector

speed: How quickly the ceiling moves

crush: Amount of damage to apply

crushmode: Sets the crushing mode

Usage

Lowers and raises the ceiling of the affected sectors continually, applying crushing damage to anything underneath it. If tag is 0, then the sector on the line's back side is used. The ceiling will rise at half the speed at which it lowers. If you want more control over how the ceiling raises, use Ceiling_CrushAndRaiseA.

Ceiling_CrushAndRaiseA

196:Ceiling_CrushAndRaiseA (tag, dspeed, uspeed, crush [,crushmode])

tag: Tag of affected sector

dspeed: How quickly the ceiling moves down

uspeed: How quickly the ceiling moves back up

crush: Amount of damage to apply

crushmode: Sets the crushing mode

Usage

Lowers and raises the ceiling of the affected sectors continually, applying crushing damage to anything underneath it. If tag is 0, then the sector on the line's back side is used.

Examples

script 42 (void)

```
{
    //Crusher that quickly falls, instantly kills any
    //player not using invincibility and then slowly rises back up
    Ceiling_CrushAndRaiseA(29, 48, 2, 300);
}
```

Ceiling_CrushAndRaiseDist

168:Ceiling_CrushAndRaiseDist (tag, dist, speed, damage [, crushmode])

tag: Tag of affected sector

dist: Lowest height above the floor during movement

speed: How quickly the ceiling moves

crush: Amount of damage to apply

crushmode: Sets the crushing mode

Usage

Lowers and raises the ceiling of the affected sectors continually, applying crushing damage to anything underneath it. If tag is 0, then the sector on the line's back side is used. The ceiling rises and lowers at the same speed, a behavior different from Ceiling_CrushAndRaise (which is why a dist parameter couldn't simply be added to that one).

Contrarily to other crushing specials, this crusher will stop before reaching the floor, making it possible to have crushers that affect tall actors but not shorter ones.

Examples

This replicates accurately Doom's linetype 49:

Ceiling_CrushAndRaiseDist (tag, 8, 8, 10)

This replicates accurately Strife's linetype 49:

Ceiling_CrushAndRaiseDist (tag, 8, 8, 0, 2)

Conversions from linedef types

The following Doom map format types can be converted as Ceiling_CrushAndRaiseDist:

Type	Conversion	Trigger
Doom 6:W1 Start Crusher, Fast Damage	Ceiling_CrushAndRaiseDist (tag, 8, 16, 10)	Player Cross
Strife 6:W1 Start Crusher, Very Fast Damage	Ceiling_CrushAndRaiseDist (tag, 8, 32, 10)	Player Cross
Doom 25:W1 Start Crusher, Slow Damage	Ceiling_CrushAndRaiseDist (tag, 8, 8, 10)	Player Cross
Strife 25:W1 Ceiling To 8 Above Floor Continually	Ceiling_CrushAndRaiseDist (tag, 8, 8, 0)	Player Cross
Doom 49:S1 Start Crusher, Slow Damage	Ceiling_CrushAndRaiseDist (tag, 8, 8, 10)	Player Use
Strife 49:S1 Ceiling To 8 Above Floor Continually	Ceiling_CrushAndRaiseDist (tag, 8, 8, 0, 2)	Player Use
Doom 73:WR Start Crusher, Slow Damage	Ceiling_CrushAndRaiseDist (tag, 8, 8, 10)	Player Cross, Repeatable
Strife 73:WR Start Crusher, Slow Damage	Ceiling_CrushAndRaiseDist (tag, 8, 8, 0)	Player Cross, Repeatable
Doom 77:WR Start Crusher, Fast Damage	Ceiling_CrushAndRaiseDist (tag, 8, 16, 10)	Player Cross, Repeatable
Strife 77:WR Start Crusher, Very Fast Damage	Ceiling_CrushAndRaiseDist (tag, 8, 32, 10)	Player Cross, Repeatable
Boom 164:S1 Start Crusher, Fast Damage	Ceiling_CrushAndRaiseDist (tag, 8, 16, 10)	Player Use
Boom 183:SR Start Crusher, Fast Damage	Ceiling_CrushAndRaiseDist (tag, 8, 16, 10)	Player Use, Repeatable
Boom 184:SR Start Crusher	Ceiling_CrushAndRaiseDist (tag, 8, 8, 10)	Player Use, Repeatable

Ceiling_CrushAndRaiseSilentA

197:Ceiling_CrushAndRaiseSilentA (tag, dspeed, uspeed, crush [,crushmode])

tag: Tag of affected sector

dspeed: How quickly the ceiling moves down

uspeed: How quickly the ceiling moves back up

crush: Amount of damage to apply

crushmode: Sets the crushing mode

Usage

Lowers and raises the ceiling of the affected sectors continually, applying crushing damage to anything underneath it. If tag is 0, then the sector on the line's back side is used. Crushers started with this special will only make noise when they reach the top or bottom of their strokes.

Ceiling_CrushAndRaiseSilentDist

104:Ceiling_CrushAndRaiseSilentDist (tag, dist, speed, damage [, crushmode])

- tag: Tag of affected sector
- dist: Lowest height above the floor during movement
- speed: How quickly the ceiling moves
- crush: Amount of damage to apply
- crushmode: Sets the crushing mode

Usage
Lowers and raises the ceiling of the affected sectors continually, applying crushing damage to anything underneath it. If tag is 0, then the sector on the line's back side is used. The ceiling rises and lowers at the same speed, a behavior different from Ceiling_CrushAndRaise (which is why a dist parameter couldn't simply be added to that one).

Contrarily to other crushing specials, this crusher will stop before reaching the floor, making it possible to have crushers that affect tall actors but not shorter ones.

Conversions from linedef types
The following Doom map format types can be converted as Ceiling_CrushAndRaiseSilentDist:

Type	Conversion	Trigger
Doom 141:W1 Start Crusher, Silent	Ceiling_CrushAndRaiseSilentDist (tag, 8, 8, 10)	Player Cross
Strife 141:W1 Start Crusher, Silent	Ceiling_CrushAndRaiseSilentDist (tag, 8, 8, 10)	Player Cross
Boom 150:WR Start Crusher Silent	Ceiling_CrushAndRaiseSilentDist (tag, 8, 8, 10)	Player Cross, Repeatable
Boom 165:S1 Start Crusher Silent	Ceiling_CrushAndRaiseSilentDist (tag, 8, 8, 10)	Player Use
Boom 185:SR Start Crusher Silent	Ceiling_CrushAndRaiseSilentDist (tag, 8, 8, 10)	Player Use, Repeatable

Ceiling_CrushRaiseAndStay

45:Ceiling_CrushRaiseAndStay (tag, speed, crush [, crushmode])

tag: Tag of affected sector

speed: How quickly the ceiling moves

crush: Amount of damage to apply

crushmode: Sets the crushing mode

Usage

Lowers the ceiling of the affected and applies crushing damage to anything underneath it, then raises the ceiling back up to its original height. If tag is 0, then the sector on the line's back side is used. The up speed for this special is always half the down speed. If you need more control over the way the crusher moves, you have to use

Ceiling_CrushRaiseAndStayA.

Ceiling_CrushRaiseAndStayA

195:Ceiling_CrushRaiseAndStayA (tag, dspeed, uspeed, crush [,crushmode])

tag: Tag of affected sector

dspeed: How quickly the ceiling moves down

uspeed: How quickly the ceiling moves back up

crush: Amount of damage to apply

crushmode: Sets the crushing mode

Usage

Lowest the ceiling of the affected and applies crushing damage to anything underneath it, then raises the ceiling back up to its original height. If tag is 0, then the sector on the line's back side is used.

Ceiling_CrushRaiseAndStaySilA

255:Ceiling_CrushRaiseAndStaySilA (tag, dspeed, uspeed, crush [, crushmode])

tag: Tag of affected sector

dspeed: How quickly the ceiling moves down

uspeed: How quickly the ceiling moves back up

crush: Amount of damage to apply

crushmode: Sets the crushing mode

Usage

Lowest the ceiling of the tagged sector(s) and applies crushing damage to anything underneath it, then raises the ceiling back up to its original height. If tag is 0, then the sector on the line's back side is used. Crushers started with this special will only make noise at the top and bottom of their strokes.

Ceiling_CrushStop

44:Ceiling_CrushStop (tag)

tag: Tag of affected sector

Usage

Stops a crushing ceiling.

This special works on Ceiling_CrushAndRaise, Ceiling_CrushAndRaiseA and Ceiling_CrushAndRaiseSilentA. However, once the ceiling is stopped after one of these specials, using a special to make it crush again will resume the type of crushing it was doing previously.

Note that ZDoom stores the ceiling's normal height and its current crushing height seperately, so stopping a crusher then attempting to use another special to lower or raise the ceiling will not work.

Conversions from linedef types

The following Doom map format types can be converted as Ceiling_CrushStop:

Type	Conversion	Trigger
Doom 57:W1 Stop Crusher	Ceiling_CrushStop(tag)	Player Cross
Strife 57:W1 Stop Crusher	Ceiling_CrushStop(tag)	Player Cross
Doom 74:WR Stop Crusher	Ceiling_CrushStop(tag)	Player Cross, Repeatable
Strife 74:WR Stop Crusher	Ceiling_CrushStop(tag)	Player Cross, Repeatable
Boom 168:S1 Stop Crusher	Ceiling_CrushStop(tag)	Player Use
Boom 188:SR Stop Crusher	Ceiling_CrushStop(tag)	Player Use, Repeatable

Ceiling_LowerAndCrush

43:Ceiling_LowerAndCrush (tag, speed, crush [, crushmode])

- tag: Tag of affected sector
- speed: How quickly the ceiling moves
- crush: Amount of damage to apply
- crushmode: Sets the crushing mode

Usage
Lowers the ceiling of affected sectors and applies crushing damage to anything under it. If tag is 0, then the sector on the line's back side is used.

This does not lower like other crushing ceilings. The actual height of the ceiling is changing. It will lower until it is 8 units off the ground.

Conversions from linedef types
The following Doom map format types can be converted as Ceiling_LowerAndCrush:

Type	Conversion	Trigger
Doom 44:W1	Ceil To 8 Above Floor	Ceiling_LowerAndCrush(tag, 8, 0, 2) Player Cross
Heretic 49:S1	Ceil To 8 Above Floor	Ceiling_LowerAndCrush(tag, 8, 0, 2) Player Use
Doom 72:WR	Ceil To 8 Above Floor	Ceiling_LowerAndCrush(tag, 8, 0, 2) Player Cross, Repeatable
Boom 167:S1	Ceil Down To 8 Above Floor	Ceiling_LowerAndCrush(tag, 8, 0, 2) Player Use
Boom 187:SR	Ceil Down To 8 Above Floor	Ceiling_LowerAndCrush(tag, 8, 0, 2) Player Use, Repeatable

Ceiling_LowerAndCrushDist

97:Ceiling_LowerAndCrushDist (tag, speed, crush[, dist [, crushmode]])

- tag: Tag of affected sector
- speed: How quickly the ceiling moves
- crush: Amount of damage to apply
- dist: Distance above the floor at which the movement will stop
- crushmode: Sets the crushing mode

Usage
Lowers the ceiling of affected sectors and applies crushing damage to anything under it. If tag is 0, then the sector on the line's back side is used.

This does not lower like other crushing ceilings. The actual height of the ceiling is changing. It will lower until it is dist units off the ground.

Conversions from linedef types
The following Doom map format types can be converted as Ceiling_LowerAndCrushDist:

Type	Conversion	Trigger
Strife 44:W1 Crusher To Floor And Stops	Ceiling_LowerAndCrushDist(tag, 8, 10)	Player Cross
Strife 72:WR Crusher To Floor And Stops	Ceiling_LowerAndCrushDist(tag, 8, 10)	Player Cross, Repeatable

Ceiling_LowerByTexture

269:Ceiling_LowerByTexture (tag, speed, change, crush)

tag:
speed:
change:
crush:

Ceiling_LowerByValue

40:Ceiling_LowerByValue (tag, speed, height)

tag: Tag of affected sector

speed: How quickly the ceiling moves

height: Amount to lower ceiling by

Usage

Lowers a tagged sector's ceiling by height units. If tag is 0, then the sector on the line's back side is used.

Examples

```
ceiling_lowerbyvalue(30, 8, 96);
```

```
//30 is the tag of the sector whose ceiling will be lowered
```

Ceiling_LowerByValueTimes8

199:Ceiling_LowerByValueTimes8 (tag, speed, height)

tag: Tag of affected sector

speed: How quickly the ceiling moves

height: Amount to lower ceiling by

Usage

Lowers a tagged sector's ceiling by (height * 8) units. If tag is 0, then the sector on the line's back side is used.

Ceiling_LowerInstant

193:Ceiling_LowerInstant (tag, arg1, height)

tag: Tag of affected sector

arg1: Unused

height: Amount to lower ceiling by

Usage

Instantly lowers a sector's ceiling by (height * 8) units. If tag is 0, then the sector on the line's back side is used.

If the player gets stuck underneath an instantly lowering ceiling, they will not be crushed. The ceiling will make a continual activation noise until the player moves out the way such that the ceiling can set its position correctly.

Ceiling_LowerToFloor

254:Ceiling_LowerToFloor (tag, speed)

tag: Tag of affected sector

speed: How quickly the ceiling moves

Usage

Lowers a ceiling to the height of the floor underneath it. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types

The following Doom map format types can be converted as Ceiling_LowerToFloor:

Type	Conversion	Trigger
Doom 41:S1 Ceiling To Floor	Ceiling_LowerToFloor (tag, 8)	Player Use
Doom 43:SR Ceiling To Floor	Ceiling_LowerToFloor (tag, 8)	Player Use, Repeatable
Boom 145:W1 Ceil Down To Floor Fast	Ceiling_LowerToFloor (tag, 8)	Player Cross
Boom 152:WR Ceil Down To Floor Fast	Ceiling_LowerToFloor (tag, 8)	Player Cross, Repeatable
Strife 179:W1 Ceiling To Floor	Ceiling_LowerToFloor (tag, 8)	Player Cross

Ceiling_LowerToHighestFloor

192:Ceiling_LowerToHighestFloor (tag, speed)

tag: Tag of affected sector
speed: How quickly the ceiling moves

Usage
Lowers a ceiling to the height of the highest floor surrounding it. If the ceiling is at a position lower than the highest floor, it will instantly rise to the level of the highest floor. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types
The following Doom map format types can be converted as Ceiling_LowerToHighestFloor:

Type	Conversion	Trigger
Boom 200:W1 Ceil Down To Highest Floor	Ceiling_LowerToHighestFloor (tag, 8)	Player Cross
Boom 202:WR Ceil Down To Highest Floor	Ceiling_LowerToHighestFloor (tag, 8)	Player Cross, Repeatable
Boom 204:S1 Ceil Down To Highest Floor	Ceiling_LowerToHighestFloor (tag, 8)	Player Use
Boom 206:SR Ceil Down To Highest Floor	Ceiling_LowerToHighestFloor (tag, 8)	Player Use, Repeatable

Ceiling_LowerToLowest
Jump to navigationJump to search
253:Ceiling_LowerToLowest (tag, speed)

tag: Tag of affected sector
speed: How quickly the ceiling moves

Usage
Lowers a ceiling to the height of the lowest surrounding ceiling. If the ceiling is at a position lower than the lowest surrounding ceiling, it will instantly rise to the level of the that ceiling. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types
The following Doom map format types can be converted as Ceiling_LowerToLowest:

Type	Conversion	Trigger
Boom 199:W1 Ceil Down To Lowest Ceil	Ceiling_LowerToLowest (tag, 8)	Player Cross
Boom 201:WR Ceil Down To Lowest Ceil	Ceiling_LowerToLowest (tag, 8)	Player Cross, Repeatable
Boom 203:S1 Ceil Down To Lowest Ceil	Ceiling_LowerToLowest (tag, 8)	Player Use
Boom 205:SR Ceil Down To Lowest Ceil	Ceiling_LowerToLowest (tag, 8)	Player Use, Repeatable

Ceiling_LowerToNearest

264:Ceiling_LowerToNearest (tag, speed, change, crush)

tag:
speed:
change:
crush:

Ceiling_MoveToValue

47:Ceiling_MoveToValue (tag, speed, height, neg)

tag: Tag of affected sector

speed: How quickly the ceiling moves

height: Absolute height of the move

neg: Whether or not the destination height is negative

Moves the ceiling of affected sectors to an absolute height. If the destination height is negative, then neg should be 1, otherwise it should be 0 for a positive height. If tag is 0, then the sector on the line's back side is used.

Ceiling_MoveToValueAndCrush

280:Ceiling_MoveToValueAndCrush (tag, speed, height, crush, crushmode)

tag:
speed:
height:
crush:
crushmode:

Ceiling_MoveToValueTimes8

69:Ceiling_MoveToValueTimes8 (tag, speed, height, neg)

tag: Tag of affected sector

speed: How quickly the ceiling moves

height: Absolute height of the move

neg: Whether or not the destination height is negative

Usage

Moves the ceiling of affected sectors to an absolute height of (height * 8) units. If the destination height is negative, then neg should be 1, otherwise it should be 0 for a positive height. If tag is 0, then the sector on the line's back side is used.

Ceiling_RaiseByTexture

268:Ceiling_RaiseByTexture (tag, speed, change)

tag:
speed:
change:

Ceiling_RaiseByValue

41:Ceiling_RaiseByValue (tag, speed, height)

tag: Tag of affected sector

speed: How quickly the ceiling moves

height: Amount to raise ceiling by

Usage

Raises a tagged sector's ceiling by height units. If tag is 0, then the sector on the line's back side is used.

Ceiling_RaiseByValueTimes8

198:Ceiling_RaiseByValueTimes8 (tag, speed, height)

tag: Tag of affected sector

speed: How quickly the ceiling moves

height: Amount to raise ceiling by

Usage

Raises a tagged sector's ceiling by (height * 8) units. If tag is 0, then the sector on the line's back side is used.

Ceiling_RaiseInstant

194:Ceiling_RaiseInstant (tag, arg1, height)

tag: Tag of affected sector

arg1: Unused

height: Amount to raise ceiling by

Usage

Instantly raises a sector's ceiling by (height * 8) units. If tag is 0, then the sector on the line's back side is used.

Ceiling_RaiseToHighest

262:Ceiling_RaiseToHighest (tag, speed, change)

tag:
speed:
change:

Ceiling_RaiseToHighestFloor

266:Ceiling_RaiseToHighestFloor (tag, speed, change)

tag:
speed:
change:

Ceiling_RaiseToLowest

265:Ceiling_RaiseToLowest (tag, speed, change)

tag:
speed:
change:

Ceiling_RaiseToNearest

252:Ceiling_RaiseToNearest (tag, speed)

tag: Tag of affected sector

speed: How quickly the ceiling moves

Usage

Raises a ceiling to the height of the nearest surrounding ceiling which is higher than its current position. If there is no higher ceiling, nothing happens. If tag is 0, then the sector on the line's back side is used.

Ceiling_Stop

276:Ceiling_Stop (tag)

tag:

Ceiling_ToFloorInstant

267:Ceiling_ToFloorInstant (tag, change, crush, gap)

tag:
change:
crush:
gap:

Ceiling_ToHighestInstant

263:Ceiling_ToHighestInstant (tag, change, crush)

tag:
change:
crush:

Ceiling_Waggle

38:Ceiling_Waggle (tag, amp, freq, offset, time)

tag: Tag of affected sector.

amp: Amplitude of the waggle (in 1/8 of a unit).

freq: Speed of the waggle.

offset: Phase offset of the waggle (0 through 63).

time: How many seconds the waggle lasts (0 means it will waggle forever).

Usage

“Waggles” the ceiling of the affected sectors in a sine wave. The ceiling starts moving upwards and downwards smoothly until it is in phase with the specified parameters. After the time the “waggle” dies down smoothly and the ceiling returns to its original height.

Effective when used as part of a group, each with slightly different offsets. It can be used in conjunction with Floor_Waggle.

Examples

script 1 OPEN

{ // at start...

// make ceiling of sector(s) tagged 333 slowly and slightly wagging

Ceiling_Waggle (333, 8, 16, 0, 0);

// make ceiling of sector(s) tagged 334 wagging vigorously

Ceiling_Waggle (334, 32, 256, 0, 0);

}

ChangeCamera

237:ChangeCamera (tid, who, revert)

tid: Thing ID of the camera to use (0 for normal player's view)

who: Set to 1 if the view change should affect all players

revert: Set to 1 if movement should cancel the special

Usage

Causes a player's view to move to a camera. If tid is 0, then the player's view is restored to a spot inside his head. If who is 0, only the player who activated the special has his view changed, otherwise, everyone's view changes.

If revert is 1, then if a player moves, his view will be returned to his body automatically. This behavior makes this special more suitable for such things as security cameras.

What can be viewed

Aiming camera

Moving camera

Security camera

Monsters (be sure to use the CameraHeight property.

Or just about any other object in the game. The results will vary, though.

Examples

There are different uses for cameras. Other than security cameras, they can be used very effectively for short cutscenes in games. See the second example of Thing_Hate.

To make a set of security cameras, a script like this can be used:

```
int cam = 0;
script 5 (int min, int max)
{
    if (cam < min || cam >= max)
        cam = min;
    else
        cam++;

    ChangeCamera(cam, FALSE, TRUE);
}
```

This will implement a set of different views which can be switched through. They will exit once the player moves. The script has two parameters, the minimum TID and the maximum TID of the sequence of cameras. It scrolls through all the TID's in between. This script can be used on many places in a map, with many different sequences of cameras. It can only be used for one player maps. An updated script which can store screen numbers for multiplayer would look like this:

```
int cam[8]; // Maximum number of players
script 5 (int min, int max)
{
    if (cam[PlayerNumber()] < min || cam[PlayerNumber()] >= max)
        cam[PlayerNumber()] = min;
    else
        cam[PlayerNumber()]++;
}
```

```
    ChangeCamera(cam[PlayerNumber()], FALSE, TRUE);  
}
```

This assumes a maximum number of 8 players, but is trivial to extend.

ChangeSkill

179:ChangeSkill (skill)

Skill: The skill that the game will be changed to. The default skill levels are:

- 0 — Very Easy
- 1 — Easy
- 2 — Normal
- 3 — Hard
- 4 — Nightmare!

Usage

Changes the current skill of the game. The skill change will take affect at the next map change. Instead of numbers (0—4), you can also use the following (defined in zdefs.acs):

```
SKILL_VERY_EASY
SKILL_EASY
SKILL_NORMAL
SKILL_HARD
SKILL_VERY_HARD
```

Using these will greatly increase code readability. Note, however, that ZDoom can support up to 16 skill levels, though only UDMF maps can provide skill filters for skills above 4.

Examples

This is a simple example script which ends the level, adjusting the skill according to a very primitive inspection of the player's final state. The script works just like an Exit_Normal special.

```
script 100 (int pos)
{
    int health = GetActorProperty(0, APROP_HEALTH);

    if (health < 25 && GameSkill() > SKILL_VERY_EASY)
        ChangeSkill(GameSkill() - 1);

    if (health > 100 && GameSkill() < SKILL_HARD)
        ChangeSkill(GameSkill() + 1);

    Exit_Normal(pos);
}
```

First the health of the player is found. Then, if they are particularly low on health, and it is possible to reduce the skill, the skill is dropped a level. On the other hand, if they are doing very well, the skill is increased. Finally the level is exited as normal.

ClearForceField

34:ClearForceField (tag)

tag: sector to clear forcefields from.

Usage

Clears all force fields from lines connected to the tagged sector. It will also clear the force field's middle texture.

Examples

If forcefields are set up in a way such that the player cannot pass (this can be done with two in a row at angles), then you can create an effective power off sequence with a script like this:

```
script 1 (int sector)
{
    Print(s:"Power off...");
    ClearForceField(sector);
    Light_Fade(sector, 64, 70);
}
Just tie it to a power switch and it will deactivate lights and forcefields.
```

Conversions from linedef types

The following Doom map format types can be converted as ClearForceField:

Type	Conversion	Trigger
Strife 147:S1 Clear Forcefield	ClearForceField(tag)	Player Use

DamageThing

73:DamageThing (amount, mod)

amount: The amount of damage to deal. If negative, it will heal.

mod: The obituary message that will appear if a player is killed. Uses same values as Thing_Damage.

Damage types are found on the Damage_types page.

Usage

Hurts the thing that activated the special. If amount is 0, then the thing is guaranteed to be killed regardless of any invulnerability (including God and Buddha). If a player is killed with this special, the game reports "Player died." as the obituary.

Examples

This script simulates the player being set on fire. It takes a parameter, which is the strength of the fire, from 1 to 20. Anything above 10 is extremely deadly.

```
script 165 (int power)
```

```
{
    if (power > 20)
        power = 20;

    while (power > 0)
    {
        FadeTo(255, 240, 0, 0.05 * power, 1.0);
        DamageThing(power);
        AmbientSound("vile/firecrkl", 5 * power--);
        Delay(35);
    }
}
```

The script first checks that the power variable is not set too high. Then, it loops using a "while" loop, checking to see if the fire still has power left each time. Inside the loop, the first command gives the screen a yellow glow depending on the power of the fire. The second command damages the player based on the power. The third command plays the Arch Vile's fire crackle effect and also reduces power by one (see the double minus). The last line delays the next strike.

There is an example of this command in the ActivatorTID article for a script that can be placed on a line so that the line kills any monster that crosses it, but leaves players be.

Door_Animated

14:Door_Animated (tag, speed, delay, lock)

tag: Tag of affected sector

speed: Duration of each frame in the door animation, in tics

delay: Tics until door closes

lock: Required key (See Key types)

Opens the affected sector like a door, but instead of raising the ceiling visibly, it instantly raises the ceiling and then plays a texture animation on the front sides of the door sector. This is used by ZDoom to manage Strife's animated doors.

There are a few limitations to observe:

The sector must be set up like a normal door, that is, its ceiling height should be equal to its floor height.

The texture to be used must be defined in the ANIMDEFS lump.

The door sector may only have 2 sides which face another sector.

Those 2 sides must have the same texture.

Examples

Here is an example of ANIMDEFS definition for a Strife door:

```
animateddoor SIGLDR01
  opensound DoorOpenLargeMetal
  closesound DoorCloseLargeMetal
  pic SIGLDR01
  pic SIGLDR02
  pic SIGLDR03
  pic SIGLDR04
  pic SIGLDR05
  pic SIGLDR06
  pic SIGLDR07
  pic SIGLDR08
```

Conversions from linedef types

The following Doom map format types can be converted as Door_Animated:

Type	Conversion	Trigger
Strife 144:DR Door Animated	Door_Animated (0, 4, 105)	Player Use, Monsters Activate, Repeatable
Strife 207:SR Door Animated	Door_Animated (tag, 4, 105)	Player Use, Repeatable

Door_AnimatedClose

274:Door_AnimatedClose (tag, speed)

tag:

speed:

Door_Close
Jump to navigationJump to search
10:Door_Close (tag, speed, lighttag)

tag: Tag of affected sector
speed: How quickly the door closes
lighttag: Tag of sector to perform a gradual lighting effect in
Lowers the ceiling of all affected sectors to the floor. If tag is 0, then the sector on the line's back side is used.

If lighttag is non-zero a gradual lighting effect is done in the tagged sectors. The light is gradually changed between the darkest neighboring sector when the door is fully closed and the brightest neighboring sector when the door is fully open.

Examples
This script activates a trap of closing a door with tag 12, darkens sector 13 to its original value, colorises sectors 13 and 14 (the room) to dark red and plays an ambient sound.

```
script 1 (void)
{
    Door_Close(12, 32, 13);
    Sector_SetColor(13, 200, 50, 50, 0);
    Sector_SetColor(14, 200, 50, 50, 0);
    AmbientSound("EvilLaugh", 127);
}
```

Conversions from linedef types
The following Doom map format types can be converted as Door_Raise:

Type	Conversion	Trigger
Doom 3: W1 Door Close	Door_Close (tag, 16)	Player Cross
Doom 42: SR Door Close	Door_Close (tag, 16)	Player Use, Repeatable
Doom 50: S1 Door Close	Door_Close (tag, 16)	Player Use
Doom 75: WR Door Close	Door_Close (tag, 16)	Player Cross, Repeatable
Doom 107: WR Door Close	Door_Close (tag, 64)	Player Cross, Repeatable
Doom 110: W1 Door Close	Door_Close (tag, 64)	Player Cross
Doom 113: S1 Door Close	Door_Close (tag, 64)	Player Use
Doom 116: SR Door Close	Door_Close (tag, 64)	Player Use, Repeatable

Door_CloseWaitOpen

249:Door_CloseWaitOpen (tag, speed, delay, lighttag)

tag: Tag of affected sector

speed: How quickly the door moves

delay: Octics until door opens

lighttag: Tag of sector to perform a gradual lighting effect in

Closes the specified door, and opens it again after delay octics have passed. If tag is 0, then the sector on the line's back side is used.

If lighttag is non-zero, a gradual lighting effect is done in the tagged sectors. The light is gradually changed between the darkest neighboring sector when the door is fully closed and the brightest neighboring sector when the door is fully open.

Conversions from linedef types

The following Doom map format types can be converted as Door_CloseWaitOpen:

Type	Conversion	Trigger
Doom 16:W1 Door Close + Open	Door_CloseWaitOpen (tag, 16, 240)	Player Cross
Doom 76:WR Door Close + Open	Door_CloseWaitOpen (tag, 16, 240)	Player Cross, Repeatable
Boom 175:S1 Door Close + Open	Door_CloseWaitOpen (tag, 16, 240)	Player Use
Boom 196:SR Door Close + Open	Door_CloseWaitOpen (tag, 16, 240)	Player Use, Repeatable

Door_LockedRaise

13:Door_LockedRaise (tag, speed, delay, lock, lighttag)

tag: Tag of affected sector

speed: How quickly the door moves

delay: Tics until door closes (0 if never)

lock: Required key (See Key types)

lighttag: Tag of sector to perform a gradual lighting effect in

Raises the ceiling of all affected sectors to four units below the lowest surrounding ceiling if the player has the proper key. After the door is opened, it will be closed again after delay tics. If tag is 0, then the sector on the line's back side is used.

If lighttag is non-zero a gradual lighting effect is done in the tagged sectors. The light is gradually changed between the darkest neighboring sector when the door is fully closed and the brightest neighboring sector when the door is fully open.

Conversions from linedef types

The following Doom map format types can be converted as Door_Raise:

Type	Conversion	Trigger
Doom 26:DR Door Blue Key	Door_LockedRaise (0, 16, 150, 130, tag)	Player Use, Repeatable
Heretic 26:DR Door Blue Key	Door_LockedRaise (0, 16, 150, 130, tag)	Player Use, Repeatable
Strife 26:DR Door ID Badge	Door_LockedRaise (0, 16, 150, 4, tag)	Player Use, Repeatable
Doom 27:DR Door Yellow Key	Door_LockedRaise (0, 16, 150, 131, tag)	Player Use, Repeatable
Heretic 27:DR Door Yellow Key	Door_LockedRaise (0, 16, 150, 131, tag)	Player Use, Repeatable
Strife 27:DR Door Passcard	Door_LockedRaise (0, 16, 150, 3, tag)	Player Use, Repeatable
Doom 28:DR Door Red Key	Door_LockedRaise (0, 16, 150, 129, tag)	Player Use, Repeatable
Heretic 28:DR Door Green Key	Door_LockedRaise (0, 16, 150, 129, tag)	Player Use, Repeatable
Doom 28:DR Door ID Card	Door_LockedRaise (0, 16, 150, 10, tag)	Player Use, Repeatable
Doom 32:D1 Door Stay Open Blue Key	Door_LockedRaise (0, 16, 0, 130, tag)	Player Use, Monsters Activate
Heretic 32:D1 Door Stay Open Blue Key	Door_LockedRaise (0, 16, 0, 130, tag)	Player Use, Monsters Activate
Strife 32:D1 Door Stay Open ID Badge	Door_LockedRaise (0, 16, 0, 4, tag)	Player Use
Doom 33:D1 Door Stay Open Red Key	Door_LockedRaise (0, 16, 0, 129, tag)	Player Use, Monsters Activate
Heretic 33:D1 Door Stay Open Green Key	Door_LockedRaise (0, 16, 0, 129, tag)	Player Use, Monsters Activate
Strife 33:D1 Door Stay Open ID Card	Door_LockedRaise (0, 16, 0, 10, tag)	Player Use
Doom 34:D1 Door Stay Open Yellow Key	Door_LockedRaise (0, 16, 0, 131, tag)	Player Use, Monsters Activate
Heretic 34:D1 Door Stay Open Yellow Key	Door_LockedRaise (0, 16, 0, 131, tag)	Player Use, Monsters Activate
Strife 34:D1 Door Stay Open Passcard	Door_LockedRaise (0, 16, 0, 3, tag)	Player Use
Doom 99:SR Door Stay Open Blue Key Fast	Door_LockedRaise (tag, 64, 0, 130)	Player Use, Repeatable
Strife 99:SR Door ID Badge Fast	Door_LockedRaise (tag, 64, 150, 4)	Player Use, Repeatable
Doom 133:S1 Door Stay Open Blue Key Fast	Door_LockedRaise (tag, 64, 0, 130)	Player Use
Strife 133:S1 Door Stay Open ID Badge Fast	Door_LockedRaise (tag, 64, 0, 4)	Player Use
Doom 134:SR Door Stay Open Red Key Fast	Door_LockedRaise (tag, 64, 0, 129)	Player Use, Repeatable
Strife 134:SR Door ID Card Fast	Door_LockedRaise (tag, 64, 150, 11)	Player Use, Repeatable
Doom 135:S1 Door Stay Open Red Key Fast	Door_LockedRaise (tag, 64, 0, 129)	Player Use
Strife 135:S1 Door Stay Open ID Card Fast	Door_LockedRaise (tag, 64, 0, 11)	Player Use
Doom 136:SR Door Stay Open Yellow Key Fast	Door_LockedRaise (tag, 64, 0, 131)	Player Use, Repeatable
Strife 136:SR Door Passcard Fast	Door_LockedRaise (tag, 64, 150, 3)	Player Use, Repeatable
Doom 137:S1 Door Stay Open Yellow Key Fast	Door_LockedRaise (tag, 64, 0, 131)	Player Use
Strife 137:S1 Door Stay Open Passcard Fast	Door_LockedRaise (tag, 64, 0, 3)	Player Use
Strife 151:SR Door Gold Key Fast	Door_LockedRaise (tag, 64, 150, 10)	Player Use, Repeatable

Strife 152:SR Door Brass Key Fast Door_LockedRaise (tag, 64, 150, 17) Player Use, Repeatable
Strife 153:SR Door Silver Key Fast Door_LockedRaise (tag, 64, 150, 12) Player Use, Repeatable
Strife 156:D1 Door Stay Open Brass Key Door_LockedRaise (0, 16, 0, 17, tag) Player Use
Strife 157:D1 Door Stay Open Silver Key Door_LockedRaise (0, 16, 0, 12, tag) Player Use
Strife 158:D1 Door Stay Open Gold Key Door_LockedRaise (0, 16, 0, 10, tag) Player Use
Strife 159:DR Door Gold Key Door_LockedRaise (0, 16, 150, 10, tag)Player Use, Repeatable
Strife 160:DR Door Silver Key Door_LockedRaise (0, 16, 150, 12, tag)Player Use, Repeatable
Strife 161:DR Door Brass Key Door_LockedRaise (0, 16, 150, 17, tag)Player Use, Repeatable
Strife 162:S1 Door Brass Key FastDoor_LockedRaise (tag, 64, 0, 17)Player Use
Strife 163:S1 Door Silver Key Fast Door_LockedRaise (tag, 64, 0, 12)Player Use
Strife 164:S1 Door Gold Key Fast Door_LockedRaise (tag, 64, 0, 10)Player Use
Strife 165:SR Fake Door 'Doesn't Seem To Work'Door_LockedRaise (0, 0, 0, 102) Player Use
Strife 166:DR Door Severed Hand Door_LockedRaise (0, 16, 150, 6, tag) Player Use, Repeatable
Strife 167:S1 Door Severed Hand Fast Door_LockedRaise (tag, 64, 0, 6) Player Use
Strife 168:SR Door Severed Hand Fast Door_LockedRaise (tag, 64, 150, 6) Player Use, Repeatable
Strife 169:DR Door Base Key Door_LockedRaise (0, 16, 150, 1, tag) Player Use, Repeatable
Strife 170:DR Door Govs Key Door_LockedRaise (0, 16, 150, 2, tag) Player Use, Repeatable
Strife 171:S1 Door Prison Key Fast Door_LockedRaise (tag, 16, 0, 5) Player Use
Strife 172:SR Door Power 1 Key Door_LockedRaise (tag, 16, 150, 7) Player Use, Repeatable
Strife 173:SR Door Power 2 Key Door_LockedRaise (tag, 16, 150, 8) Player Use, Repeatable
Strife 176:SR Door Power 3 Key Door_LockedRaise (tag, 16, 150, 9) Player Use, Repeatable
Strife 190:DR Door Order Key Door_LockedRaise (0, 16, 150, 15, tag)Player Use, Repeatable
Strife 191:SR Door Military ID Door_LockedRaise (tag, 16, 150, 14) Player Use, Repeatable
Strife 192:SR Door Warehouse Key Door_LockedRaise (tag, 16, 150, 16) Player Use, Repeatable
Strife 205:SR Fake Door 'Only Available In Retail' Door_LockedRaise (0, 16, 0, 103) Player Use, Repeatable
Strife 217:DR Door Core Key Door_LockedRaise (0, 16, 0, 23, tag) Player Use
Strife 221:DR Door Mauler Key Door_LockedRaise (0, 16, 0, 24, tag) Player Use
Strife 223:SR Door Mine Key Door_LockedRaise (tag, 16, 150, 26) Player Use, Repeatable
Strife 224:DR Door Chapel Key Door_LockedRaise (0, 16, 0, 20, tag) Player Use
Strife 225:DR Door Catacomb Key Door_LockedRaise (0, 16, 0, 21, tag) Player Use

Door_Open

11:Door_Open (tag, speed, lighttag)

tag: Tag of affected sector
speed: How quickly the door opens
lighttag: Tag of sector to perform a gradual lighting effect in
Raises the ceiling of all affected sectors to four units below the lowest surrounding ceiling. If tag is 0, then the sector on the line's back side is used.

If lighttag is non-zero a gradual lighting effect is done in the tagged sectors. The light is gradually changed between the darkest neighboring sector when the door is fully closed and the brightest neighboring sector when the door is fully open.

Conversions from linedef types
The following Doom map format types can be converted as Door_Open:

Type	Conversion	Trigger
Doom 2: W1 Door Stay Open	Door_Open (tag, 16)	Player Cross
Doom 31: D1 Door Stay Open	Door_Open (0, 16, tag)	Player Use
Strife 31: D1 Door Stay Open	Door_Open (0, 16, tag)	Player Use, Monsters Activate
Doom 46: GR Door Stay Open	Door_Open (tag, 16)	Attack Hit, Missile Cross, Monsters Activate, Repeatable
Doom 61: SR Door Stay Open	Door_Open (tag, 16)	Player Use, Repeatable
Doom 86: WR Door Stay Open	Door_Open (tag, 16)	Player Cross, Repeatable
Doom 103: S1 Door Stay Open	Door_Open (tag, 16)	Player Use
Doom 106: WR Door Stay Open Fast	Door_Open (tag, 64)	Player Cross, Repeatable
Doom 109: W1 Door Stay Open Fast	Door_Open (tag, 64)	Player Cross
Doom 112: S1 Door Stay Open Fast	Door_Open (tag, 64)	Player Use
Doom 115: SR Door Stay Open Fast	Door_Open (tag, 64)	Player Use, Repeatable
Doom 118: D1 Door Stay Open Fast	Door_Open (0, 64, tag)	Player Use

Door_Raise

12:Door_Raise (tag, speed, delay, lighttag)

tag: Tag of affected sector.
speed: Door's movement speed. Doom's standard doors move at 16, and blazing doors move at 64.
delay: Tics until door closes. Doom waits 150 tics.
lighttag: Tag of sector to perform a gradual lighting effect in
Raises the ceiling of all affected sectors to four units below the lowest surrounding ceiling. After the door is opened, it will be closed again after delay tics. If tag is 0, then the sector on the line's back side is used.

If lighttag is non-zero a gradual lighting effect is done in the tagged sectors. The light is gradually changed between the darkest neighboring sector when the door is fully closed and the brightest neighboring sector when the door is fully open.

Conversions from linedef types
The following Doom map format types can be converted as Door_Raise:

Type	Conversion	Trigger
1: DR Door	Door_Raise (0, 16, 150, tag)	Player Use, Monsters Activate, Repeatable
Doom 4: W1 Door	Door_Raise (tag, 16, 150)	Player Cross, Monsters Activate
Doom 29: S1 Door	Door_Raise (tag, 16, 150)	Player Use
Doom 63: SR Door	Door_Raise (tag, 16, 150)	Player Use, Repeatable
Doom 90: WR Door	Door_Raise (tag, 16, 150)	Player Cross, Repeatable
Heretic 100: WR Door Triple Speed	Door_Raise (tag, 48, 150)	Player Cross, Repeatable
Doom 105: WR Door Fast	Door_Raise (tag, 64, 150)	Player Cross, Repeatable
Doom 108: W1 Door Fast	Door_Raise (tag, 64, 150)	Player Cross
Doom 111: S1 Door Fast	Door_Raise (tag, 64, 150)	Player Use
Doom 114: SR Door Fast	Door_Raise (tag, 64, 150)	Player Use, Repeatable
Doom 117: DR Door Fast	Door_Raise (0, 64, 150, tag)	Player Use, Repeatable

Door_WaitClose
Jump to navigationJump to search
106:Door_WaitClose (tag, speed, wait, lighttag)

tag:
speed:
wait:
lighttag:

Door_WaitRaise
Jump to navigationJump to search
105:Door_WaitRaise (tag, speed, delay, wait, lighttag)

tag:
speed:
delay:
wait:
lighttag:

Elevator_LowerToNearest

247:Elevator_LowerToNearest (tag, speed)

tag: Tag of affected sector

speed: How fast the elevator moves

Lowers the floor and ceiling of the tagged sector so that its floor aligns with the floor of the next lower sector while keeping the same amount of distance between the floor and ceiling.

Conversions from linedef types

The following Doom map format types can be converted as Elevator_LowerToNearest:

Type	Conversion	Trigger
Boom 231:W1 Elevator To Lower Floor	Elevator_LowerToNearest (tag, 32)	Player Cross
Boom 232:WR Elevator To Lower Floor	Elevator_LowerToNearest (tag, 32)	Player Cross, Repeatable
Boom 233:S1 Elevator To Lower Floor	Elevator_LowerToNearest (tag, 32)	Player Use
Boom 234:SR Elevator To Lower Floor	Elevator_LowerToNearest (tag, 32)	Player Use, Repeatable

Elevator_MoveToFloor

246:Elevator_MoveToFloor (tag, speed)

tag: Tag of affected sector

speed: How fast the elevator moves

Moves the floor and ceiling of the tagged sector so that its floor aligns with the floor of the sector the activating linedef is facing, while keeping the same amount of distance between the floor and ceiling.

Conversions from linedef types

The following Doom map format types can be converted as Elevator_MoveToFloor:

Type	Conversion	Trigger
Boom 235:W1 Elevator To Current Floor	Elevator_MoveToFloor (tag, 32)	Player Cross
Boom 236:WR Elevator To Current Floor	Elevator_MoveToFloor (tag, 32)	Player Cross, Repeatable
Boom 237:S1 Elevator To Current Floor	Elevator_MoveToFloor (tag, 32)	Player Use
Boom 238:SR Elevator To Current Floor	Elevator_MoveToFloor (tag, 32)	Player Use, Repeatable

Elevator_RaiseToNearest

245:Elevator_RaiseToNearest (tag, speed)

tag: Tag of affected sector

speed: How fast the elevator moves

Raises the floor and ceiling of the tagged sector so that its floor aligns with the floor of the next higher sector while keeping the same amount of distance between the floor and ceiling.

Conversions from linedef types

The following Doom map format types can be converted as Elevator_RaiseToNearest:

Type	Conversion	Trigger
Boom 227:W1 Elevator To Higher Floor	Elevator_RaiseToNearest (tag, 32)	Player Cross
Boom 228:WR Elevator To Higher Floor	Elevator_RaiseToNearest (tag, 32)	Player Cross, Repeatable
Boom 229:S1 Elevator To Higher Floor	Elevator_RaiseToNearest (tag, 32)	Player Use
Boom 230:SR Elevator To Higher Floor	Elevator_RaiseToNearest (tag, 32)	Player Use, Repeatable

Exit_Normal

243:Exit_Normal (pos)

pos: Corresponds to destination player start spot arg0
Usage
Teleports the player to the next map defined for this map in MAPINFO and to the player start spot whose arg0 matches pos. If you are making a vanilla-style doom map, just use 0.

Note that the map will not immediately exit as soon as this command is encountered; instead, a flag will be set that causes the map to exit during processing of the next tic. Therefore, if there are any further commands in the ACS script it's present in, they will be executed up until the first delay. To ensure that the script halts immediately at this command, make it the final command within its script or put a terminate/delay(1) command immediately after it.

Examples
This command is generally used on lines or sectors, and less often in scripts. However, some unusual situations can be created using scripting. A locked exit can be made by using a script and accessing it with a ACS_LockedExecute special.

Another example is an exit which can be activated, for example by turning on a power generator.

```
bool exit = FALSE;

script 1 (void)
{
    if (exit)
        Exit_Normal(0);
    else
        Print(s:"You need to activate the power first.");
}
```

```
script 10 (int sector)
{
    exit = TRUE;
    Light_RaiseByValue(sector, 96);
    Print(s:"Power restored.");
}
```

Script 1 controls access to the exit. It should be set to a switch and flagged as repeatable. Script 10 restores the power to the area. The sector tag is passed as a parameter, and it raises the light level there to represent the lights coming back on with the power.

Conversions from linedef types
The following Doom map format types can be converted as :

Type	Conversion	Trigger
Doom 11:S1 Exit (Normal)	Exit_Normal (0)	Player Use
Doom 52:W1 Exit (Normal)	Exit_Normal (0)	Player Cross
Boom 197:G1 Exit (Normal)	Exit_Normal (0)	Attack Hit, Missile Cross

Exit_Secret

244:Exit_Secret (pos)

pos: Corresponds to destination player start spot arg0

Usage

Teleports the player to the secret map defined for this map in MAPINFO and to the player start spot whose arg0 matches pos. Note that on standard Doom 1 maps, ZDoom will only use this special on maps E1M3, E2M5, E3M6 and E4M2. In Doom 2, only maps MAP15 and MAP31 will be affected by its use. This can be overridden by the use of MAPINFO.

Note that the map will not immediately exit as soon as this command is encountered; instead, a flag will be set that causes the map to exit during processing of the next tic. Therefore, if there are any further commands in the ACS script it is present in, they will be executed up until the first delay. To ensure that the script halts immediately at this command, make it the final command within its script or put a terminate/delay(1) command immediately after it.

Examples

It is possible to make an exit that is both normal and secret, and only goes to the secret level based on some condition, for example: How many tokens were collected – tokens being some sort of predefined thing that can be picked up, be it a DECORATE item or a standard doom item.

```
int tokens = 0;
```

```
script 1 (void)
{
    if (tokens == 5)
        Exit_Secret(0);
    else
        Exit_Normal(0);
}
```

```
script 7 (void)
{
    tokens++;
    PrintBold(d:tokens, s:"/5 collected!");
}
```

The first script exits the level. If the player has all five tokens, the next level will be the secret level. Otherwise they will have missed it and are sent to the next level.

The script numbered 7 is set activate upon each of the token's specials. That is, make their thing special to be ACS_Execute and the arguments to be (7, 0, 0, 0, 0). When a thing is picked up, its special is activated, so these five items will add to the token count and allow the secret level to be reached.

Conversions from linedef types

The following Doom map format types can be converted as Exit_Secret:

Type	Conversion	Trigger
Doom 51:S1 Exit (Secret)	Exit_Secret (0)	Player Use
Heretic 105:W1 Exit (Secret)	Exit_Secret (0)	Player Cross
Doom 124:W1 Exit (Secret)	Exit_Secret (0)	Player Cross
Boom 198:G1 Exit (Secret)	Exit_Secret (0)	Attack Hit, Missile Cross

ExtraFloor_LightOnly

50:ExtraFloor_LightOnly (tag, type)

tag: tag of sector to apply effect to

type: Type of light effect (see below)

This special, when placed on a line, will transfer the control sector's (the sector on the first side) light level to the walls of the sector referenced by tag at the heights specified from the control sector.

If you have a real sector, floor/ceiling heights 0 and 128, and have a control sector with heights 64 and 96, you would get sort of a white bar from 64 to 96 in the real sector. You can, of course, apply slopes, colored lighting and any effect to the control sector.

The second parameter is the type of light effect to do. The available values are as follows:

0 — Extra light extends from ceiling of control sector down to top of another type 0 light.

1 — Extra light extends from ceiling down to the floor of the control sector.

2 — Extra light extends from control sector's ceiling down to the top of another extra light.

Note that when using two or more extra lights on a sector, the linedef with the lower number has priority.

Example: let's say the real sector's floor/ceiling heights are 0 and 128, the first control sector's heights are 0 and 64, and the second control sector's heights are 32 and 128. If the first control sector's type 50 linedef is linedef #100, and the second's is linedef #200, that would mean heights 0–64 would be the light level of the first control sector, and 64–128 the level of the second control sector. If the linedef numbers were swapped over, then heights 0–32 would be the light level of the first control sector, and 32–128 the level of the second control sector.

Note that this is implemented as a Sector_Set3DFloor with invisible textures, so sector effects such as damaging specials are also transferred.

An example wad can be found [here](#).

Floor_CrushStop

46:Floor_CrushStop (tag)

tag: Tag of affected sector
Stops a crushing floor.

Floor_Donut

250:Floor_Donut (ptag, pspeed, sspeed)

ptag: Tag of the pillar in the center of the donut
pspeed: How quickly to lower the pillar
sspeed: How quickly to raise the surrounding sector's floor
Performs DoomWikiLogoIcon.pnga donut action on the specified sectors.

Examples
In this example, the donut's pillar has a tag of 1, the pillar lowers with a speed of 10 and the floor around the pillar raises with a speed of 15, after which its floor is changed to that of the sector around.

```
Script 1 (void)
{
  Floor_Donut(1, 10, 15);
  Print(s:"Unflooding the area, moving the column down...");
}
```

Conversions from linedef types
The following Doom map format types can be converted as :

Type	Conversion	Trigger
Doom 9:S1 Floor Donut	Floor_Donut (tag, DORATE, DORATE)	Player Use
Boom 146:W1 Floor Donut Raise	Floor_Donut (tag, DORATE, DORATE)	Player Cross
Boom 155:WR Floor Donut Raise	Floor_Donut (tag, DORATE, DORATE)	Player Cross, Repeatable
Boom 191:SR Floor Donut Raise	Floor_Donut (tag, DORATE, DORATE)	Player Use, Repeatable

Floor_LowerByTexture

261:Floor_LowerByTexture (tag, speed, change, crush)

tag:
speed:
change:
crush:

Floor_LowerByValue

20:Floor_LowerByValue (tag, speed, height)

tag: Tag of affected sectors, or 0 to use the line's back sector.

speed: Floor's movement speed. Doom's standard floors move at 8, and fast floors move at 32.

height: Distance to lower the floor.

Lowers any tagged sector's floor by height units.

Floor_LowerInstant

66:Floor_LowerInstant (tag, arg1, height)

tag: Tag of affected sector

arg1: Unused

height: Height of move

Instantly lowers the floor of the affected sectors by ($\text{height} * 8$) units. If tag is 0, then the sector on the line's back side is used.

Floor_LowerToHighest

242:Floor_LowerToHighest (tag, speed, adjust, force_adjust)

tag: Tag of affected sector

speed: How quickly the floor moves

adjust: Amount of difference from target height + 128

force_adjust: Forces to apply the adjusted height.

Lowers a tagged sector's floor to the height of the highest surrounding floor + adjust - 128. So if you want the floor to lower to the height of the highest surrounding floor, use an adjust of 128. If you want it to lower to 8 units below the other floor, use an adjust of 120. Similar for other values of adjust. If tag is 0, then the sector on the line's back side is used.

The adjustment is normally done only if the lowest found floor height is different from the tagged sector's current floor height. However, if force_adjust is set to 1, it is done nonetheless.

Conversions from linedef types

The following Doom map format types can be converted as Floor_LowerToHighest:

Type	Conversion	Trigger
Doom 19:W1 Floor To Highest Adjacent Floor	Floor_LowerToHighest (tag, 8, 128)	Player Cross
Doom 36:W1 Floor To 8 Above HAF Fast	Floor_LowerToHighest (tag, 32, 136)	Player Cross
Heretic 36:W1 Floor To 8 Above HAF Fast	Floor_LowerToHighest (tag, 32, 136, 1)	Player Cross
Strife 36:W1 Floor To Highest Adjacent Floor Fast	Floor_LowerToHighest (tag, 32, 128)	Player Cross
Doom 45:SR Floor To Highest Adjacent Floor	Floor_LowerToHighest (tag, 8, 128)	Player Use, Repeatable
Doom 70:SR Floor To 8 Above HAF Fast	Floor_LowerToHighest (tag, 32, 136)	Player Use, Repeatable
Heretic 70:SR Floor To 8 Above HAF Fast	Floor_LowerToHighest (tag, 32, 136, 1)	Player Use, Repeatable
Strife 70:SR Floor To Highest Adjacent Floor Fast	Floor_LowerToHighest (tag, 32, 128)	Player Use, Repeatable
Doom 71:S1 Floor To 8 Above HAF Fast	Floor_LowerToHighest (tag, 32, 136)	Player Use
Heretic 71:S1 Floor To 8 Above HAF Fast	Floor_LowerToHighest (tag, 32, 136, 1)	Player Use
Strife 71:S1 Floor To Highest Adjacent Floor Fast	Floor_LowerToHighest (tag, 32, 128)	Player Use
Doom 83:WR Floor To Highest Adjacent Floor	Floor_LowerToHighest (tag, 8, 128)	Player Cross, Repeatable
Doom 98:WR Floor To 8 Above HAF Fast	Floor_LowerToHighest (tag, 32, 136)	Player Cross, Repeatable
Heretic 98:WR Floor To 8 Above HAF Fast	Floor_LowerToHighest (tag, 32, 136, 1)	Player Cross, Repeatable
Strife 98:WR Floor To Highest Adjacent Floor Fast	Floor_LowerToHighest (tag, 32, 128)	Player Cross, Repeatable
Doom 102:S1 Floor To Highest Adjacent Floor	Floor_LowerToHighest (tag, 8, 128)	Player Use

Floor_LowerToHighestEE
Jump to navigationJump to search
256:Floor_LowerToHighestEE (tag, speed, change)

tag:
speed:
change:

Floor_LowerToLowest

21:Floor_LowerToLowest (tag, speed)

tag: Tag of affected sector

speed: How quickly the floor moves

Lowers a tagged sector's floor to the height of the lowest surrounding floor. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types

The following Doom map format types can be converted as Floor_LowerToLowest:

Type	Conversion	Trigger
Doom 23:S1 Floor To Lowest Adjacent Floor	Floor_LowerToLowest(tag, 8)	Player Use
Strife 23:S1 Floor To Lowest Adjacent Floor	Floor_LowerToLowest(tag, 8)	Player Use
Doom 38:W1 Floor To Lowest Adjacent Floor	Floor_LowerToLowest(tag, 8)	Player Cross
Strife 38:W1 Floor To Lowest Adjacent Floor	Floor_LowerToLowest(tag, 8)	Player Cross
Doom 60:SR Floor To Lowest Adjacent Floor	Floor_LowerToLowest(tag, 8)	Player Use, Repeatable
Strife 60:SR Floor To Lowest Adjacent Floor	Floor_LowerToLowest(tag, 8)	Player Use, Repeatable
Doom 82:WR Floor To Lowest Adjacent Floor	Floor_LowerToLowest(tag, 8)	Player Cross, Repeatable
Strife 82:WR Floor To Lowest Adjacent Floor	Floor_LowerToLowest(tag, 8)	Player Cross, Repeatable

Floor_LowerToLowestCeiling

258:Floor_LowerToLowestCeiling (tag, speed, change)

tag:
speed:
change:

Floor_LowerToLowestTxTy

241:Floor_LowerToLowestTxTy (tag, speed)

tag: Tag of affected sector

speed: How quickly the floor moves

Lowers a tagged sector's floor to the height of the lowest surrounding floor and uses the trigger change model to give it a new picture and special. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types

The following Doom map format types can be converted as Floor_LowerToLowestTxTy:

Type	Conversion	Trigger
Doom 37:W1 Floor To LAF Change Tex + Type	Floor_LowerToLowestTxTy (tag, 8)	Player Cross
Doom 84:WR Floor To LAF Change Tex + Type	Floor_LowerToLowestTxTy (tag, 8)	Player Cross, Repeatable
Boom 159:S1 Floor To Lowest Adjacent Floor	Floor_LowerToLowestTxTy (tag, 8)	Player Use
Boom 177:SR Floor To Lowest Adjacent Floor	Floor_LowerToLowestTxTy (tag, 8)	Player Use, Repeatable

Floor_LowerToNearest

22:Floor_LowerToNearest (tag, speed)

tag: Tag of affected sector
speed: How quickly the floor moves
Lowers a tagged sector's floor to the height of the highest surrounding floor that is lower than the tagged sector's current height. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types
The following Doom map format types can be converted as Floor_LowerToNearest:

Type	Conversion	Trigger
Boom 219:W1 Floor Down To Adjacent Floor	Floor_LowerToNearest (tag, 8)	Player Cross
Boom 220:WR Floor Down To Adjacent Floor	Floor_LowerToNearest (tag, 8)	Player Cross, Repeatable
Boom 221:S1 Floor Down To Adjacent Floor	Floor_LowerToNearest (tag, 8)	Player Use
Boom 222:SR Floor Down To Adjacent Floor	Floor_LowerToNearest (tag, 8)	Player Use, Repeatable

Floor_MoveToValue

37:Floor_MoveToValue (tag, speed, height, neg)

tag: Tag of affected sector

speed: How quickly the floor moves

height: Absolute height of the move

neg: Whether or not the destination height is negative

Moves the floor of affected sectors to an absolute height. If the destination height is negative, then neg should be 1, otherwise it should be 0 for a positive height. If tag is 0, then the sector on the line's back side is used.

Floor_MoveToValueAndCrush

279:Floor_MoveToValueAndCrush (tag, speed, height, crush, crushmode)

tag:
speed:
height:
crush:
crushmode:

Floor_MoveToValueTimes8

68:Floor_MoveToValueTimes8 (tag, speed, height, neg)

tag: Tag of affected sector

speed: How quickly the floor moves

height: Absolute height of the move

neg: Whether or not the destination height is negative

Moves the floor of affected sectors to an absolute height of (height * 8) units. If the destination height is negative, then neg should be 1, otherwise it should be 0 for a positive height. If tag is 0, then the sector on the line's back side is used.

Floor_RaiseAndCrush

28:Floor_RaiseAndCrush (tag, speed, crush [,crushmode])

tag: Tag of affected sector

speed: How quickly the floor moves

crush: Amount of damage to apply

crushmode: Sets the crushing mode

Raises the floor to a height of 8 units below the ceiling and applies crushing damage to anything standing on it. If tag is 0, then the sector on the line's back side is used.

Floor_RaiseAndCrushDoom

99:Floor_RaiseAndCrushDoom (tag, speed, crush [,crushmode])

tag: Tag of affected sector

speed: How quickly the floor moves

crush: Amount of damage to apply

crushmode: Sets the crushing mode

Raises the floor to a height of 8 units below the lowest surrounding ceiling and applies crushing damage to anything standing on it. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types

The following Doom map format types can be converted as Floor_RaiseAndCrushDoom:

Type	Conversion	Trigger
Doom 55:S1 Floor To 8 Below LAC + Crush	Floor_RaiseAndCrushDoom(tag, 8, 10, 2)	Player Use
Doom 56:W1 Floor To 8 Below LAC + Crush	Floor_RaiseAndCrushDoom(tag, 8, 10, 2)	Player Cross
Doom 65:SR Floor To 8 Below LAC + Crush	Floor_RaiseAndCrushDoom(tag, 8, 10, 2)	Player Use, Repeatable
Doom 94:WR Floor To 8 Below LAC + Crush	Floor_RaiseAndCrushDoom(tag, 8, 10, 2)	Player Cross, Repeatable

Floor_RaiseByTexture

240:Floor_RaiseByTexture (tag, speed)

tag: Tag of affected sectors

speed: How quickly the floor moves

Raises a tagged sector's floor by the height of the shortest lower texture on its defining linedefs. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types

The following Doom map format types can be converted as :

Type	Conversion	Trigger
Doom 30:W1 Floor Up Shortest Lo Tex	Floor_RaiseByTexture (tag, 8)	Player Cross
Doom 96:WR Floor Up Shortest Lo Tex	Floor_RaiseByTexture (tag, 8)	Player Cross, Repeatable
Boom 158:S1 Floor Up By Shortest Lo Tex	Floor_RaiseByTexture (tag, 8)	Player Use
Boom 176:SR Floor Up By Shortest Lo Tex	Floor_RaiseByTexture (tag, 8)	Player Use, Repeatable

Floor_RaiseByValue

23:Floor_RaiseByValue (tag, speed, height)

tag: Tag of affected sector
speed: How quickly the floor moves
height: Amount to raise floor by
Raises a tagged sector's floor by height units. If tag is 0, then the sector on the line's back side is used.

Examples

```
#include "zcommon.acs"
script 2 (void)
{
    Floor_RaiseByValue(69,10,80);
}
```

Conversions from linedef types
The following Doom map format types can be converted as Floor_RaiseByValue:

Type	Conversion	Trigger
Doom 58:W1 Floor Up 24	Floor_RaiseByValue(tag, 8, 24)	Player Cross
Strife 58:W1 Floor Up 64	Floor_RaiseByValue(tag, 8, 64)	Player Cross
Doom 92:WR Floor Up 24	Floor_RaiseByValue(tag, 8, 24)	Player Cross, Repeatable
Strife 92:WR Floor Up 64	Floor_RaiseByValue(tag, 8, 64)	Player Cross, Repeatable
Boom 161:S1 Floor Up 24	Floor_RaiseByValue(tag, 8, 24)	Player Use
Boom 180:SR Floor Up 24	Floor_RaiseByValue(tag, 8, 24)	Player Use, Repeatable
EDGE 434:S1 Floor Up 2	Floor_RaiseByValue(tag, 8, 2)	Player Use
EDGE 435:SR Floor Up 2	Floor_RaiseByValue(tag, 8, 2)	Player Use, Repeatable
EDGE 436:W1 Floor Up 2	Floor_RaiseByValue(tag, 8, 2)	Player Cross
EDGE 437:WR Floor Up 2	Floor_RaiseByValue(tag, 8, 2)	Player Cross, Repeatable
EDGE 438:G1 Floor Up 2	Floor_RaiseByValue(tag, 8, 2)	Attack Hit, Missile Cross
EDGE 439:GR Floor Up 2	Floor_RaiseByValue(tag, 8, 2)	Attack Hit, Missile Cross, Repeatable

Floor_RaiseByValueTimes8

35:Floor_RaiseByValueTimes8 (tag, speed, height)

tag: Tag of affected sector

speed: How quickly the floor moves

height: Amount to raise floor by

Raises a tagged sector's floor by (height * 8) units. If tag is 0, then the sector on the line's back side is used. This special is useful in the Hexen map format where argument cannot exceed 255; in UDMF large values can be used with Floor_RaiseByValue.

Conversions from linedef types

The following Doom map format types can be converted as Floor_RaiseByValueTimes8:

Type	Conversion	Trigger
Doom 140:S1 Floor Up 512	Floor_RaiseByValueTimes8(tag, 8, 64)	Player Use
Strife 140:S1 Floor Up 512	Floor_RaiseByValueTimes8(tag, 8, 64)	Player Use
Boom 142:W1 Floor Up 512	Floor_RaiseByValueTimes8(tag, 8, 64)	Player Cross
Boom 147:WR Floor Up 512	Floor_RaiseByValueTimes8(tag, 8, 64)	Player Cross, Repeatable
Boom 178:SR Floor Up 512	Floor_RaiseByValueTimes8(tag, 8, 64)	Player Use, Repeatable

Floor_RaiseByValueTxTy

239:Floor_RaiseByValueTxTy (tag, speed, height)

tag: Tag of affected sector

speed: How quickly the floor moves

height: Amount to raise floor by

Raises a tagged sector's floor by height units and uses the trigger change model to give it a new picture and special. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types

The following Doom map format types can be converted as Floor_RaiseByValueTxTy:

Type	Conversion	Trigger
Doom 59:W1 Floor Up 24 Change Tex + Type	Floor_RaiseByValueTxTy (tag, 8, 24)	Player Cross
Doom 93:WR Floor Up 24 Change Tex + Type	Floor_RaiseByValueTxTy (tag, 8, 24)	Player Cross, Repeatable
Boom 160:S1 Floor Up 24 Change Tex & Effect	Floor_RaiseByValueTxTy (tag, 8, 24)	Player Use
Boom 179:SR Floor Up 24 Change Tex & Effect	Floor_RaiseByValueTxTy (tag, 8, 24)	Player Use, Repeatable

Floor_RaiseInstant

67:Floor_RaiseInstant (tag, arg1, height)

tag: Tag of affected sector

arg1: Unused

height: Height of move

Instantly raises the floor of the affected sectors by (height * 8) units. If tag is 0, then the sector on the line's back side is used.

Floor_RaiseToCeiling

259:Floor_RaiseToCeiling (tag, speed, change, crush, gap)

tag:
speed:
change:
crush:
gap:

Floor_RaiseToHighest

24:Floor_RaiseToHighest (tag, speed)

tag: Tag of affected sector

speed: How quickly the floor moves

Raises a tagged sector's floor to the height of the highest surrounding floor. If tag is 0, then the sector on the line's back side is used.

Floor_RaiseToLowest
Jump to navigationJump to search
257:Floor_RaiseToLowest (tag, change, crush)

tag:
change:
crush:

Floor_RaiseToLowestCeiling

238:Floor_RaiseToLowestCeiling (tag, speed)

tag: Tag of affected sector

speed: How quickly the floor moves

Raises a tagged sector's floor to the height of the lowest surrounding ceiling. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types

The following Doom map format types can be converted as Floor_RaiseToLowestCeiling:

Type	Conversion	Trigger
Doom 5: W1 Floor To Lowest Adjacent Ceiling	Floor_RaiseToLowestCeiling (tag, 8)	Player Cross
Doom 24: G1 Floor To Lowest Adjacent Ceiling	Floor_RaiseToLowestCeiling (tag, 8)	Attack Hit, Missile Cross
Doom 64: SR Floor To Lowest Adjacent Ceiling	Floor_RaiseToLowestCeiling (tag, 8)	Player Use, Repeatable
Doom 91: WR Floor To Lowest Adjacent Ceiling	Floor_RaiseToLowestCeiling (tag, 8)	Player Cross, Repeatable
Doom 101: S1 Floor To Lowest Adjacent Ceiling	Floor_RaiseToLowestCeiling (tag, 8)	Player Use

Floor_RaiseToNearest

25:Floor_RaiseToNearest (tag, speed)

tag: Tag of affected sector

speed: How quickly the floor moves

Raises a tagged sector's floor to the height of the next higher surrounding floor. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types

The following Doom map format types can be converted as Floor_RaiseToNearest:

Type	Conversion	Trigger
Doom 18:S1 Floor To Higher Adjacent Floor	Floor_RaiseToNearest(tag, 8)	Player Use
Strife 18:S1 Floor To Higher Adjacent Floor	Floor_RaiseToNearest(tag, 8)	Player Use
Doom 69:SR Floor To Higher Adjacent Floor	Floor_RaiseToNearest(tag, 8)	Player Use, Repeatable
Strife 69:SR Floor To Higher Adjacent Floor	Floor_RaiseToNearest(tag, 8)	Player Use, Repeatable
Doom 119:W1 Floor To Higher Adjacent Floor	Floor_RaiseToNearest(tag, 8)	Player Cross
Strife 119:W1 Floor To Higher Adjacent Floor	Floor_RaiseToNearest(tag, 8)	Player Cross
Doom 128:WR Floor To Higher Adjacent Floor	Floor_RaiseToNearest(tag, 8)	Player Cross, Repeatable
Strife 128:WR Floor To Higher Adjacent Floor	Floor_RaiseToNearest(tag, 8)	Player Cross, Repeatable
Doom 129:WR Floor To Higher Floor Fast	Floor_RaiseToNearest(tag, 32)	Player Cross, Repeatable
Strife 129:WR Floor To Higher Floor Fast	Floor_RaiseToNearest(tag, 32)	Player Cross, Repeatable
Doom 130:W1 Floor To Higher Floor Fast	Floor_RaiseToNearest(tag, 32)	Player Cross
Strife 130:W1 Floor To Higher Floor Fast	Floor_RaiseToNearest(tag, 32)	Player Cross
Doom 131:S1 Floor To Higher Floor Fast	Floor_RaiseToNearest(tag, 32)	Player Use
Strife 131:S1 Floor To Higher Floor Fast	Floor_RaiseToNearest(tag, 32)	Player Use
Doom 132:SR Floor To Higher Floor Fast	Floor_RaiseToNearest(tag, 32)	Player Use, Repeatable
Strife 132:SR Floor To Higher Floor Fast	Floor_RaiseToNearest(tag, 32)	Player Use, Repeatable

Floor_Stop

275:Floor_Stop (tag)

tag:

Floor_ToCeilingInstant

260:Floor_ToCeilingInstant (tag, change, crush, gap)

tag:
change:
crush:
gap:

Floor_TransferNumeric

236:Floor_TransferNumeric (tag)

tag: Tag of affected sector

Transfers the floor picture and sector special from one sector to another using the numeric change model: the sector used as a model is the one with floor at destination height across the lowest-numbered two-sided line surrounding the affected sector.

Conversions from linedef types

The following Doom map format types can be converted as Floor_TransferNumeric:

Type	Conversion	Trigger
Boom 78:SR Change Floor Tex & Effect (Numeric Model)	Floor_TransferNumeric (tag)	Player Use, Repeatable
Boom 239:W1 Change Floor Tex & Effect (Numeric Model)	Floor_TransferNumeric (tag)	Player Cross
Boom 240:WR Change Floor Tex & Effect (Numeric Model)	Floor_TransferNumeric (tag)	Player Cross, Repeatable
Boom 241:S1 Change Floor Tex & Effect (Numeric Model)	Floor_TransferNumeric (tag)	Player Use

Floor_TransferTrigger

235:Floor_TransferTrigger (tag)

tag: Tag of affected sector
Transfers the floor picture and sector special from one sector to another.

The source sector is chosen by looking at the sector on Side 1 of the triggered linedef.

Conversions from linedef types

Type	Conversion	Trigger
Boom 153:W1	Change Floor Texture & Effect (Trigger Model)	Floor_TransferTrigger (tag) Player Cross
Boom 154:WR	Change Floor Texture & Effect (Trigger Model)	Floor_TransferTrigger (tag) Player Cross, Repeatable
Boom 189:S1	Change Floor Texture & Effect (Trigger Model)	Floor_TransferTrigger (tag) Player Use
Boom 190:SR	Change Floor Texture & Effect (Trigger Model)	Floor_TransferTrigger (tag) Player Use, Repeatable

Floor_Waggle

138:Floor_Waggle (tag, amp, freq, offset, time)

tag: Tag of affected sector.

amp: Amplitude of the waggle (in 1/8 of a unit).

freq: Speed of the waggle.

offset: Phase offset of the waggle (0 through 63).

time: How many seconds the waggle lasts (0 means it will waggle forever).

Usage

“Waggles” the floor of the affected sectors in a sine wave. The floor starts moving upwards and downwards smoothly until it is in phase with the specified parameters. After the time the “waggle” dies down smoothly and the floor returns to its original height.

Effective when used as part of a group, each with slightly different offsets. It can be used in conjunction with Ceiling_Waggle.

Examples

script 1 OPEN

{ // at start...

// make floor of sector(s) tagged 333 slowly and slightly wagging

Floor_Waggle (333, 8, 16, 0, 0);

// make floor of sector(s) tagged 334 wagging vigorously

Floor_Waggle (334, 32, 256, 0, 0);

}

Conversions from linedef types

The following Doom map format types can be converted as Floor_Waggle:

Type Conversion Trigger

ZDoom 338 Floor_Waggle (tag, 24, 32, 0, 0) Player Cross

ZDoom 339 Floor_Waggle (tag, 12, 32, 0, 0) Player Cross

FloorAndCeiling_LowerByValue

95:FloorAndCeiling_LowerByValue (tag, speed, height)

tag: Tag of affected sector

speed: Speed of the move

height: Amount to move

Moves both the floor and ceiling of the affected sectors down by height units.

FloorAndCeiling_LowerRaise

251:FloorAndCeiling_LowerRaise (tag, fspeed, cspeed, boomemu)

tag: Tag of affected sector

fspeed: Speed to move the floor

cspeed: Speed to move the ceiling

boomemu: Whether the function should emulate a Boom bug where only either the floor or the ceiling moved, but not both. Set it to 1998 if you want the Boom bug.

Lowers the sector's floor to the lowest surrounding floor and raises its ceiling to the highest surrounding ceiling.

Conversions from linedef types

The following Doom map format types can be converted as FloorAndCeiling_LowerRaise:

Type	Conversion	Trigger
Boom 151:WR Ceil Up To Highest Ceil	FloorAndCeiling_LowerRaise (tag, 8, 8)	Player Cross, Repeatable
Boom 166:S1 Ceil Up To Highest Ceil	FloorAndCeiling_LowerRaise (tag, 8, 8, 1998)	Player Use
Boom 186:SR Ceil Up To Highest Ceil	FloorAndCeiling_LowerRaise (tag, 8, 8, 1998)	Player Use, Repeatable

Technical details

The Boom bug is caused by the use of the boolean operator `||` instead of the bitwise operator `|` in the logic used for linedef types 166 and 186, for example:

```
// Raise ceiling, Lower floor
// 186 SR EV_DoCeiling(raiseToHighest), EV_DoFloor(lowerFloortoLowest)
if (EV_DoCeiling(line, raiseToHighest) ||
    EV_DoFloor(line, lowerFloorToLowest))
    P_ChangeSwitchTexture(line,1);
break;
```

In boolean mode, C uses a form of lazy evaluation: `(a || b)` is true if either a or b are true, so if a is true, there is no need to evaluate b as well: whether it's true or not is irrelevant as the entire expression will be true anyway. Inversely, if a is false, then and only then will b be evaluated, because then the expression is true if b is true, and false otherwise. On the other hand, the bitwise operation is created by combining the values of both a and b, which are treated as numbers rather than as boolean values, so both a and b will be evaluated all the time.

Because of the confusion between both operators, Boom will not call `EV_DoFloor()` in this function unless the call to `EV_DoCeiling` returned a failure. So in Boom, this function will only move the ceiling, or the floor if the ceiling was blocked, or neither if both were blocked; but it will never move both planes at the same time contrarily to what was originally intended.

ZDoom fixed this bug, but naturally some existing Boom maps were designed around it and the corrected behavior may expose or create problems in these maps that were never caught by playtesting on the bugged engine.

The value of 1998 (Boom's year of publication) to flag the special as demanding Boom's behavior was chosen as being unlikely to have been set by error or mistake to a normal linedef in a UDMF map and impossible on a Hexen format map, contrarily to using a boolean logic such as "any non-null value".

FloorAndCeiling_RaiseByValue

96:FloorAndCeiling_RaiseByValue (tag, speed, height)

tag: Tag of affected sector

speed: Speed of the move

height: Amount to move

Moves both the floor and ceiling of the affected sectors up by height units.

ForceField

33:ForceField (No parameters required)

Turns the line into a forcefield that will hurt (8 damage on SKILL_VERY_EASY, 16 on all other) and push the player when activating. It does not block the player, you have to set the blocking flags on the map. Note that you will have to set it to "repeatable" to make the force field persistent and you have to set the proper activation type.

This special can be cleared by the A_RemoveForceField codepointer. In Strife levels with forcefields, a ForceFieldGuard actor is always placed in a sector bordering the forcefield linedef so that the explosion of a DegrinOre will destroy it and trigger the removal of the force field effect even if the ore was placed within range but outside of the sector.

Conversions from linedef types
The following Doom map format types can be converted as ForceField:

Type	Conversion	Trigger
Strife 148:SR Forcefield	ForceField()	Player Use, Repeatable

FS_Execute

Warning: The feature described on this page is only there for compatibility purposes with Doom Legacy and will not be extended or maintained beyond this point. It is therefore recommended to look for equivalent ZDoom method to achieve the same effects instead of using this feature.

158:FS_Execute (int scriptnumber, side, keynum, message)

Executes a Fragglescript script.

scriptnumber: Number of the script to run.

side: on a two-sided linedef, determines if it can be activated from the front side.

keynum: The number of the key required for activation, as defined in LOCKDEFS.

message: The type of message given if the player lacks the needed key.

Sides:

0: Can be activated from both side

1: Can only be activated from the front side

Keynums:

See key types for the list of default key numbers, and LOCKDEFS to define your own.

Messages:

0: Displays a "you need X key to open this door" message

1: Displays a "you need X key to activate this object" message

Generic_Ceiling

201:Generic_Ceiling (tag, speed, height, target, flag)

tag: Tag of affected sector, or 0 to use the sector on the back side of the line.

speed: Ceiling's movement speed. Doom's standard ceilings move at 8, and fast crushers move at 16.

height: Distance to move (when target is 0).

target: How to determine the ceiling's destination height.

flags: Various flags; see below.

This special encapsulates BOOM's generalized ceilings.

Parameters

Target

target can be one of the following:

0: Move by height units

1: Move to highest neighboring ceiling

2: Move to lowest neighboring ceiling

3: Move to nearest neighboring ceiling

4: Move to highest neighboring floor

5: Move to the sector's own floor

6: Move by the height of the shortest upper texture on the sector

Flags

A moving ceiling may optionally copy its ceiling texture and special from a "model" sector. There are two ways to choose a model sector. The trigger model uses the sector on the front side of the line that caused the move. The numeric model looks for neighboring sectors whose ceilings are at the destination height, and uses the one with the lowest-numbered shared linedef. If no model sector can be found, nothing is copied.

flags is composed of:

0: Don't copy anything

1: Copy ceiling texture, and remove special (Tx0)

2: Copy ceiling texture only (Tx)

3: Copy both ceiling texture and special (TxTy)

+4: Use the numeric model (default is trigger model)

+8: Raise ceiling (default is to lower)

+16: Inflict crushing damage

Conversions from linedef types

The following Doom map format types can be converted as Generic_Ceiling:

Type	Conversion	Trigger
------	------------	---------

Doom 40:W1 Ceil To Highest Ceiling	Generic_Ceiling (tag, 8, 0, 1, 8)	Player Cross
------------------------------------	-----------------------------------	--------------

Generic_Crusher

205:Generic_Crusher (tag, dspeed, uspeed, silent, crush)

169:Generic_Crusher2 (tag, dspeed, uspeed, silent, crush)

tag: Tag of affected sector

dspeed: How quickly the ceiling lowers

uspeed: How quickly the ceiling raises

silent: Set this to 1 to prevent the crusher from making noise

crush: Amount of damage to apply

This special encapsulates Boom's generalized crushers. It is the same as Ceiling_CrushAndRaiseA, except that it takes an argument to control the "noisiness" of the crusher.

Unlike the other crushing specials where the crush mode is a parameter, the generic version uses a different special type because it already has all parameters in use. Generic_Crusher uses Doom mode and Generic_Crusher2 uses Hexen mode.

Generic_Door

202:Generic_Door (tag, speed, kind, delay, lock)

tag: Tag of affected sector, or 0 to use the line's back sector.
speed: Door's movement speed. Doom's standard doors move at 16, and blazing doors move at 64.
kind: Type of movement to apply to the door.
delay: Octics until door is automatically closed/opened. Doom waits about 34 octics.
lock: Required key, if any (see Key types).
This special encapsulates Boom's generalized doors.

Parameters

Tag
If tag is 0, then the sector on the line's back side is used.

If 128 is added to the kind parameter, the tag is used as a light tag, not a door tag; see below.

Kind
Kind can be one of four values:

- 0 — Raise door and close it after delay octics
- 1 — Open door and leave it open
- 2 — Close door and open it after delay octics
- 3 — Close door and leave it closed

If 64 is added to the kind parameter, the door is not retriggerable even if it is a manual door. This is consistent with Boom's behavior for generalized door.

If 128 is added to the kind parameter, the tag is used as a light tag, not a door tag. In this case, this special only opens the door on the line's back side. Instead, a gradual lighting effect is done in the tagged sectors. The light is gradually changed between the darkest neighboring sector when the door is fully closed and the brightest neighboring sector when the door is fully open.

Delay
Because the largest values used in Boom maps could not fit in tics in the Hexen map format, this generalized line uses values in octics. For reference, the following table provides equivalence for the values available in Boom. Use the values from the octics column. You can also use any arbitrary value you want.

Seconds	Tics	Octics	Notes
1	35	8	
4	150	34	Equivalent to vanilla door delay
9	300	69	
30	1050	240	

Lock
For reference, Boom allows twelve possible lock types, but it is also possible to use any defined custom lock value.

ValueKey needed to unlock

- 100 Any key
- 130 Any blue key
- 129 Any red key
- 131 Any yellow key
- 2 Blue key card
- 1 Red key card
- 3 Yellow key card
- 5 Blue skull key

- 4 Red skull key
- 6 Yellow skull key
- 229 All three colors
- 101 All six keys

Generic_Floor

200:Generic_Floor (tag, speed, height, target, flags)

tag: Tag of affected sectors, or 0 to use the line's back sector.

speed: Floor's movement speed. Doom's standard floors move at 8, and fast floors move at 32.

height: Distance to move (when target is 0).

target: How to determine the floor's destination height.

flags: Various flags; see below.

This special encapsulates BOOM's generalized floors.

Parameters

Target

target can be one of the following:

0: Move by height units

1: Move to highest neighboring floor

2: Move to lowest neighboring floor

3: Move to nearest neighboring floor

4: Move to lowest neighboring ceiling

5: Move to the sector's own ceiling

6: Move by the height of the shortest lower texture on the sector

Flags

A moving floor may optionally copy its floor texture and special from a "model" sector. There are two ways to choose a model sector. The trigger model uses the sector on the front side of the line that caused the move. The numeric model looks for neighboring sectors whose floors are at the destination height, and uses the one with the lowest-numbered shared linedef. If no model sector can be found, nothing is copied.

flags is composed of:

0: Don't copy anything

1: Copy floor texture, and remove special (Tx0)

2: Copy floor texture only (Tx)

3: Copy both floor texture and special (TxTy)

+4: Use the numeric model (default is trigger model)

+8: Raise floor (default is to lower)

+16: Inflict crushing damage

Examples

Many other floor specials can be expressed in terms of Generic_Floor:

Special Equivalent Generic_Floor

Floor_LowerByValue(tag, speed, height) Generic_Floor(tag, speed, height, 0, 0)

Floor_LowerToLowest(tag, speed)Generic_Floor(tag, speed, 0, 2, 0)

Floor_LowerToLowestTxTy(tag, speed) Generic_Floor(tag, speed, 0, 2, 3)

Floor_LowerToNearest(tag, speed) Generic_Floor(tag, speed, 0, 3, 0)

Floor_RaiseByTexture(tag, speed)Generic_Floor(tag, speed, 0, 6, 8)

Floor_RaiseByValue(tag, speed, height) Generic_Floor(tag, speed, height, 0, 8)

Floor_RaiseByValueTxTy(tag, speed, height) Generic_Floor(tag, speed, height, 0, 11)

Floor_RaiseToHighest(tag, speed)Generic_Floor(tag, speed, 0, 1, 8)

Floor_RaiseToLowestCeiling(tag, speed) Generic_Floor(tag, speed, 0, 4, 8)

Floor_RaiseToNearest(tag, speed)Generic_Floor(tag, speed, 0, 3, 8)

Floor_TransferNumeric(tag)Generic_Floor(tag, 0, 0, 0, 7)

Floor_TransferTrigger(tag)Generic_Floor(tag, 0, 0, 0, 3)

Generic_Lift

203:Generic_Lift (tag, speed, delay, type, height)

tag: Tag of affected sector, or 0 to use the line's back side.
speed: Lift's movement speed. Doom's standard lifts move at 32, and blazing lifts move at 64.
delay: Octics before reversing direction. Doom waits for 24 octics (3 seconds).
type: Specifies what type of platform this is; see below.
height: Distance to move; only used when target is 0.
This special encapsulates Boom's generalized lifts.

Parameters

Delay
Because the largest values used in Boom maps could not fit in tics in the Hexen map format, this generalized special uses values in octics. For reference, the following table provides equivalence for the values available in Boom. Use the values from the octics column. You can also use any arbitrary value you want.

Seconds	Tics	Octics	Notes
1	35	8	
3	105	24	Equivalent to vanilla lift delay
4.7	165	~38	
10	350	80	

Type

Value	Target	Default sound sequence
0	Plat_UpByValue	Platform
1	Plat_DownWaitUpStay	Platform
2	Plat_DownToNearestFloor (does not exist as an explicit action special)	Platform
3	Plat_DownToLowestCeiling (does not exist as an explicit action special)	Platform
4	Plat_PerpetualRaise	Floor

Generic_Stairs

204:Generic_Stairs (tag, speed, height, flags, reset)

tag: Tag of first sector in staircase
speed: How quickly the steps move
height: Height of each step
flags: controls direction and ignorance of floor textures
reset: Tics until the stairs return to their original heights (0 if never)
This special encapsulates BOOM's generalized stairs. The sectors of the staircase are determined as they are in BOOM. If bit 0 of flags is set (0x01), the stairs are built up, otherwise they build down. If bit 1 is set (0x02), then the determination of stair sectors ignores different floor textures. If a line activates this special, and the line is marked as repeatable, then the direction the stairs build will alternate between up and down each time the special is activated.

Conversions from linedef types
The following Doom map format types can be converted as Generic_Stairs:

Type	Conversion	Trigger
Strife 146:S1	Build Stairs 16	Down Generic_Stairs (tag, 16, 16, 0, 0) Player Use
Strife 178:W1	Build Stairs 16	Down Generic_Stairs (tag, 16, 16, 0, 0) Player Cross

GlassBreak

49:GlassBreak (dontspawnjunk, type)

The GlassBreak special is used to create glass that breaks when the line is activated. Normally you should set the line's activation type to projectile hits and the flags to impassable. When this special is activated three things happen:

The line's switch texture state is changed (from on to off or vice versa) and the switch sound is played.

The line's flags are changed to be non-blocking.

If dontspawnjunk is 0, seven actors of type are spawned at the line's center (at the side the line is facing to). type is the spawn number associated with the actor class to spawn. If it is 0, the actor class to spawn defaults to GlassJunk.

Conversions from linedef types

The following Doom map format types can be converted as GlassBreak:

Type	Conversion	Trigger
Strife 182:G1 Break Glass	GlassBreak()	Impact Hit, Missile Cross, Monsters Activate

HealThing

248:HealThing (amount, max)

Usage

Gives health to the actor that activated the special.

Parameters

amount: the amount by which to heal.

max: the maximum value the actor's health can reach when healed.

If max is 0 or the actor to heal is a non-player, GiveBody is called to handle the healing, with the specified amount passed to it while leaving GiveBody's max parameter's value as the default.

If max is greater than 0 and the actor to heal is a player, amount is added to the actor's health with no modifications to it; if max is 1, the actor is healed up to the maximum set by the soulsphere, which is 200 (this maximum value could be changed via DeHackEd). Otherwise, if it is greater than 1, the actor is healed up to that value.

Warning: do not use negative amounts if max is greater than 0. While negative amounts are acceptable for use with the call to GiveBody, if max is greater than 0, HealThing uses those negative amounts as they are, thus reducing the actor's health. HealThing does not kill the actor if it reduces its health to 0 or less, and instead, leaves it in a broken state.

Examples

This new berserk powerup uses HealThing to heal the player to full health.

```
class NewBerserk : Berserk
{
    States
    {
        Pickup:
            TNT1 A 0 A_GiveInventory("PowerStrength");
            TNT1 A 0 HealThing(100);
            TNT1 A 0 A_SelectWeapon("KoolFist");
            Stop;
    }
}
```

Light_ChangeToValue

112:Light_ChangeToValue (tag, value)

tag: Tag of affected sector
value: New light level
Sets the light level in a sector to value.

Conversions from linedef types
The following Doom map format types can be converted as Light_ChangeToValue:

Type	Conversion	Trigger
Doom 13:W1 Light To 255	Light_ChangeToValue (tag, 255)	Player Cross
Doom 35:W1 Light To 35	Light_ChangeToValue (tag, 35)	Player Cross
Doom 79:WR Light To 35	Light_ChangeToValue (tag, 35)	Player Cross, Repeatable
Doom 81:WR Light To 255	Light_ChangeToValue (tag, 255)	Player Cross, Repeatable
Doom 138:SR Light To 255	Light_ChangeToValue (tag, 255)	Player Use, Repeatable
Doom 139:SR Light To 35	Light_ChangeToValue (tag, 35)	Player Use, Repeatable
Boom 170:S1 Light To 35	Light_ChangeToValue (tag, 35)	Player Use
Boom 171:S1 Light To 255	Light_ChangeToValue (tag, 255)	Player Use

Light_Fade

113:Light_Fade (tag, value, tics)

tag: Tag of affected sector

value: New light level

tics: How long the light takes to fade to the new level

Changes the light level in a sector gradually over a period of time until it reaches value.

Examples

//add this line to your script. it will affect a sector with tag #9.

```
light_fade(9,225,35);
```

Light_Flicker

115:Light_Flicker (tag, upper, lower)

tag: Tag of affected sector

upper: Upper light level

lower: Lower light level

Switches the light level in a sector between upper and lower at random intervals between approximately 0.2 and 1.8 seconds.

Light_ForceLightning

109:Light_ForceLightning (mode)

mode: Decides which operation to perform

0: Makes lightning flash immediately and keeps the map in lightning mode as if it had been set in MAPINFO.

1: Makes lightning flash only once and terminates lightning mode right away.

2: Terminates lightning mode now. (Will not interrupt a current lightning flash, however)

Light_Glow

114:Light_Glow (tag, upper, lower, tics)

tag: Tag of affected sector

upper: Upper light level

lower: Lower light level

tics: Fading time between these two light levels

Fades the light level in a tagged sector cyclically between upper and lower with tics duration between each extreme.

Light_LowerByValue

111:Light_LowerByValue (tag, value)

tag: Tag of affected sector
value: Amount to lower the light level by
Decreases the light level in a sector by value.

Light_MaxNeighbor

234:Light_MaxNeighbor (tag)

tag: Tag of affected sector
Sets the light level of a sector to match the highest light level found in one of its neighboring sectors.

Conversions from linedef types
The following Doom map format types can be converted as Light_MaxNeighbor:

Type	Conversion	Trigger
Doom 12:W1	Light To Highest Adjacent Level	Light_MaxNeighbor (tag) Player Cross
Doom 80:WR	Light To Highest Adjacent Level	Light_MaxNeighbor (tag) Player Cross, Repeatable
Boom 169:S1	Light To Highest Adjacent Level	Light_MaxNeighbor (tag) Player Use
Boom 192:SR	Light To Highest Adjacent Level	Light_MaxNeighbor (tag) Player Use, Repeatable

Light_MinNeighbor

233:Light_MinNeighbor (tag)

tag: Tag of affected sector
Sets the light level of a sector to match the lowest light level found in one of its neighboring sectors.

Conversions from linedef types
The following Doom map format types can be converted as Light_MinNeighbor:

Type	Conversion	Trigger
Doom 104:W1	Light To Lowest Adjacent Level	Light_MinNeighbor (tag) Player Cross
Boom 157:WR	Light To Lowest Adjacent Level	Light_MinNeighbor (tag) Player Cross, Repeatable
Boom 173:S1	Light To Lowest Adjacent Level	Light_MinNeighbor (tag) Player Use
Boom 194:SR	Light To Lowest Adjacent Level	Light_MinNeighbor (tag) Player Use, Repeatable

Light_RaiseByValue

110:Light_RaiseByValue (tag, value)

tag: Tag of affected sector
value: Amount to raise the light by
Increases the light level in a sector by value.

Light_Stop

117:Light_Stop (tag)

Terminates lighting effect in tagged sector. The light level is left at whatever it was when the effect was halted

Light_Strobe

116:Light_Strobe (tag, upper, lower, u-tics, l-tics)

tag: Tag of affected sector

upper: Upper light level

lower: Lower light level

u-tics: Time to stay at upper light level

l-tics: Time to stay at lower light level

Switches the light level in a sector between upper and lower at the given intervals.

Light_StrobeDoom

232:Light_StrobeDoom (tag, u-tics, l-tics)

tag: Tag of affected sector

u-tics: Time to stay at upper light level

l-tics: Time to stay at lower light level

This is the same as Light_Strobe, except that upper is whatever the sector's light level is right now, and lower is whatever the lowest light level in the surrounding sectors is. If the surrounding sectors are at the same light level as the affected sector, then lower is 0.

Conversions from linedef types

The following Doom map format types can be converted as Light_StrobeDoom:

Type	Conversion	Trigger
Doom 17:W1 Light Blink 1.0 Sec	Light_StrobeDoom (tag, 5, 35)	Player Cross
Boom 156:WR Light Start Blinking	Light_StrobeDoom (tag, 5, 35)	Player Cross, Repeatable
Boom 172:S1 Light Start Blinking	Light_StrobeDoom (tag, 5, 35)	Player Use
Boom 193:SR Light Start Blinking	Light_StrobeDoom (tag, 5, 35)	Player Use, Repeatable

Line_AlignCeiling

183:Line_AlignCeiling (lineid, side)

lineid: Line to use for alignment

side: Align sector in front of or behind the line?

Causes the tops of the ceiling textures in the front (0) or back (1) of the specified line to align with the line. Any future scrolling or rotation of the ceiling will use this new alignment as a basis, unless changed again with one of these specials.

Note that this is not a static init linedef type, but one that has to be explicitly activated, either by a switch, a walkover trigger, a script, etc. Just putting it on the line will not do anything. You must first assign a line identification number to the linedef to which you want the flat aligned, then use the activator. If you want the flat aligned right from the start of the map, use the special in an OPEN script.

Line_AlignFloor

184:Line_AlignFloor (lineid, side)

lineid: Line to use for alignment

side: Align sector in front of or behind the line?

Causes the tops of the floor textures in the front (0) or back (1) of the specified line to align with the line. Any future scrolling or rotation of the floor will use this new alignment as a basis, unless changed again with one of these specials. The reason for this is so that once rotating sectors are a reality, you only need to align the texture once, and it will stay aligned as you rotate the sector.

Note that this is not a static init linedef type, but one that has to be explicitly activated, either by a switch, a walkover trigger, a script, etc. Just putting it on the line will not do anything. You must first assign a line identification number to the linedef to which you want the flat aligned, then use the activator. If you want the flat aligned right from the start of the map, use the special in an OPEN script

Line_Horizon

9:Line_Horizon (no parameters required)

This special is to be placed on a line in the map, which is then "extended" into infinity. It will only create an infinite view on that particular line, so you may need to place it on multiple lines to get the desired effect. Useful for skyboxes. Do not use with slopes!

Examples

Note: This article lists no examples. If you make use of this feature in your own project(s) or know of any basic examples that could be shared, please add them. This will make it easier to understand for future authors seeking assistance. Your contributions are greatly appreciated.

Conversions from linedef types

The following Doom map format types can be converted as Line Horizon:

Type	Conversion	Trigger
MiniZDoomLogoIcon.png	337:Line Horizon	Line_Horizon ()
MiniEternityLogoIcon.png	450:Line Horizon	Line_Horizon ()

Line_Mirror

182:Line_Mirror (no parameters required)

Turns the line into a mirror.

Note that for ZDoom, the line must have enough void space behind it to fit all of the reflected geometry. Otherwise, areas behind the mirror will show up through the mirror as merged with the reflection. It is best to leave the same size of the current room with the mirror + 1 unit size in width, length and height, else HOMS (Hall of Mirrors) may appear. Mirrors done for GZDoom (OpenGL or Vulkan) no longer need a large void space behind the mirror, it can be as small as a sliver of 1/4 mu.

Examples

An example of a mirror can be found at [here](#).

Conversions from linedef types

The following Doom map format types can be converted as Line_Mirror:

Type	Conversion	Trigger
ZDoom 336:	Line_Mirror	Line_Mirror ()

Line_SetAutomapFlags

281:Line_SetAutomapFlags (lineid, setflags, clearflags)

lineid: The id of the line of which to set the automap flags.

setflags: The flags to be set.

clearflags: The flags to be un-set.

Changes the in-game flags used by the automap for the line with the id.

The flags are as follows (multiple flags can be combined by using the | character between the constant names):

AMLF_Secret (1): Line is a secret, so it is drawn as one-sided.

AMLF_DontDraw (2): Line is not drawn (hidden). This flag respects the setting of the am_cheat and am_showalllines console variables. This flag takes precedence over AMLF_Mapped and AMLF_Revealed.

AMLF_Mapped (4): Line is always drawn, whether it has already been seen or not.

AMLF_Revealed (8): Line is drawn as if revealed by a map revealer, such as the computer area map.

Line_SetAutomapStyle

282:Line_SetAutomapStyle (lineid, style)

lineid: The id of the line of which to set the forced-style.

style: The kind of line the line should mimic.

Forces a line to be displayed as if it had specific criteria in the automap.

The styles to set are as follows:

AMLS_Default (0): Automap style is based on line properties (default).

AMLS_OneSided (1): One-sided wall.

AMLS_TwoSided (2): Two-sided wall.

AMLS_FloorDiff (3): Floor levels of front and back sectors are different.

AMLS_CeilingDiff (4): Ceiling levels of front and back sectors are different.

AMLS_ExtraFloor (5): 3D floor border.

AMLS_Special (6): Wall with special non-door action.

AMLS_Secret (7): Secret door.

AMLS_NotSeen (8): Wall not seen yet.

AMLS_Locked (9): Locked door.

AMLS_IntraTeleport (10): Intra-level teleporter.

AMLS_InterTeleport (11): Inter-level or game-ending teleporter.

AMLS_UnexploredSecret (12): Unexplored secret wall.

AMLS_Portal (13): Portal line.

Line_SetBlocking

55:Line_SetBlocking (lineid, setflags, clearflags)

lineid: The ID of the line(s) to affect.

setflags: The blocking flags to set.

clearflags: The blocking flags to clear.

Alters the types of things which are blocked from crossing the linedefs with the specified lineid. Any block types present in setflags are turned on, while those present in clearflags are turned off. Types not specified in either argument are left alone.

The following block types are available for use in either of the flags arguments:

BLOCKF_CREATURES (1): Blocks walking things (players and enemies)

BLOCKF_MONSTERS (2): Blocks monsters (but not players)

BLOCKF_PLAYERS (4): Blocks players

BLOCKF_FLOATERS (8): Blocks floating creatures

BLOCKF_PROJECTILES (16): Blocks projectiles

BLOCKF_EVERYTHING (32): Blocks all of the above

BLOCKF_RAILING (64): Emulates Strife's railing behavior (blocks actors under 32 units of the line)

BLOCKF_USE (128): Blocks switches from being used across the line

BLOCKF_SIGHT (256): Blocks monster line of sight

BLOCKF_HITSCAN (512): Blocks hitscan attacks

BLOCKF_SOUND (1024): Blocks sound

BLOCKF_LANDMONSTERS (2048): Blocks land monsters (New from 4.10.0)

Line_SetIdentification

121:Line_SetIdentification (lineid, moreflags, reserved1, reserved2, lineid_hi)

Warning: This special exists only in the Hexen map format. The line ID and all flags available with this special can instead be set directly within the linedef properties of a UDMF map, giving this special no reason to be available in this format.

lineid: Identification number for this line

moreflags: Allows setting of line flags that don't fit in the flag word anymore.

lineid_hi: High byte of the line id

Used to identify this line for certain specials and ACS commands, as well as to give it additional linedef flags that are not supported by the Hexen map format.

The following bits are defined for moreflags:

Value	UDMF name	Description
1	zoneboundary	Defines a zone boundary for sound environments.
2	jumpover	Defines a railing. A railing has the lowest 32 map units blocked but is passable above that.
4	blockfloaters	Blocks floating monsters.
8	clipmidtex	Clip mid textures to floor and ceiling.
16	wrapmidtex	Wrap mid textures so that they fill the entire height between floor and ceiling. This implies clipping them to floor and ceiling.
32	midtex3d	3dMidtex: Treats the mid texture on this linedef as a part of solid geometry. The texture part of such a linedef blocks actor movement (like a railing) and players and monsters can walk on such textures. Linedefs with this flag set can be attached to moving sectors via the Sector_Attach3dMidtex action special.
64	checkswitchrange	Performs a check whether a switch is reachable for the player before activating it. If a switch is too high or low or completely in the ground or ceiling it will not activate.
128	firstsideonly	Line can only be triggered from the front side.

How to use the high byte in lineid_hi:

Parameter 1 has the low byte, and parameter 5 has the high byte, so the lineid is $\text{parm1} + (\text{parm5} \times 256)$. This lets you affect lineids higher than 255. For example, if the lineid is 4 then the low byte is 4 and the high byte is 0. But if the lineid is 3027, the low byte is 211 and the high byte is 11 ($11 \times 256 + 211 = 3027$).

Note: For the original Hexen maps, or any maps using the Hexen MAPINFO format, the 2nd–5th parameters of this special are ignored. This is to restore compatibility with classic maps which may have set these parameters even though they were not used at the time. To ensure that this special works as expected, make sure to use ZDoom's new MAPINFO format as defined here.

Examples

in 'linedefs mode', right-click on a line, and at the 'action' textbox, input #121

from then on, fill the options that appear.

Use the lineid you set in your scripts.

Line_SetPortal

156:Line_SetPortal (targetline, thisline, type, planeanchor)

Note: This special behaves slightly differently with the Hexen map format compared to the UDMF map format. The thisline parameter, noted below, is unnecessary with UDMF because the linedef structure has a dedicated line ID property. This parameter should be set to 0 when using the UDMF format.

Usage

Sets a portal upon a line. Once created, a portal can be redefined with Line_SetPortalTarget.

The line should have enough empty space to allow for something to pass through it. The line should also be double sided unless on a polyobject.

This special will not allow to create crossable portals on one-sided lines unless it's a polyobject.

Parameters

Creates a portal between two linedefs in a similar fashion to stacked sectors.

targetline: The tag number (line ID) of the line that will act as the "exit" of the current portal.

thisline: The tag number of the current line. In UDMF, this argument should be 0.

type: The type of portal.

0: Visual only. Actors cannot interact with this portal.

1: Visual plus simple teleporter. Triggering this line will cause the actor to be teleported to the "exit" line.

2: Interactive. Has all features of the teleporter type but allows certain other actions to recognize the portal.

3: Static. The portal should work like the linked portals in Eternity Engine, including its same limitations. They cannot be redefined.

NOTE: If your map has two or more contiguous sections connected to each other through two or more portals, those pairs of portals must be completely identical in how far away each one is from the other and the lengths and angles of the lines. See this explanation for details.

4: Eternity-compatibility. This is reserved for XLAT to make this line special compatible with the Eternity definition. Functionally this is identical to type 3, but with different parameter use. A line with a special Line_SetPortal(0, id, 4) will create a portal with another line with a special Line_SetPortal(1, id, 4). For the 'id' parameter the same rules as for the other types apply when being used in UDMF.

planeanchor: Determines how the planes at the other side of the portal are relative to this line. Only portals of type 0 and 1 can have an alignment other than 'none'.

0: No alignment. Neither the floor nor ceiling at the other side of the portal will be aligned. The sectors at both sides should have similar z-heights.

1: Align floors. Both sides of the portal will be aligned so the floor heights match.

2: Align ceilings. Both sides of the portal will be aligned so the ceilings match.

Conversions from linedef types

The following Doom map format types can be converted as :

Type	Conversion	Trigger
------	------------	---------

Eternity 376:Portal_LinkedLineToLine	Line_SetPortal (0, tag, 4)	
--------------------------------------	----------------------------	--

Eternity 377:Portal_LinkedLineToLineAnchor	Line_SetPortal (1, tag, 4)	
--	----------------------------	--

Line_SetPortalTarget

107:Line_SetPortalTarget (sourceline, targetline)

Usage

Changes a line portal's destination. Static portals cannot be redefined.

Parameters

Changes a portal created by Line_SetPortal.

sourceline: The line ID of the portal.

targetline: The line ID of the new destination.

Line_SetTextureOffset

53:Line_SetTextureOffset (lineid, x, y, side, flags)

Usage

Changes the offset of a texture at run-time. This can be used during runtime for special effects, and it is possible to set the offsets of upper, mid, and lower textures separately. This special cannot be used directly on a line in the map, but must be called from an ACS script instead.

side can be SIDE_FRONT or SIDE_BACK.

x and y are fixed point values; using NO_CHANGE leaves it alone.

flags can be a combination of the following flags:

TEXFLAG_TOP (1): upper texture

TEXFLAG_MIDDLE (2): mid texture

TEXFLAG_BOTTOM (4): lower texture

TEXFLAG_ADDOFFSET (8): add to offset (as opposed to directly setting it)

Examples

This script will move the middle texture on a line with an id of 1 in a clockwise circular pattern.

script 1 OPEN

```
{
    int a, x, y;
    while(TRUE)
    {
        for(a=0; a<1.0; a+=512)
        {
            x = FixedMul(cos(a), 128.0);
            y = FixedMul(sin(a), 128.0);
            Line_SetTextureOffset(1, x, y, SIDE_FRONT, TEXFLAG_MIDDLE);
            delay(1);
        }
    }
}
```

Line_SetTextureScale

56:Line_SetTextureScale (lineid, x, y, side, flags)

Usage

Changes the scale of a texture at run-time. This can be used during runtime for special effects, and it is possible to set the scales of upper, mid, and lower textures separately. This special cannot be used directly on a line in the map, but must be called from an ACS script instead. If you are using the UDMF format and do not intend to adjust texture scales later on, you can use the native scaling features located under the sidedefs custom tab.

side can be LINE_FRONT or LINE_BACK.

x and y are fixed point values; using NO_CHANGE leaves it alone.

flags can be a combination of the following flags:

1: upper texture

2: mid texture

4: lower texture

8: apply to current scale (as opposed to directly setting it)

Examples

This example sets the lines with the matching line id into a sort of pulsing growing and shrinking pattern. It uses a scaled sin and cos wave pattern for the scale values. Wonky!

```
script 1 (int lineid)
{
    while (TRUE)
    {
        for (int angle=0; angle<1.0; angle+=0.01)
        {
            int x = 1.0 + cos(angle) / 4;
            int y = 1.0 + sin(angle) / 4;
            Line_SetTextureScale(lineid, x, y, SIDE_FRONT, 2);
            delay(1);
        }
    }
}
```

NoiseAlert

173:NoiseAlert (target_tid, emitter_tid)

target_tid: Target (usually player). If 0, refers to the activator of the script.
emitter_tid: Thing that emits alert noise. If 0, refers to the activator of the script.
This special alerts monsters of a target's presence. You can specify both the object which is doing this alert and the target which the monsters should attack. If both parameters are 0, the special's activator is both emitter and target.

Which monsters are woken up is determined in a similar way to the player shooting: lines with the block sounds flag and sectors with 0 height will not allow the imaginary sound to travel past them. Note that if the monster's AMBUSH flag is set, it will not wake up immediately; the monster will wait until it has a line of sight with the player, which is the normal behavior of the ambush flag.

Examples

```
// Alerts monsters near thing with assigned tag = 667
// to the player's presence.
script 1 (void)
{
    NoiseAlert(0, 667);
}
// Alerts monsters near thing with assigned tag = 667
// to the presence of another actor with tag 668 (may
// or may not be a player).
script 2 (void)
{
    NoiseAlert(668, 667);
}
```

Conversions from linedef types

The following Doom map format types can be converted as NoiseAlert:

Type	Conversion	Trigger	Strife 150:WR Raise Alarm	NoiseAlert (0, 0)	Player Cross, Repeatable
------	------------	---------	---------------------------	-------------------	--------------------------

Pillar_Build

29:Pillar_Build (tag, speed, height)

tag: Tag of affected sector

speed: Speed of the build

height: Height (relative to floor) where the floor and ceiling meet

Raises the floor of a sector and lowers its ceiling so that they meet. The speed argument controls the speed of whichever part of the pillar has to move further, and the other part will have its speed set so that it arrives at its destination at the same time. If height is 0, then the floor and ceiling will meet halfway between their original positions. If tag is 0, then the sector on the line's back side is used.

Pillar_BuildAndCrush

94:Pillar_BuildAndCrush (tag, speed, height, crush [,crushmode])

tag: Tag of affected sector

speed: Speed of the build

height: Height (relative to floor) where the floor and ceiling meet

crush: Amount of damage to apply

crushmode: Sets the crushing mode

This is the same as Pillar_Build except that it will also apply crushing damage to anything blocking the pillar from closing.

Pillar_Open

30:Pillar_Open (tag, speed, fdist, cdist)

tag: Tag of affected sector

speed: Speed of the open

fdist: How far the floor should lower

cdist: How far the ceiling should rise

Opens a pillar. If fdist is 0, then the pillar's floor will lower to the lowest surrounding floor. Similarly, if cdist is 0, then the pillar's ceiling will rise to the highest surrounding ceiling. The speed argument controls the speed of whichever part of the pillar has to move further, and the other part will have its speed set so that it arrives at its destination at the same time. If tag is 0, then the sector on the line's back side is used.

Note: This will only work on sectors that have equal ceiling and floor heights.

Plane_Align

181:Plane_Align (floor, ceiling, lineid)

Warning: This special is not fully supported by the UDMF map format. Certain parameters, noted below, have been made obsolete by the ability to directly specify properties within the linedef structure. These parameters should be set to 0 when using the UDMF format.

floor: "1" slopes the floor on front side of the line; "2" slopes the floor on the back side.

ceiling: Same as for floor, but for the ceiling plane, respectively.

lineid: also sets this line's id if non-zero. Obsolete in UDMF.

Note: A value of 0 means no sloping is applied.

Aligns the plane of the floor and/or ceiling depending on the values floor and ceiling. For a more extensive look, check out slopes.

Examples

An example WAD can be found [here](#).

Conversions from linedef types

The following Doom map format types can be converted as Plane_Align:

Type Conversion Trigger

ZDoom 340:Slope Front Floor	Plane_Align (1, 0)
ZDoom 341:Slope Front Ceiling	Plane_Align (0, 1)
ZDoom 342:Slope Front Floor & Ceiling	Plane_Align (1, 1)
ZDoom 343:Slope Back Floor	Plane_Align (2, 0)
ZDoom 344:Slope Back Ceiling	Plane_Align (0, 2)
ZDoom 345:Slope Back Floor & Ceiling	Plane_Align (2, 2)
ZDoom 346:Slope Back Floor & Front Ceiling	Plane_Align (2, 1)
ZDoom 347:Slope Front Floor & Back Ceiling	Plane_Align (1, 2)
Eternity 386:Slope_FrontsectorFloor	Plane_Align (1, 0)
Eternity 387:Slope_FrontsectorCeiling	Plane_Align (0, 1)
Eternity 388:Slope_FrontsectorFloorAndCeiling	Plane_Align (1, 1)
Eternity 389:Slope_BacksectorFloor	Plane_Align (2, 0)
Eternity 390:Slope_BacksectorCeiling	Plane_Align (0, 2)
Eternity 391:Slope_BacksectorFloorAndCeiling	Plane_Align (2, 2)
Eternity 392:Slope_BackFloorAndFrontCeiling	Plane_Align (2, 1)
Eternity 393:Slope_BackCeilingAndFrontFloor	Plane_Align (1, 2)

Plane_Copy

118:Plane_Copy (front floor, front ceiling, back floor, back ceiling, share)

This is an alternative to using slope copy things. On a single-sided linedef, only the first two argument are relevant. As all slope-related specials, it is only valid during initialization.

front floor: Tag of the sector whose plane to copy. The floor plane of the first tagged sector found will be copied for the front sector's floor plane. If this parameter is left to zero, no copy takes place.

front ceiling: Same thing, but for the ceiling plane. The tagged sector's ceiling plane is copied.

back floor: Same as front floor, but for back sector.

back ceiling: Same as front ceiling, but for back sector.

share: if non-zero, the slope from one side of the linedef can be copied by the other side, as such:

1 — front floor slope is copied to back floor slope

2 — back floor slope is copied to front floor slope

4 — front ceiling slope is copied to back ceiling slope

8 — back ceiling slope is copied to front ceiling slope

A front and a ceiling value can be combined, so 5, 6, 9 and 10 are also valid values. Conflicting values such as 3 or 12 are ignored.

Slope copies from tagged sectors are performed first, followed by shared slopes across the line.

Conversions from linedef types

The following Doom map format types can be converted as Plane_Copy:

Type Conversion Trigger

Eternity 394:Slope_FrontFloorToTaggedSlopePlane_Copy (tag, 0)

Eternity 395:Slope_FrontCeilingToTaggedSlope Plane_Copy (0, tag)

Eternity 396:Slope_FrontFloorAndCeilingToTaggedSlope Plane_Copy (tag, tag)

Plat_DownByValue

63:Plat_DownByValue (tag, speed, delay, height)

tag: Tag of affected sector, or 0 to use the line's back sector.

speed: Platform's movement speed. Doom's standard platforms move at 32, and blazing platforms move at 64.

delay: Tics before going back up. Doom's platforms wait 105 tics.

height: Distance to lower, divided by 8.

Lowers a platform by (height * 8) units, waits, and then raises it back to its original position. If tag is 0, then the sector on the line's back side is used.

Plat_DownWaitUpStay

62:Plat_DownWaitUpStay (tag, speed, delay)

tag: Tag of affected sector

speed: Speed of move

delay: Tics before going back up. Vanilla Doom lifts waited for 3 seconds, or 105 tics.

Lowers a platform, waits, and then raises it back to its original position. This is the same as using Plat_DownWaitUpStayLip with a lip of 8. If tag is 0, then the sector on the line's back side is used.

Plat_DownWaitUpStayLip

206:Plat_DownWaitUpStayLip (tag, speed, delay, lip, sound)

tag: Tag of affected sector

speed: Speed of move

delay: Tics before going back up. Vanilla Doom lifts waited for 3 seconds, or 105 tics.

lip: Amount of floor to leave showing when platform is lowered

sound: if 0 the platform sound is used, if 1 the moving floor sound is used

Lowers a platform, waits, and then raises it back to its original position. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types

The following Doom map format types can be converted as DownWaitUpStayLip:

Type	Conversion	Trigger
------	------------	---------

Doom 10:W1 Lift Also Monsters	Plat_DownWaitUpStayLip (tag, 32, 105, 0)	Player Cross, Monsters Activate
-------------------------------	--	---------------------------------

Heretic 10:W1 Lift	Plat_DownWaitUpStayLip (tag, 32, 105, 0)	Player Cross
--------------------	--	--------------

Doom 21:S1 Lift	Plat_DownWaitUpStayLip (tag, 32, 105)	Player Use
-----------------	---------------------------------------	------------

Doom 62:SR Lift	Plat_DownWaitUpStayLip (tag, 32, 105, 0)	Player Use, Repeatable
-----------------	--	------------------------

Doom 88:WR Lift Also Monsters	Plat_DownWaitUpStayLip (tag, 32, 105, 0)	Player Cross, Repeatable, Monsters Activate
-------------------------------	--	---

Heretic 88:WR Lift	Plat_DownWaitUpStayLip (tag, 32, 105, 0)	Player Cross, Repeatable
--------------------	--	--------------------------

Doom 120:WR Lift Fast	Plat_DownWaitUpStayLip (tag, 64, 105, 0)	Player Cross, Repeatable
-----------------------	--	--------------------------

Doom 121:W1 Lift Fast	Plat_DownWaitUpStayLip (tag, 64, 105, 0)	Player Cross
-----------------------	--	--------------

Doom 122:S1 Lift Fast	Plat_DownWaitUpStayLip (tag, 64, 105, 0)	Player Use
-----------------------	--	------------

Doom 123:SR Lift Fast	Plat_DownWaitUpStayLip (tag, 64, 105, 0)	Player Use, Repeatable
-----------------------	--	------------------------

Strife 214:SR Lift Slow	Plat_DownWaitUpStayLip (tag, 8, 1050, 0, 1)	Player Use, Repeatable
-------------------------	---	------------------------

Plat_PerpetualRaise

60:Plat_PerpetualRaise (tag, speed, delay)

tag: Tag of affected sector

speed: Speed of move

delay: Tics between rise and falls

Starts a perpetual raise platform with a lip of 8 units. If tag is 0, then the sector on the line's back side is used.

Plat_PerpetualRaiseLip

207:Plat_PerpetualRaiseLip (tag, speed, delay, lip)

- tag: Tag of affected sector
 - speed: Speed of move
 - delay: Tics between rise and falls
 - lip: Amount of floor to leave showing when platform is lowered
- Starts a perpetual raise platform. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types
The following Doom map format types can be converted as Plat_PerpetualRaiseLip:

Type	Conversion	Trigger
Doom 53:W1 Start Moving Floor	Plat_PerpetualRaiseLip (tag, 8, 105, 0)	Player Cross
Doom 87:WR Start Moving Floor	Plat_PerpetualRaiseLip (tag, 8, 105, 0)	Player Cross, Repeatable
Boom 162:S1 Start Moving Floor	Plat_PerpetualRaiseLip (tag, 8, 105, 0)	Player Use
Boom 181:SR Start Moving Floor	Plat_PerpetualRaiseLip (tag, 8, 105, 0)	Player Use, Repeatable

Plat_RaiseAndStayTx0

228:Plat_RaiseAndStayTx0 (tag, speed, lockout)

tag: Tag of affected sector

speed: Speed of move

lockout: Whether the platform can move again afterwards.

Raises a platform to the next highest floor, sets its floor texture to match the floor texture on the front side of the triggering line, and sets the plat's sector special to zero. If tag is 0, then the sector on the line's back side is used.

If lockout is 0, the behavior depends on the game. In Heretic, to emulate a vanilla bug, the platform sector's floor becomes locked out and no other form of sector movement will work on it. In other games, this behaves normally, without locking up. A value of 1 forces the correct behavior in all games, including Heretic; and a value of 2 forces the bugged behavior in all games.

Conversions from linedef types

The following Doom map format types can be converted as Plat_RaiseAndStayTx0:

Type	Conversion	Trigger
Doom 20:S1 Floor To Higher Floor Change Tex	Plat_RaiseAndStayTx0 (tag, 4)	Player Use
Doom 22:W1 Floor To Higher Floor Change Tex	Plat_RaiseAndStayTx0 (tag, 4)	Player Cross
Doom 47:G1 Floor To Higher Floor Change Tex	Plat_RaiseAndStayTx0 (tag, 4)	Attack Hit, Missile Cross
Doom 68:SR Floor To Higher Floor Change Tex	Plat_RaiseAndStayTx0 (tag, 4)	Player Use, Repeatable
Doom 95:WR Floor To Higher Floor Change Tex	Plat_RaiseAndStayTx0 (tag, 4)	Player Cross, Repeatable

Plat_Stop

61:Plat_Stop (tag)

tag: Tag of affected sector
Stops a perpetual raise platform.

Conversions from linedef types
The following Doom map format types can be converted as Plat_Stop:

Type	Conversion	Trigger
Doom 54:W1 Stop Moving Floor	Plat_Stop(tag)	Player Cross
Strife 54:W1 Stop Moving Floor	Plat_Stop(tag)	Player Cross
Doom 89:WR Stop Moving Floor	Plat_Stop(tag)	Player Cross, Repeatable
Strife 89:WR Stop Moving Floor	Plat_Stop(tag)	Player Cross, Repeatable
Boom 163:S1 Lift Stop	Plat_Stop(tag)	Player Use
Boom 182:SR Lift Stop	Plat_Stop(tag)	Player Use, Repeatable

Plat_ToggleCeiling

231:Plat_ToggleCeiling (tag)

tag: Tag of affected sector
Switches a platform between lowered and touching the ceiling. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types
The following Doom map format types can be converted as Plat_ToggleCeiling:

Type	Conversion	Trigger
Boom 211:SR Toggle Floor To Ceiling (Instant)	Plat_ToggleCeiling (tag)	Player Use, Repeatable
Boom 212:WR Toggle Floor To Ceiling (Instant)	Plat_ToggleCeiling (tag)	Player Cross, Repeatable

Plat_UpByValue

65:Plat_UpByValue (tag, speed, delay, height)

tag: Tag of affected sector

speed: Speed of move

delay: Tics before going back down

height: Amount to raise by

Raises a platform by (height * 8) units, waits, and then lowers it back to its original position. If tag is 0, then the sector on the line's back side is used.

Plat_UpByValueStayTx

230:Plat_UpByValueStayTx (tag, speed, height)

tag: Tag of affected sector

speed: Speed of move

height: Amount to raise by

Raises a platform by (height * 8) units, and sets its floor texture to match the floor texture on the front side of the triggering line. If tag is 0, then the sector on the line's back side is used.

Conversions from linedef types

The following Doom map format types can be converted as Plat_UpByValueStayTx:

Type	Conversion	Trigger
Doom 14:S1 Floor Up 32 Change Texture	Plat_UpByValueStayTx (tag, 4, 4)	Player Use
Doom 15:S1 Floor Up 24 Change Texture	Plat_UpByValueStayTx (tag, 4, 3)	Player Use
Doom 66:SR Floor Up 24 Change Texture	Plat_UpByValueStayTx (tag, 4, 3)	Player Use, Repeatable
Doom 67:SR Floor Up 32 Change Texture	Plat_UpByValueStayTx (tag, 4, 4)	Player Use, Repeatable
Boom 143:W1 Lift Up 24 Change Texture	Plat_UpByValueStayTx (tag, 4, 3)	Player Cross
Boom 144:W1 Lift Up 32 Change Texture	Plat_UpByValueStayTx (tag, 4, 4)	Player Cross
Boom148:WR Lift Up 24 Change Texture	Plat_UpByValueStayTx (tag, 4, 3)	Player Cross, Repeatable
Boom 149:WR Lift Up 32 Change Texture	Plat_UpByValueStayTx (tag, 4, 4)	Player Cross, Repeatable

Plat_UpNearestWaitDownStay

172:Plat_UpNearestWaitDownStay (tag, speed, delay)

tag: Tag of affected sector

speed: Speed of move

delay: Tics before going back down

Raises the tagged sector's floor to the height of the next-highest neighboring floor, waits for delay tics, then lowers to its original height.

Conversions from linedef types

The following Doom map format types can be converted as Plat_UpNearestWaitDownStay:

Type	Conversion	Trigger
Strife 155:SR Lift Up	Plat_UpNearestWaitDownStay (tag, 16, 105)	Player Use, Repeatable

Plat_UpWaitDownStay

64:Plat_UpWaitDownStay (tag, speed, delay)

tag: Tag of affected sector

speed: Speed of move

delay: Tics before going back down

Raises a platform, waits, and then lowers it back to its original position. If tag is 0, then the sector on the line's back side is used.

Player_SetTeam

Warning: This feature is Skulltag specific, and is not compatible with ZDoom!
To see all of Skulltag's specific features, see Skulltag features.

145:Player_SetTeam (team)

team: What team to set the player to.

Sets the player's team. Usually used at the start of a CTF map in the team selection area. A valid team is anywhere between 0 and sv_maxteams (so 2, 3, 4). The teams are 0 = Blue, 1 = Red, 2 = Green, 3 = Gold. Using the current value of sv_maxteams removes the player from his current team without making him join another.

PointPush_SetForce

227:PointPush_SetForce (tag, tid, amount, useline)

tag: Tag of sector containing the point pusher/puller

tid: Thing ID of the point pusher/puller

amount: Strength of the point pusher/puller

useline: Set to 1 to use the line's length to determine strength

Only valid during initialization. Sets the amount of force for a point pusher or puller. If tag is non-zero, it looks for point pusher/pullers in the tagged sectors. Otherwise, it looks for them by their tids. If useline is 1, then the amount of force is determined by the length of the linedef. Otherwise, the amount parameter is used.

Conversions from linedef types

The following Doom map format types can be converted as PointPush_SetForce:

Type	Conversion	Trigger
Boom 226:Set Push	PointPush_SetForce (tag, 0, 0, 1)	

Polyobj_DoorSlide

8:Polyobj_DoorSlide (po, speed, angle, dist, delay)

Moves a polyobject, waits, and then moves it back to its original location.

po: polyobject to move

speed: how quickly to move the polyobject

angle: direction to move the polyobject in (this is a byte angle)

dist: distance to move the polyobject

delay: delay in tics before returning to original position

Polyobj_DoorSwing

7:Polyobj_DoorSwing (po, speed, angle, delay)

Rotates a polyobject, waits, and then rotates it in the opposite direction until it has returned to its original orientation.

po: polyobj to move

speed: how quickly to spin the polyobj

angle: byte angle to rotate the polyobj through

delay: delay in tics before returning to original orientation

By default, the polyobj rotates in a counter-clockwise direction.

Using this function in ACS, or when using UDMF, you can reverse that direction by supplying a negative value to the speed parameter.

Polyobj_ExplicitLine

5:Polyobj_ExplicitLine (po, order, mirror, sound, lineid)

Warning: This special is not fully supported by the UDMF map format. Certain parameters, noted below, have been made obsolete by the ability to directly specify properties within the linedef structure. These parameters should be set to 0 when using the UDMF format.

Explicitly includes a line as part of a polyobject.

po: polyobj that is being defined

order: rendering order of this line

mirror: polyobj that will mirror the moves of this one

sound: door sound sequence to play when this polyobj moves

lineid: also sets this line's id if non-zero. Obsolete in UDMF.

Polyobj_Move

4:Polyobj_Move (po, speed, angle, dist)

Moves a polyobject.

po: polyobj to move

speed: how quickly the polyobj should move

angle: direction the polyobj should move (this is a byte angle)

dist: distance to move

Polyobj_MoveTimes8

6::Polyobj_MoveTimes8 (po, speed, angle, dist)

Moves a polyobject (dist * 8) units.

po: polyobj to move

speed: how quickly the polyobj should move

angle: direction the polyobj should move (this is a byte angle)

dist: distance to move in units of 8

See Also

Polyobj_MoveTo

88:Polyobj_MoveTo (po, speed, pos_x, pos_y)

Moves a polyobject in a straight line towards an absolute destination.

po: polyobj to move

speed: how quickly the polyobject should move, in map units per octic.

pos_x: the absolute X coordinate to which the polyobject must move.

pos_y: the absolute Y coordinate to which the polyobject must move.

Polyobj_MoveToSpot

86:Polyobj_MoveToSpot (po, speed, target)

Moves a polyobject in a straight line towards a given destination.

po: polyobj to move

speed: how quickly the polyobject should move, in map units per octic.

target: the tid of the spot to which the polyobject should move.

Polyobj_OR_Move

92:Polyobj_OR_Move (po, speed, angle, distance)

Moves a polyobject.

po: polyobj to move

speed: how quickly the polyobj should move

angle: direction the polyobj should move (this is a byte angle)

dist: distance to move

The "OR" in this special stands for OverRide. Under normal circumstances, if a polyobject is doing something, you cannot make it do something else until it has finished whatever it is doing. This can be a problem with perpetual polyobjects (such as Polyobj_RotateLeft or Right with a byte angle of 255). Using one of these four specials, you can force the polyobject to stop whatever it is doing and do something else.

Polyobj_OR_MoveTimes8

93:Polyobj_OR_MoveTimes8 (po, speed, angle, distance)

Moves a polyobject (dist * 8) units.

po: polyobj to move

speed: how quickly the polyobj should move

angle: direction the polyobj should move (this is a byte angle)

dist: distance to move in units of 8

The "OR" in this special stands for OverRide. Under normal circumstances, if a polyobject is doing something, you cannot make it do something else until it has finished whatever it is doing. This can be a problem with perpetual polyobjects (such as Polyobj_RotateLeft or Right with a byte angle of 255). Using one of these four specials, you can force the polyobject to stop whatever it is doing and do something else.

Polyobj_OR_MoveTo

89:Polyobj_OR_MoveTo (po, speed, pos_x, pos_y)

Moves a polyobject in a straight line towards an absolute destination.

po: polyobj to move

speed: how quickly the polyobject should move, in map units per octic.

pos_x: the absolute X coordinate to which the polyobject must move.

pos_y: the absolute Y coordinate to which the polyobject must move.

The "OR" in this special stands for OverRide. Under normal circumstances, if a polyobject is doing something, you cannot make it do something else until it has finished whatever it is doing. This can be a problem with perpetual polyobjs (such as Polyobj_RotateLeft or Polyobj_RotateRight with a byte angle of 255). Using one of the "OR" specials, you can force the polyobj to stop whatever it is doing and do something else.

Polyobj_OR_MoveToSpot

Jump to navigationJump to search

59:Polyobj_OR_MoveToSpot (po, speed, target)

Moves a polyobject in a straight line towards a given destination.

po: polyobj to move

speed: how quickly the polyobject should move, in map units per octic.

target: the tid of the spot to which the polyobject should move.

The "OR" in this special stands for OverRide. Under normal circumstances, if a polyobject is doing something, you cannot make it do something else until it has finished whatever it is doing. This can be a problem with perpetual polyobjs (such as Polyobj_RotateLeft or Polyobj_RotateRight with a byte angle of 255). Using one of the "OR" specials, you can force the polyobj to stop whatever it is doing and do something else.

Polyobj_OR_RotateLeft

90:Polyobj_OR_RotateLeft (po, speed, angle)

Rotates a polyobject right through the specified angle. If angle is 255, then the polyobject will rotate continuously and never stop.

po: polyobj to rotate

speed: how quickly the polyobj should rotate

angle: byte angle to rotate the polyobj through

The "OR" in this special stands for OverRide. Under normal circumstances, if a polyobject is doing something, you cannot make it do something else until it has finished whatever it is doing. This can be a problem with perpetual polyobjs (such as Polyobj_RotateLeft or Polyobj_RotateRight with a byte angle of 255). Using one of these four specials, you can force the polyobj to stop whatever it is doing and do something else.

Polyobj_OR_RotateRight

91:Polyobj_OR_RotateRight (po, speed, angle)

Rotates a polyobject right through the specified angle. If angle is 255, then the polyobject will rotate continuously and never stop.

po: polyobj to rotate

speed: how quickly the polyobj should rotate

angle: byte angle to rotate the polyobj through

The "OR" in this special stands for OverRide. Under normal circumstances, if a polyobject is doing something, you cannot make it do something else until it has finished whatever it is doing. This can be a problem with perpetual polyobjects (such as Polyobj_RotateLeft or Right with a byte angle of 255). Using one of these four specials, you can force the polyobj to stop whatever it is doing and do something else.

Polyobj_RotateLeft

2:Polyobj_RotateLeft (po, speed, angle)

Rotates a polyobject left through the specified angle. If angle is 255, then the polyobject will rotate continuously and never stop.

po: polyobj to rotate

speed: how quickly the polyobj should rotate

angle: byte angle to rotate the polyobj through

Polyobj_RotateRight

3:Polyobj_RotateRight (po, speed, angle)

Rotates a polyobject right through the specified angle. If angle is 255, then the polyobject will rotate continuously and never stop.

po: polyobj to rotate

speed: how quickly the polyobj should rotate

angle: byte angle to rotate the polyobj through

Polyobj_StartLine

1:Polyobj_StartLine (po, mirror, sound, lineid)

Warning: This special is not fully supported by the UDMF map format. Certain parameters, noted below, have been made obsolete by the ability to directly specify properties within the linedef structure. These parameters should be set to 0 when using the UDMF format.

Usage

Starts defining a polyobject.

po: which polyobj is being defined.

mirror: polyobj that will mirror this one's movements.

sound: door sound sequence to play when this polyobj moves.

lineid: also sets this line's ID if non-zero. Obsolete in UDMF.

Polyobj_Stop

87:Polyobj_Stop (po)

Stops a polyobject's movement instantly.

po: polyobj to stop

Radius_Quake

120:Radius_Quake (intensity, duration, damrad, tremrad, tid)

intensity: Strength of earthquake, ranging from 0 to 9

duration: Duration in tics

damrad: Radius of damage in 64x64 cells

tremrad: Radius of tremor in 64x64 cells

tid: Thing ID of map thing(s) for quake foci (if 0, the activator is used as the sole focus)

Creates an earthquake at the specified foci (map spots).

Scroll_Ceiling

line version:
224:Scroll_Ceiling (tag, scrollbits, unused, x-move, y-move)

script version:
224:Scroll_Ceiling (tag, x-move, y-move, unused)

tag: tag of affected sector
scrollbits: how the scrolling speed is determined (see below)
unused: does nothing, contrary to Scroll_Floor.
x-move/y-move: how quickly and in what directions to scroll.
scrollbits: Selects the type of scroller and how the speed and angle is determined.

bit 0(1): Displacement scroller
bit 1(2): Accelerative scroller
bit 2(4): Use this linedef to get dx and dy
This special takes 5 arguments when placed on a line, but 4 arguments when used in a script.

x-move and y-move are only used if scrollbits does not indicate to use the linedef to determine the scroll rate. When placed on a line, 128 is subtracted from these values to determine the actual direction and rate of scroll. (So 128 would be no scroll.) When used in a script, positive values move north/east, while negative values move south/west.

NOTE: When using Doom Builder 2 the scripting popup help will show the line version instead of the script version. Do not follow it, use the script version.

Examples
The following results in a ceiling (with the tag of 14) that quickly scrolls to the north.

```
script 1 OPEN
{
    scroll_ceiling (14, 0, 150, 0);
}
```

Conversions from linedef types
The following Doom map format types can be converted as Scroll_Ceiling:

Type	Conversion	Trigger
Boom 214:Accel Tagged Ceiling w.r.t. 1st Side's Sector	Scroll_Ceiling (tag, 6, 0, 0, 0)	
Boom 245:Scroll Tagged Ceiling w.r.t. 1st Side's Sector	Scroll_Ceiling (tag, 5, 0, 0, 0)	
Boom 250:Scroll Tagged Ceiling	Scroll_Ceiling (tag, 4, 0, 0, 0)	

Scroll_Floor

line version:

223:Scroll_Floor (tag, scrollbits, type, x-move, y-move)

script version:

223:Scroll_Floor (tag, x-move, y-move, type)

This special takes 5 arguments when placed on a line, but 4 arguments when used in a script.

tag: tag of affected sector

scrollbits: how the scrolling speed is determined (see below)

type: set to 0 to scroll the floor texture. A setting of 1 pushes objects along the floor without scrolling the texture. 2 does both. When using it in a script, definitions exist for the type field to allow easier readability.

SCROLL " Scrolls the floor

CARRY " Moves objects without scrolling

SCROLL_AND_CARRY " Does both

x-move/y-move: how quickly and in what directions to scroll.

scrollbits: Selects the type of scroller and how the speed and angle is determined. (Only when placed on a line)

bit 0(1): Displacement scroller

bit 1(2): Accelerative scroller

bit 2(4): Use this linedef to get dx and dy

Scroll_Floor can also only scroll the flat, only carry objects, or scroll and carry objects depending on what the type argument is set to. x-move and y-move are only used if scrollbits does not indicate to use the linedef to determine the scroll rate. When placed on a line, 128 is subtracted from these values to determine the actual direction and rate of scroll. (So 128 would be no scroll.) When used in a script, positive values move north/east, while negative values move south/west.

Conversions from linedef types

The following Doom map format types can be converted as Scroll_Floor:

Type	Conversion	Trigger
------	------------	---------

Boom 215:Accel Tagged Floor w.r.t. 1st Side's Sector	Scroll_Floor (tag, 6, 0, 0, 0)	
--	--------------------------------	--

Boom 216:Accel Objects on Tagged Floor wrt 1st Side's Sector	Scroll_Floor (tag, 6, 1, 0, 0)	
--	--------------------------------	--

Boom 217:Accel Objects&Tagged Floor wrt 1st Side's Sector	Scroll_Floor (tag, 6, 2, 0, 0)	
---	--------------------------------	--

Boom 246:Scroll Tagged Floor w.r.t. 1st Side's Sector	Scroll_Floor (tag, 5, 0, 0, 0)	
---	--------------------------------	--

Boom 247:Push Objects on Tagged Floor wrt 1st Side's Sector	Scroll_Floor (tag, 5, 1, 0, 0)	
---	--------------------------------	--

Boom 248:Push Objects & Tagged Floor wrt 1st Side's Sector	Scroll_Floor (tag, 5, 2, 0, 0)	
--	--------------------------------	--

Boom 251:Scroll Tagged Floor	Scroll_Floor (tag, 4, 0, 0, 0)	
------------------------------	--------------------------------	--

Boom252:Carry Objects on Tagged Floor	Scroll_Floor (tag, 4, 1, 0, 0)	
---------------------------------------	--------------------------------	--

Boom 253:Scroll Tagged Floor, Carry Objects	Scroll_Floor (tag, 4, 2, 0, 0)	
---	--------------------------------	--

Scroll_Texture_Both

221:Scroll_Texture_Both (lineid, left, right, down, up)

lineid: Line ID of line to scroll

left: Speed to scroll left

right: Speed to scroll right

down: Speed to scroll down

up: Speed to scroll up

Scrolls a texture both horizontally and vertically. If the lineid is 0, this special will affect the speed of the texture scrolling on the line it is placed on. The other four parameters determine how quickly the texture scrolls in each direction.

If you use a non-zero lineid, you can change the scrolling of the specified lines when this special is activated (either directly or in a script). A positive lineid changes the scrolling on the front of the line, and a negative lineid changes the scrolling on the back of the line.

Of note: when applying to this property to a row of textures, only the visibly front-facing textures seem to be affected.

Conversions from linedef types

The following Doom map format types can be converted as :

Type	Conversion	Trigger
EDGE 425:Scroll Up & Left	Scroll_Texture_Both	(0, 64, 0, 0, 64)
EDGE 426:Scroll Down & Left	Scroll_Texture_Both	(0, 64, 0, 64, 0)
EDGE 427:Scroll Up & Right	Scroll_Texture_Both	(0, 0, 64, 0, 64)
EDGE 428:Scroll Down & Right	Scroll_Texture_Both	(0, 0, 64, 64, 0)

Scroll_Texture_Down

103:Scroll_Texture_Down (speed [, flags])

speed: Speed to scroll
flags: Which parts of the linedef should scroll
Scrolls a texture downwards.

flags can be a combination of the following values:

- 1: scroll the upper texture
 - 2: scroll the mid texture
 - 4: scroll the lower texture
- For compatibility, 0 and any value larger than 7 will scroll all parts of the linedef.

Conversions from linedef types
The following Doom map format types can be converted as :

Type	Conversion	Trigger
Strife 143:Scrolling Wall Down Triple Speed	Scroll_Texture_Down (192)	
EDGE 424:Scrolling Wall Down	Scroll_Texture_Down (64)	

Scroll_Texture_Left

100:Scroll_Texture_Left (speed [, flags])

speed: Speed to scroll
flags: Which parts of the linedef should scroll
Scrolls a texture to the left.

flags can be a combination of the following values:

- 1: scroll the upper texture
 - 2: scroll the mid texture
 - 4: scroll the lower texture
- For compatibility, 0 and any value larger than 7 will scroll all parts of the linedef.

The Doom linetype 48 is translated to Scroll_Texture_Left(64, 0).

Conversions from linedef types
The following Doom map format types can be converted as Scroll_Texture_Left:

Type	Conversion	Trigger
Doom 48:Scrolling Wall Left	Scroll_Texture_Left (64)	

Scroll_Texture_Model

222:Scroll_Texture_Model (lineid, scrollbits)

Warning: This special is not fully supported by the UDMF map format. Certain parameters, noted below, have been made obsolete by the ability to directly specify properties within the linedef structure. These parameters should be set to 0 when using the UDMF format.

lineid: The ID of the line to be affected. Obsolete in UDMF.

scrollbits: The type of scroller:

bit 0(1): Displacement scroller

bit 1(2): Accelerative scroller

Conversions from linedef types

The following Doom map format types can be converted as :

Type	Conversion	Trigger
Boom 218:Accel Tagged Wall w.r.t 1st Side's Sector	Scroll_Texture_Model (lineid, 2)	
Boom 249:Scroll Tagged Wall w.r.t 1st Side's Sector	Scroll_Texture_Model (lineid, 1)	
Boom 254:Scroll Tagged Wall, Same as Floor/Ceiling	Scroll_Texture_Model (lineid, 0)	

Scroll_Texture_Offsets

225:Scroll_Texture_Offsets (flags)

flags: Which parts of the linedef should scroll. Can be a combination of the following values:

- 1: scroll the upper texture
- 2: scroll the mid texture
- 4: scroll the lower texture

For compatibility, 0 and any value larger than 7 will scroll all parts of the linedef.

Scrolls a texture both horizontally and vertically. This special only affects the line it is being placed on. The amount of scrolling is determined from the texture offsets on the first side of this line. Only the first side can be scrolled with this special. This special is mainly provided for Boom compatibility.

Conversions from linedef types

The following Doom map format types can be converted as Scroll_Texture_Offsets:

Type	Conversion	Trigger
Boom 255:Scroll Wall Using Sidedef Offsets		Scroll_Texture_Offsets ()

Scroll_Texture_Right

101:Scroll_Texture_Right (speed [, flags])

speed: Speed to scroll
flags: Which parts of the linedef should scroll
Scrolls a texture to the right.

flags can be a combination of the following values:

- 1: scroll the upper texture
 - 2: scroll the mid texture
 - 4: scroll the lower texture
- For compatibility, 0 and any value larger than 7 will scroll all parts of the linedef.

Conversions from linedef types
The following Doom map format types can be converted as Scroll_Texture_Right:

Type	Conversion	Trigger
Boom 85:Scrolling Wall Right	Scroll_Texture_Right(64)	
Heretic 99:Scrolling Wall Right	Scroll_Texture_Right(64)	
Strife 149:Scrolling Wall Right	Scroll_Texture_Right(64)	
EDGE 422:Scrolling Wall Right	Scroll_Texture_Right(64)	

Scroll_Texture_Up

102:Scroll_Texture_Up (speed [, flags])

speed: Speed to scroll
flags: Which parts of the linedef should scroll
Scrolls a texture upwards.

flags can be a combination of the following values:

- 1: scroll the upper texture
 - 2: scroll the mid texture
 - 4: scroll the lower texture
- For compatibility, 0 and any value larger than 7 will scroll all parts of the linedef.

Conversions from linedef types
The following Doom map format types can be converted as Scroll_Texture_Up:

Type	Conversion	Trigger
Strife 142:Scrolling Wall Up	Scroll_Texture_Up (64)	
EDGE 423:Scrolling Wall Up	Scroll_Texture_Up (64)	

Scroll_Wall

52:Scroll_Wall (lineid, x, y, side, flags)

lineid: The id of the line to apply the scrolling effect to.

x: The horizontal scroll speed as a fixed point value.

y: The vertical scroll speed as a fixed point value.

side: The side of the line to affect. 0 for front; 1 for back.

flags: Which parts of the wall to affect. (see below)

Causes one or more sections on the given side of the indicated line(s) to begin scrolling at the rate specified. Upper, lower, and mid textures can be set to scroll separately. If x and y are both 0, scrolling for the given section of the line will be stopped.

flags specifies which parts of the wall will scroll. This can be a combination of the following values:

1: scroll the upper texture

2: scroll the mid texture

4: scroll the lower texture

Note: Sidedefs with the 3DMIDTEX flag will not scroll because scrolling would interfere with their intended function.

Sector_Attach3dMidtex

48:Sector_Attach3dMidtex (lineid, tag, floororceiling)

lineid: ID of lines to attach

tag: Tag of sector which lines to attach

floororceiling: Which plane of the sector to attach to (0=floor, 1=ceiling)

Attaches lines with the 3dMidtex flag to the front sector of this linedef. If the floor or ceiling of this sector moves, all attached linedefs will move their mid texture accordingly. This allows to create lifts or other moving things purely out of mid textures.

There are 3 modes:

If tag is 0, and lineid is not, attach all lines with the 3dMidtex flag and the matching line ID.

If lineid is 0, and tag is not, attach all lines with the 3dMidtex flag in the sector(s) with the matching tag.

If tag and lineid are both non-zero, attach all lines with the 3dMidtex flag and the matching line ID in the sector(s) with the matching tag.

This special cannot be used in a script. It has to be placed on a linedef of the controlling sector in the map.

Sector_ChangeFlags

Jump to navigationJump to search

54:Sector_ChangeFlags (tag, setflags, clearflags)

Sets or clears certain sector-specific flags.

tag: The tag of the sector(s) to affect.

setflags: The flags to turn on for the specified sector(s).

clearflags: The flags to turn off for the specified sector(s).

Currently supported flags are:

Identifier constant	Value	UDMF name	Description
SECF_SILENT	1	silent	Actors do not emit any sound within this sector.
SECF_NOFALLINGDAMAGE	2	nofallingdamage	Falling damage is disabled on this sector's floor.
SECF_FLOORDROP	4	dropactors	All actors standing on this floor will remain on it when it lowers very fast.
SECF_NORESPAWN	8	norespawn	Players cannot respawn in this sector.
SECF_FRICTION	16	N/A	Friction is enabled on the sector.
SECF_PUSH	32	N/A	Pushers are enabled on the sector.
SECF_SILENTMOVE	64	N/A	Sector's movement makes no sound.
SECF_DMGTERRAINFX	128	damageteraineffect	A terrain splash is spawned at the end of each damage interval.
SECF_DMGENDGODMODE	256	N/A	Getting damaged by the sector ends god mode.
SECF_DMGENLEVEL	512	N/A	Being in the sector ends the level when health goes below 10 points.
SECF_DMGHAZARD	1024	damagehazard	Changes the damage model of the sector to Strife's delayed damage.
SECF_NOATTACK	2048	noattack	Monsters in this sector do not attack.

Sector_ChangeSound

140:Sector_ChangeSound (tag, newsequence)

tag: Tag of affected sector

newsequence: Index of new sound sequence

Changes the sound sequence attached to the sector. This is the same as the one specified by placing one of the sound sequence things (#1400-1411) in a map editor, but with this special the sequence can be changed at run time.

Sector_CopyScroller

58:Sector_CopyScroller (tag, flags)

Causes the line's front sector to copy the scroll effect from a tagged sector.

- tag: The tag of the sector from which to copy the scroll effect
flags: Which parts of the sector to affect. This can be a combination of the following values:
1: copy a ceiling scroller
2: copy a floor scroller
4: copy a carrying effect

Conversions from linedef types
The following Doom map format types can be converted as Sector_CopyScroller:

Type	Conversion	Trigger
ZDoom 352	Sector_CopyScroller (tag, 1)	
ZDoom 353	Sector_CopyScroller (tag, 2)	
ZDoom 354	Sector_CopyScroller (tag, 6)	

Sector_Set3dFloor

160:Sector_Set3dFloor (tag, type, flags, alpha, hi-tag/line ID)

Draws a 3D floor. Use this with a control sector in the same fashion as Transfer_Heights.

The ceiling of the control sector will be the floor of the 3D floor and the floor of the control sector will be the bottom of the 3d floor. The texture used on this line type will be the texture used for the sides of the 3D floor. The height of the 3D floor will be that of the distance between floor and ceiling.

Normally the lighting (including light color) of the control sector is transferred to the volume of the 3D floor (if transparent) and the space below it down to the next lower 3D floor.

Parameters

tag: Tag of affected sectors.

type: The type of 3D floor to create.

flags: The flags to apply to the 3D floor.

alpha: Specifies the translucency of the 3D-floor: 0 = invisible, 255 = opaque.

hi-tag/line ID: If type has the value 8 added to it this will give the control linedef a line ID. Otherwise this parameter serves as a high-byte for the sector tag in order to allow more than 256 3D floor definitions. This parameter is completely unnecessary in UDMF and should not be used in that format.

Types

0: Defines a Vavoom-style 3D-floor. This means that the 3D floor's top surface will correspond to the control sector's floor, and the bottom surface will correspond to the ceiling. In other words, the control sector needs to have a negative height.

1: Defines a solid 3D floor.

2: Defines a swimmable 3D floor.

3: Defines a non-solid 3D-floor.

4: If you add 4 to the type the inside will also be rendered. Normally this is only done for liquids. This does not work for the Vavoom type. Using 4 alone is an "undocumented hack type that can be used to patch some missing texture hacks the engine can't detect".

16: If you add 16 to the type the visibility rules will be inverted. Monsters can see through solid 3D floors but not through non-solid ones. This does not work for the Vavoom type.

32: If you add 32 to the type the shootability rules will be inverted. You can shoot through solid 3D floors but not through non-solid ones. This does not work for the Vavoom type.

Flags

1: Disables any lighting effects created by this 3D floor.

2: Restricts the lighting properties into the area between the 3D floor's top and bottom. Logically this is only useful if the 3D floor is not solid.

4: GZDoom only: 'Fog' effect. Fills all 3D floor surfaces with solid color using control sector's fade color. Also tints the view when inside and below this 3D floor, depending on the lighting settings for the 3D floor. This flag is not implemented in the software renderer and currently produces undefined results!

8: Ignores the bottom height of the model sector and draws top and bottom of the 3D floor at the model sector's ceiling height.

16: Uses a sidedef's upper texture to draw the sides of the 3D floor instead of the transfer linedef's mid texture.

32: Uses a sidedef's lower texture to draw the sides of the 3D floor instead of the transfer linedef's mid texture.

64: Renders the 3D floor using additive translucency.

512: 'Fade' effect. Applies control sector's fade color to the area below this 3D floor.

1024: Resets lighting effects created by 3D floors above this 3D floor.

Important notes

Sloped 3D floors are not available in the software renderer.

Sloped translucent 3D floors cannot be defined. They can be either sloped or translucent, not both.

When making a sloped 3D sector make sure that the control sectors are aligned with the in-game sectors on the editing grid. Odd results may happen if they are not aligned.

Translucent 3D floors may create glitches in the display of XY-billboarded sprites.

A same sector cannot have both a 3D floor and a Transfer_Heights effect.

3D floors can move up and down by moving the model sector's floor and ceiling. But due to the way the Doom engine is doing this, it does not work for the Vavoom-type!

Sector damage on the control sector is transferred to the target sectors. TERRAIN-based damage and friction are transferred as well.

If the control sector has a fade color, it will be used as the palette blend within and underneath the 3D floor.

Actor hits floor / ceiling doesn't work on Swimmable 3D Floors due to a limitation.

For optimization purposes, it is important to understand that this effect works by creating new visplanes to be rendered in the sectors which have the tag. As this is basically how a single sector is rendered anyway, they don't use a large amount of resources, so this should be the preferred method of creating multi-level structures, as opposed to using the resource-intensive portals.

Conversions from linedef types

The following Doom map format types can be converted as Sector_Set3dFloor:

Type	Conversion	Trigger
Legacy 281	Sector_Set3dFloor	(tag, 1, 0, 255)
Legacy 289	Sector_Set3dFloor	(tag, 1, 1, 255)
Legacy 300	Sector_Set3dFloor	(tag, 1, 1, 127)
Legacy 301	Sector_Set3dFloor	(tag, 2, 2, 127)
Legacy 302	Sector_Set3dFloor	(tag, 3, 6, 127)
Legacy 303	Sector_Set3dFloor	(tag, 3)
Legacy 304	Sector_Set3dFloor	(tag, 2, 2, 255)
Legacy 305	Sector_Set3dFloor	(tag, 3, 2)
Legacy 306	Sector_Set3dFloor	(tag, 1)
Legacy 332	Sector_Set3dFloor	(tag, 4)
EDGE 400	Sector_Set3dFloor	(tag, 1, 0, 255)
EDGE 401	Sector_Set3dFloor	(tag, 1, 16, 255)
EDGE 402	Sector_Set3dFloor	(tag, 1, 32, 255)
EDGE 403	Sector_Set3dFloor	(tag, 2, 2, 255)
EDGE 404	Sector_Set3dFloor	(tag, 2, 2, 204)
EDGE 405	Sector_Set3dFloor	(tag, 2, 2, 153)
EDGE 406	Sector_Set3dFloor	(tag, 2, 2, 102)
EDGE 407	Sector_Set3dFloor	(tag, 2, 2, 51)
EDGE 408	Sector_Set3dFloor	(tag, 2, 2)
EDGE 413	Sector_Set3dFloor	(tag, 1, 8, 255)
EDGE 414	Sector_Set3dFloor	(tag, 1, 8, 204)
EDGE 415	Sector_Set3dFloor	(tag, 1, 8, 153)
EDGE 416	Sector_Set3dFloor	(tag, 1, 8, 102)
EDGE 417	Sector_Set3dFloor	(tag, 1, 8, 51)

Sector_SetCeilingGlow

278:Sector_SetCeilingGlow (tag, height, r, g, b)

tag:
height:
r:
g:
b:

Sector_SetCeilingPanning

186:Sector_SetCeilingPanning (tag, u-int, u-frac, v-int, v-frac)

tag: Tag of affected sector

u-int: Integral part of the horizontal offset

u-frac: Fractional part of the horizontal offset

v-int: Integral part of the vertical offset

v-frac: Fractional part of the vertical offset

Pans the ceiling texture in the specified sector horizontally (x) and/or vertically (y). The formula for selecting a panning value for the two parts is the same as for Sector_SetGravity. Example: Horizontal panning is calculated as $u\text{-int} + (u\text{-frac} * 0.01)$. For most purposes, you can probably leave the ?-frac parts as 0 and just set the ?-int parts.

Sector_SetCeilingScale

188:Sector_SetCeilingScale (tag, u-int, u-frac, v-int, v-frac)

170:Sector_SetCeilingScale2 (tag, u-fixed, v-fixed)

tag: Tag of affected sector

u-int: Integral part of the horizontal scalar

u-frac: Fractional part of the horizontal scalar

v-int: Integral part of the vertical scalar

v-frac: Fractional part of the vertical scalar

u-fixed: horizontal scaling factor as fixed point value

v-fixed: vertical scaling factor as fixed point value

Scales the ceiling texture in the specified sector horizontally (x) and/or vertically (y). The formula for calculating a scaling value with the specials is the same as Sector_SetCeilingPanning and Sector_SetFloorPanning.

Sector_SetCeilingScale2 allows to specify the scaling factor directly as fixed point value. As a result, it cannot be used on a line; only in a script.

UDMF

If you are using UDMF and do not intend to modify the ceiling's scale later on, you can directly set this in fixed point under the sector's custom tab.

Sector_SetColor

212:Sector_SetColor (tag, r, g, b, desat)

tag: Tag of affected sector

r: Amount of red in the light

g: Amount of green in the light

b: Amount of blue in the light

desat: Desaturation of the color

Sets the color of light in a sector. By default, sectors have white light (red, green, blue are all 255). Because there can be a noticeable delay while the game constructs the tables for a color for the first time, another sector should already have the desired color – preferably by using an “open” type script to set the color of a dummy sector that the player will never see.

You can use the testcolor console command to test a color from within the game.

The desat parameter specifies how much the colors in the sector are desaturated. 0 means no desaturation at all, resulting in normal colors. 255 means full desaturation, resulting in a black and white display.

UDMF

If your map is in the UDMF format, and you don't intend to modify sector colors later on, you can apply colors without the use of this special under the sector's custom tab. The color must be provided in hex, i.e. RRGGBB.

Additionally, desaturation can be applied in UDMF as well. The values for desaturation are in fixed point, meaning 0 is normal, and 1 is fully desaturated.

Examples

The following script applies red coloring to any sector with a tag of 3 on map start.

Script 1 Open

```
{
  Sector_SetColor(const:3, 255, 0, 0);
}
```

Sector_SetContents

Warning: This feature does not work in ZDoom but in its OpenGL children ports.

Warning: The feature described on this page is only there for compatibility purposes with Vavoom and will not be extended or maintained beyond this point. It is therefore recommended to look for equivalent ZDoom method to achieve the same effects instead of using this feature.

161:Sector_SetContents (type, translucency, flags)

This function is left undocumented at Graf Zahl's request.

Sector_SetCurrent

220:Sector_SetCurrent (tag, amount, angle, useline)

tag: Tag of affected sector
amount: Strength of the current
angle: Angle the current flows
useline: Whether to get the strength and angle from the line instead
If useline is 1, this behaves just like BOOM's 225 linedef. If useline is 0, then the amount and angle parameters are used instead of the size and orientation of the line.

Conversions from linedef types
The following Doom map format types can be converted as Sector_SetCurrent:

Type	Conversion	Trigger
Boom 225:Set Current	Sector_SetCurrent (tag, 0, 0, 1)	
Eternity 294:TransferHereticCurrent	Sector_SetCurrent (tag, 0, 0, 1)	

Sector_SetDamage

214:Sector_SetDamage (tag, amount, mod, interval,leaky)

tag: Tag of affected sector

amount: Amount of damage to apply

mod: Means-of-death identifier; see the "means of death" column on the damage types for a full list.

0 = MOD_UNKNOWN

12 = MOD_WATER

13 = MOD_SLIME

14 = MOD_LAVA

15 = MOD_CRUSH

16 = MOD_TELEFRAG

17 = MOD_FALLING

18 = MOD_SUICIDE

19 = MOD_BARREL

20 = MOD_EXIT

21 = MOD_SPLASH

22 = MOD_HIT

23 = MOD_RAILGUN

24 = MOD_ICE (ala Hexen)

25 = MOD_DISINTEGRATE (ala Strife's Mauler)

26 = MOD_POISON

27 = MOD_ELECTRIC

1000 = Massacre (no constant)

interval: Time between two inflictions of damage in tics.

leaky: Probability of a radiation suit 'leaking' damage, 0 meaning it doesn't leak. 256 means the radiation suit has no effect at all. Doom's 20% damage sector type uses a value of 5.

Sets the amount of damage done to a player in a sector. Damage settings from regular Doom sector types or UDMF map settings will be overridden by this.

If 'interval' is set to 0, the defaults of older versions will be used: Damage amount below 20 will never hurt the player if he has an environment suit. Damages between 20-49 will occasionally hurt the player even with an environment suit on. Damages of 50 and above will always hurt the player unless he is invulnerable.

There are some newer built-in damage types for which no MOD code exists and custom damage types are identified by names; neither can be used with this special. If you want to attach this special outside a script (e.g., to a line or a thing) but need to use a damage type that is recognized by the game by name, but is not listed above, set the line/thing special to run a script instead and use the SectorDamage or SetSectorDamage ACS function in the script.

Sector_SetFade

213:Sector_SetFade (tag, r, g, b)

tag: Tag of affected sector

r: Amount of red to fade to

g: Amount of green to fade to

b: Amount of blue to fade to

Sets the color that light in the tagged sectors fade to as they get further away from the viewer. This will give the sector a "fog" effect and can greatly enhance the mood of a level. By default, this is whatever the level's fade is specified as being in a MAPINFO lump, or black, if it does not specify a fade. The note about changing a sector's color during the middle of gameplay also applies here. You can use the testfade console command to test a fade from within the game. Keep in mind that Sector_SetFade is affected by a sector's light, so a faded sector with a light value of 255 will have virtually no visible fade color, and a faded sector with a light value of 0 will be nearly a solid color.

UDMF

If you are using UDMF for your map, and do not intend to change the fade color later on, you can set a sector's fade without the need of this special. You can find it under a sector's custom tab, and the values it requires are in hex, i.e. RRGGBB.

Sector_SetFloorGlow

277:Sector_SetFloorGlow (tag, height, r, g, b)

tag:
height:
r:
g:
b:

Sector_SetFloorPanning

187:Sector_SetFloorPanning (tag, u-int, u-frac, v-int, v-frac)

tag: Tag of affected sector

u-int: Integral part of the horizontal offset

u-frac: Fractional part of the horizontal offset

v-int: Integral part of the vertical offset

v-frac: Fractional part of the vertical offset

Pans the floor texture in the specified sector horizontally (x) and/or vertically (y). The formula for selecting a panning value for the two parts is the same as for Sector_SetGravity. Example: Horizontal panning is calculated as $u\text{-int} + (u\text{-frac} * 0.01)$. For most purposes, you can probably leave the ?-frac parts as 0 and just set the ?-int parts.

UDMF

If you are using UDMF and do not intend to modify the floor's panning later on, you can directly set this in fixed point under the sector's custom tab.

Sector_SetFloorScale

189:Sector_SetFloorScale (tag, u-int, u-frac, v-int, v-frac)

171:Sector_SetFloorScale2 (tag, u-fixed, v-fixed)

tag: Tag of affected sector

u-int: Integral part of the horizontal scalar

u-frac: Fractional part of the horizontal scalar

v-int: Integral part of the vertical scalar

v-frac: Fractional part of the vertical scalar

u-fixed: horizontal scaling factor as fixed point value

v-fixed: vertical scaling factor as fixed point value

Scales the floor texture in the specified sector horizontally (x) and/or vertically (y). The formula for calculating a scaling value with the specials is the same as Sector_SetCeilingPanning and Sector_SetFloorPanning.

Sector_SetFloorScale2 allows to specify the scaling factor directly as fixed point value. As a result, it cannot be used on a line; only in a script.

UDMF

If you are using UDMF and do not intend to modify the floor's scale later on, you can directly set this in fixed point under the sector's custom tab.

Sector_SetFriction

219:Sector_SetFriction (tag, amount)

tag: Tag of affected sector

amount: Amount of friction

Sets a sector's friction. If amount is 0, then the amount of friction is determined by the line's length (just like BOOM's 223 linedef).

If Sector_SetFriction is used on a linedef, it is initialized during initialization of the map.

You can use it in a script to alter the friction in a sector, valid values are from 1 to 255:

A value of 100 results in normal friction.

Decreasing the value from 100 results in increased friction (~ sludgy).

Increasing the value from 100 results in decreased friction (~ icy).

Conversions from linedef types

The following Doom map format types can be converted as Sector_SetFriction:

Type	Conversion	Trigger
------	------------	---------

Boom 223:Set Friction to Length	Sector_SetFriction (tag, 0)	
---------------------------------	-----------------------------	--

Sector_SetGravity

216:Sector_SetGravity (tag, ipart, fpart)

tag: Tag of affected sector

ipart: Integral part of the gravity multiplier

fpart: Fractional part of the gravity multiplier

Sets the amount of gravity in a sector. The actual formula used is $sv_gravity * (ipart + fpart * 0.01)$. `sv_gravity` is a cvar that defines "normal" gravity. So `Sector_SetGravity(1, 0, 50)` would set a sector to half normal gravity, `Sector_SetGravity(1, 2, 0)` would be double normal gravity, `Sector_SetGravity(1, 1, 0)` would be normal gravity, etc.

Sector_SetLink

51:Sector_SetLink (controtag, tag, surface, movetype)

controtag: Tag of the control sector.

tag: Tag of the sector(s) to link to the control sector.

surface: Whether to link to the control sector's ceiling (true) or floor (false).

movetype: Type of movement that will be linked. See below.

Creates a link between two or more sectors so that they will move together. This allows notably for the creation of complex lifts comprised of many sectors that will hold together if their movement gets blocked by the player or another actor.

If the control tag is 0, this special will be executed when the map is loaded with the line's front sector as the control sector. If the control tag is not 0, it will be executed at run time so it can be used in scripts or with switches. The control sector will be the first one with the matching tag. A sector can be linked to itself, allowing to link a ceiling's movement to its floor's, or vice-versa. To do that, the sector must have a tag, referenced in the tag argument; however controtag must be zero and so the special must be placed on one of the sector's front-facing lines.

type can be a combination of the following flags:

1: link the tagged sector's floor to the specified surface of the control sector

2: link the tagged sector's ceiling to the specified surface of the control sector

4: the floor movement is the opposite direction as the control sector's surface (bit 1 is required)

8: the ceiling movement is the opposite direction as the control sector's surface (bit 2 is required)

If type is 0, the sectors will be unlinked from the control sector.

Sector_SetPlaneReflection

Warning: This feature does not work in ZDoom but in its OpenGL children ports.

159:Sector_SetPlaneReflection (tag, floor, ceiling)

tag: Tag of affected sectors

floor: Amount of reflectiveness of the floor. 0 means none and 255 means fully reflective.

ceiling: Amount of reflectiveness of the ceiling. 0 means none and 255 means fully reflective.

Makes the tagged sector floors and/or ceilings reflective like mirrors. If used improperly this can reduce the frame rate quite drastically. You should keep the amount of reflective planes as small as possible.

Sector_SetPortal

57:Sector_SetPortal (tag, type, plane, misc, alpha)

Establishes a one-way portal between two sectors.

Parameters

tag: Tag of sectors in which the portal must be seen. Note that the sectors seen through the portal must not share this tag. For the 'copy to line' type this will be a line id, not a sector tag.

type: The type of portal.

plane: Whether the portal is set on the floor (0), the ceiling (1), or both planes (2). For the copy portal type, setting this to 3 will copy any portal with the given tag.

misc: The meaning of this argument depends on the type.

For a normal view portal (type 0): Specifies whether the line belongs to the sector viewed through the portal (1) or the sector in which the portal is seen (0).

For a transferred portal (type 1): Specifies the sector tag of the portal to copy.

For a skybox portal (type 2): Ignored.

alpha: Gives a translucency value to the portal plane.

Types

0: Normal view portal. This is what will generally be used. Two linedefs must be defined, one in the "source" sector and one in the "destination" sector; the first three arguments for both of these linedefs must be identical while the fourth must be different. Any number of sectors can share this link by having the same tag. To make a two-way portal, another pair of linedef must be used.

1: Copied portal. 'misc' specifies the tag of the portal to copy. This copies the given portal to all sectors tagged with 'tag' or the line's front sector if 'tag' is 0. Use this for sectors which need to have a different tag, such as a lift, yet should share the portal.

2: Skybox portal. The linedef's front sector is the skybox and must contain a SkyCamCompat object. The sky from this skybox will be visible on the concerned plane of all tagged sectors; even if the sky flat is not used.

Graficon.gif Warning: This feature is GZDoom specific, and is not compatible with ZDoom!

To see all of GZDoom's specific features, see GZDoom features.

3: Plane portal. Renders the linedef's frontsector's planes into infinity at a fixed distance from the camera.

4: Horizon portal. Renders the linedef's frontsector's planes into infinity at the planes' heights.

5: Copy portal to line: 'misc' specifies the portal to copy. If 'tag' is non-zero, copy the portal to all lines with the given ID, otherwise affects the line itself.

6: Interactive portal. Actors can move through it. Although implemented in ZDoom, it is of more use with GZDoom's renderer which is far less limited.

Conversions from linedef types

The following Doom map format types can be converted as Sector_SetPortal:

Type	Conversion	Trigger
Eternity 283:Portal_PlaneCeiling	Sector_SetPortal (tag, 3, 1, 0, 0)	
Eternity 284:Portal_PlaneFloor	Sector_SetPortal (tag, 3, 0, 0, 0)	
Eternity 285:Portal_PlaneFloorCeiling	Sector_SetPortal (tag, 3, 2, 0, 0)	
Eternity 286:Portal_HorizonCeiling	Sector_SetPortal (tag, 4, 1, 0, 0)	
Eternity 287:Portal_HorizonFloor	Sector_SetPortal (tag, 4, 0, 0, 0)	
Eternity 288:Portal_HorizonFloorCeiling	Sector_SetPortal (tag, 4, 2, 0, 0)	
Eternity 289:Portal_LineTransfer	Sector_SetPortal (0, 5, 0, tag)	
Eternity 290:Portal_SkyboxCeiling	Sector_SetPortal (tag, 2, 1, 1, 0)	
Eternity 291:Portal_SkyboxFloor	Sector_SetPortal (tag, 2, 0, 1, 0)	
Eternity 292:Portal_SkyboxFloorCeiling	Sector_SetPortal (tag, 2, 2, 1, 0)	
Eternity 295:Portal_AnchoredCeiling	Sector_SetPortal (tag, 0, 1, 1, 0)	

Eternity 296:Portal_AnchoredFloor Sector_SetPortal (tag, 0, 0, 1, 0)
Eternity 297:Portal_AnchoredFloorCeiling Sector_SetPortal (tag, 0, 2, 1, 0)
Eternity 298:Portal_AnchorLine Sector_SetPortal (tag, 0, 1, 0, 0)
Eternity 299:Portal_AnchorLineFloor Sector_SetPortal (tag, 0, 0, 0, 0)
Eternity 344:Portal_AnchoredCeiling Sector_SetPortal (tag, 0, 1, 1, 0)
Eternity 345:Portal_AnchoredFloor Sector_SetPortal (tag, 0, 0, 1, 0)
Eternity 346:Portal_AnchorLine Sector_SetPortal (tag, 0, 1, 0, 0)
Eternity 347:Portal_AnchorLineFloor Sector_SetPortal (tag, 0, 0, 0, 0)
Eternity 358:Portal_AnchoredCeiling Sector_SetPortal (tag, 6, 1, 1, 0)
Eternity 359:Portal_AnchoredFloor Sector_SetPortal (tag, 6, 0, 1, 0)
Eternity 360:Portal_AnchorLine Sector_SetPortal (tag, 6, 1, 0, 0)
Eternity 361:Portal_AnchorLineFloor Sector_SetPortal (tag, 6, 0, 0, 0)
Eternity 385:Apply_PortalToFrontsector Sector_SetPortal (0, 1, 3, tag)

Sector_SetRotation

185:Sector_SetRotation (tag, floor-angle, ceiling-angle)

tag: Tag of affected sector

floor: angle Angle to rotate floor

ceiling: angle Angle to rotate ceiling

Sets the angles that the floor and ceiling textures in the specified sector are rendered at. 0 is normal. The angles are specified in degrees (0-359). In UDMF, unless you intend to modify the rotation later on, this can be set without the need for a special. The parameter is located under the custom tab for sectors.

Sector_SetTranslucent

98:Sector_SetTranslucent (tag, plane, amount, type)

tag: Tag of affected sector

plane: Chooses the floor (0) or the ceiling (1)

amount: Amount of alpha transparency, 0 is fully transparent and 255 is fully opaque

type: Chooses the type of transparency, 0 is normal, 1 is additive (Planned feature)

Usage

Changes the translucency value of a given portal plane in all affected sectors.

Note that additive translucency is not yet implemented.

Sector_SetWind

218:Sector_SetWind (tag, amount, angle, useline)

tag: Tag of affected sector

amount: Strength of the wind

angle: Angle the wind blows

useline: Get strength and angle from the line instead?

If useline is 1, this behaves just like BOOM's 224 linedef. If useline is 0, then the magnitude and angle parameters are used instead of the size and orientation of the line.

If used in a script, keep in mind that the sector needs the "pusher/puller/wind enabled" flag (4096) for this to have an effect.

SendToCommunicator

174:SendToCommunicator (voc_id, front_only, identify, nolog)

voc_id: VOC lump in which to play and the respective LOG lump number.

front_only: If non-zero, only activate on front side of the line.

identify: The incoming message to print. If this is 0, "Incoming Message" is printed. If it is 1, "Incoming Message from BlackBird" is printed.

Custom messages are possible by way of defining them in a LANGUAGE lump. The identifiers of those messages are required to be in the form: TXT_COMM#, where # is a value which corresponds to that set for identify. For instance, setting identify to 7, prints the message identified as TXT_COMM7.

nolog: If non-zero, the message will not be placed in the objectives popup.

This special will only activate if the player has the communicator in their inventory. It will play the VOC lump with the corresponding number and, assuming nolog is 0, print the LOG lump with the same number to the objectives popup screen.

Conversions from linedef types

The following Doom map format types can be converted as SendToCommunicator:

Type	Conversion	Trigger
Strife 201:W1 Message #Tag (Front Side Only)	SendToCommunicator (tag, 1, 0)	Player Cross
Strife 202:W1 Message #Tag	SendToCommunicator (tag, 0, 0)	Player Cross
Strife 210:W1 Message #Tag If Flamethrower (No Log)	SendToCommunicator (tag, 0, 1)	Player Cross
Strife 211:SR Message #Tag (No Log)	SendToCommunicator (tag, 0, 2, 1)	Player Use, Repeatable

sePuzzleItem

Jump to navigationJump to search

129:UsePuzzleItem (item, script, s_arg1, s_arg2, s_arg3)

item: Puzzle artifact number

script: Script to execute

s_arg1: First argument to pass to script

s_arg2: Second argument to pass to script

s_arg3: Third argument to pass to script

Executes a specified script on the current map if the player holds the specified puzzle artifact, and removes the artifact in the process. If the player does not have the artifact, this plays the "puzzfail" player sound instead. This special must be placed on a thing or a line and cannot be used in a script.

Standard puzzle artifacts (class names):

0: Skull

1: GemBig

2: GemRed

3: GemGreen1

4: GemGreen2

5: GemBlue1

6: GemBlue2

7: Book1

8: Book2

9: FlameMask

10: FWeapon

11: CWeapon

12: MWeapon

13: Gear1

14: Gear2

15: Gear3

16: Gear4

This special can also be put on any thing in the map. If that is the case and the player uses the item near this thing the special is also executed.

SetGlobalFogParameter

Warning: This feature is GZDoom specific, and is not compatible with ZDoom!
To see all of GZDoom's specific features, see GZDoom features.

157:SetGlobalFogParameter (property, value)

Changes the fog properties specified in MAPINFO.

Parameters

property: One of the following constant identifiers must be used (numbers are provided for use on a linedef or map object):

0â€”FOGP_DENSITY corresponds to the fogdensity property.

1â€”FOGP_OUTSIDEDENSITY corresponds to the outsidefogdensity property.

2â€”FOGP_SKYFOG corresponds to the skyfog property.

value: The new value of the chosen property.

SetPlayerProperty

191:SetPlayerProperty (who, set, which)

who: Set to 1 to affect all players, 0 for just the one who activated the special.

set: Set to 1 to turn on the property, or 0 to turn it off. PROP_INVULNERABILITY has a special use for this field.

which: Which property to change

Sets a property for a player.

0 — PROP_FROZEN — The player cannot move, but can still fire and perform other actions. (In order to instantly freeze the player in place, use in conjunction with Thing_Stop to set the player's current momentum to zero.)

1 — PROP_NOTARGET — Monsters ignore the player, unless they have already seen him or the player harms them.

2 — PROP_INSTANTWEAPONSWITCH — The weapons do not delay between changing them.

3 — PROP_FLY — The player is (not) subjected to gravity. The behavior is about the same as having Heretic's/Hexen's Wings of Wrath.

4 — PROP_TOTALLYFROZEN — Same as PROP_FROZEN, but does not allow the player to look around or shoot. (In fact, the only control they can still use is the "+use" key)

5 — PROP_INVULNERABILITY (deprecated) — Invulnerability sphere. Using a "set" value of 1 makes the player(s) invulnerable and applies the inverted greyscale invulnerable palette. Using a "set" value of 2 makes the player(s) invulnerable but does not apply the invulnerability palette. SetActorProperty with APROP_Invulnerable or APROP_DamageFactor is a valid substitution.

6 — PROP_STRENGTH (deprecated) — Berserk pack

7 — PROP_INVISIBILITY (deprecated) — Partial Invisibility sphere

8 — PROP_RADIATIONSUIT (deprecated) — Radiation suit

9 — PROP_ALLMAP (deprecated) — Computer area map

10 — PROP_INFRARED (deprecated) — Light Amplification Goggles

11 — PROP_WEAPONLEVEL2 (deprecated) — Tome of Power

12 — PROP_FLIGHT (deprecated) — Wings of Wrath

13 — PROP_UNUSED1 — Does nothing. Do not use.

14 — PROP_UNUSED2 — Does nothing. Do not use.

15 — PROP_SPEED (deprecated) — Speed boots

16 — PROP_BUDDHA — Buddha Mode (1 indestructible hit point)

17 — PROP_BUDDHA2 — Ultimate buddha mode, the player will be kept at 1 health even against hazards that normally kill them with normal buddha, such as instant kill floors. (development version 458142e only)

18 — PROP_FRIGHTENING — The player scares away any monsters targeting them which do not have NOFEAR enabled. (development version 458142e only)

19 — PROP_NOCLIP — The player can walk right through any blocking actors and level geometry. (development version 458142e only)

20 — PROP_NOCLIP2 — Same as above, but the player can also fly around, similar to Quakes' noclip. Allowing them to also noclip straight through 3D floors. (development version 458142e only)

21 — PROP_GODMODE — Degreelessness mode. The player is invulnerable to most damage, barring some hazards such as telefragging and instant kill sectors. (development version 458142e only)

22 — PROP_GODMODE2 — Ultimate degreeless mode. The player is immune to any and all damage. (development version 458142e only)

Examples

This function is almost like an ACS pause game function, and can be toggled on and off.

Function void PlayerFreeze (bool IsOn)

```
{
    if(IsOn)
    {
        Thing_Stop(TID_Player);
    }
}
```

```
SetPlayerProperty(TRUE, ON, PROP_TOTALLYFROZEN);  
if(CheckActorInventory(TID_Player, "PowerTimeFreezer") == 0)  
    GiveActorInventory(TID_Player, "PowerGiver_ACSTimeFreeze", 1);
```

```
}  
else  
{
```

```
SetPlayerProperty(TRUE, OFF, PROP_TOTALLYFROZEN);  
TakeActorInventory(TID_Player, "PowerTimeFreezer", 0x7ffffff);
```

```
}
```

```
}
```

Stairs_BuildDown

26:Stairs_BuildDown (tag, speed, height, delay, reset)

tag: Tag of first sector in staircase

speed: How quickly the steps lower

height: Height of each step

delay: Number of tics the staircase waits between steps

reset: Tics until the stairs return to their original heights (0 if never)

Builds a staircase in sectors marked with the Stairs_Specials.

Stairs_BuildDownDoom

270:Stairs_BuildDownDoom (tag, speed, height, delay, reset)

tag:
speed:
height:
delay:
reset:

Stairs_BuildDownDoomSync

272:Stairs_BuildDownDoomSync (tag, speed, height, reset)

tag:
speed:
height:
reset:

Stairs_BuildDownSync

31:Stairs_BuildDownSync (tag, speed, height, reset)

tag: Tag of first sector in staircase

speed: How quickly the first step lowers

height: Height of each step

reset: Tics until the stairs return to their original heights (0 if never)

Builds a staircase in sectors marked with the Stairs_Specials. Each step moves at a different speed so that they all reach their destination heights at the same time.

Stairs_BuildUp

27:Stairs_BuildUp (tag, speed, height, delay, reset)

tag: Tag of first sector in staircase

speed: How quickly the steps rise

height: Height of each step

delay: Number of tics the staircase waits between steps

reset: Tics until the stairs return to their original heights (0 if never)

Builds a staircase in sectors marked with the Stairs_Specials.

Stairs_BuildUpDoom

217:Stairs_BuildUpDoom (tag, speed, height, delay, reset)

tag: Tag of first sector in staircase

speed: How quickly the steps rise

height: Height of each step

delay: Number of tics the staircase waits between steps

reset: Tics until the stairs return to their original heights (0 if never)

Builds a staircase. The sectors that make up the staircase are determined using the normal Doom mechanism.

Conversions from linedef types

The following Doom map format types can be converted as :

Type	Conversion	Trigger
Doom 7:S1 Build Stairs 8	Stairs_BuildUpDoom (tag, 2, 8)	Player Use
Heretic 7:S1 Build Stairs 8	Stairs_BuildUpDoom (tag, 8, 8)	Player Use
Doom 8:W1 Build Stairs 8	Stairs_BuildUpDoom (tag, 2, 8)	Player Cross
Heretic 8:W1 Build Stairs 8	Stairs_BuildUpDoom (tag, 8, 8)	Player Cross
Strife 100:W1 Build Stairs 16 Crush	Stairs_BuildUpDoom (tag, 32, 16, 0, 0)	Player Cross
Heretic 106:W1 Build Stairs 16	Stairs_BuildUpDoom (tag, 8, 16)	Player Cross
Heretic 107:S1 Build Stairs 16	Stairs_BuildUpDoom (tag, 8, 16)	Player Use
Strife 127:S1 Build Stairs 16 Crush	Stairs_BuildUpDoom (tag, 32, 16, 0, 0)	Player Use
Boom 256:WR Build Stairs 8	Stairs_BuildUpDoom (tag, 2, 8, 0, 0)	Player Cross, Repeatable
Boom 258:SR Build Stairs 8	Stairs_BuildUpDoom (tag, 2, 8, 0, 0)	Player Use, Repeatable

Stairs_BuildUpDoomCrush

273:Stairs_BuildUpDoomCrush (tag, speed, height, delay, reset)

tag:
speed:
height:
delay:
reset:

Conversions from linedef types

The following Doom map format types can be converted as Stairs_BuildUpDoomCrush:

Type	Conversion	Trigger
Doom 100:W1 Build Stairs 16 + Crush	Stairs_BuildUpDoomCrush (tag, 32, 16, 0, 0)	Player Cross
Doom 127:S1 Build Stairs 16 + Crush	Stairs_BuildUpDoomCrush (tag, 32, 16, 0, 0)	Player Use
Boom 257:WR Build Stairs 16 + Crush	Stairs_BuildUpDoomCrush (tag, 32, 16, 0, 0)	Player Cross, Repeatable
Boom 259:SR Build Stairs 16 + Crush	Stairs_BuildUpDoomCrush (tag, 32, 16, 0, 0)	Player Use, Repeatable

Stairs_BuildUpDoomSync

271:Stairs_BuildUpDoomSync (tag, speed, height, reset)

tag:
speed:
height:
reset:

Stairs_BuildUpSync

32:Stairs_BuildUpSync (tag, speed, height, reset)

tag: Tag of first sector in staircase

speed: How quickly the first step raises

height: Height of each step

reset: Tics until the stairs return to their original heights (0 if never)

Builds a staircase in sectors marked with the Stairs_Specials. Each step moves at a different speed so that they all reach their destination heights at the same time.

StartConversation

18:StartConversation (talker_tid, facetalker)

Starts a Strife dialog with the specified actor. If you set facetalker to 1 the player will face the talking actor. If it is set to 0 both the player's and the actor's angle will remain unchanged.

Static_Init

190:Static_Init (tag, prop, flip/ceiling, movetype)

tag: Tag of affected sector, or lineid of concerned linedefs.

prop: Sector property to set.

flip/ceiling: If prop is set to 255, this is a boolean for whether to flip the sky texture. If prop is set to 3, this is a boolean for whether the control surface is the ceiling or not (and it not, obviously, it's the floor).

movetype: If prop is set to 3, determines which type of movement to use, using the same values as in Sector_SetLink.

Sets a sector property during level initialization. Static_Init was designed to copy the functionality of several Boom, MBF and Eternity Engine linetypes; the specific action depends on the second argument.

Property

Value	Function	Notes
-------	----------	-------

0	Gravity	Sets the gravity in tagged sectors to the length of the linedef (333 in Doom-format maps).
---	---------	--

1	Color	Sets the light or fog color in a sector. An RRGGBB hex format color used as an upper texture name will set the light color; a lower texture will set the fog color.
---	-------	---

2	Damage	Sets damage in tagged sectors to the length of the linedef (335 in Doom-format maps).
---	--------	---

3	Link	Defines a sector link with line IDs instead of tags, as in Eternity.
---	------	--

253	Sector	Define special handling for adding Eternity Engine ExtraData to sector and line. Used by XLAT for Eternity levels.
-----	--------	--

254	Line	
-----	------	--

255	Sky	Uses the line's upper texture as the sky in any tagged sectors (that is, the sky flat will be replaced with that texture instead of the sky indicated in MAPINFO). If flip is set to 1, the texture will be flipped, as is normal behavior for Doom skies. Offsets and scrolling of the texture are transferred to the sky as well. Scrolling, however, is only at a tiny fraction of the wall texture's scrolling speed. This corresponds to the MBF sky transfer linetypes (271 and 272).
-----	-----	---

Note: The line's lower texture, if set and of the same dimensions as the upper texture, will be used during lightning flashes.

Conversions from linedef types

The following Doom map format types can be converted as Static_Init:

Type	Conversion	Trigger
------	------------	---------

Eternity 270:ExtraDataLine	Static_Init (tag, 254)	
----------------------------	------------------------	--

MBF 271:Transfer Sky Texture	Static_Init (tag, 255, 0)	
------------------------------	---------------------------	--

MBF LogoIcon.png 272:Transfer Sky Texture (Flipped)	Static_Init (tag, 255, 1)	
---	---------------------------	--

Legacy LogoIcon.png 282:Set Color (Upper Light, Lower Fog)	Static_Init (tag, 1)	
--	----------------------	--

ZDoom LogoIcon.png 333	Static_Init (tag, 0)	
------------------------	----------------------	--

ZDoom LogoIcon.png 334	Static_Init (tag, 1)	
------------------------	----------------------	--

ZDoom LogoIcon.png 335	Static_Init (tag, 2)	
------------------------	----------------------	--

Eternity 379:Attach_SetCeilingControl	Static_Init (tag, 3, 1)	
---------------------------------------	-------------------------	--

Eternity 380:Attach_SetFloorControl	Static_Init (tag, 3, 0)	
-------------------------------------	-------------------------	--

Eternity 381:Attach_FloorToControl	Static_Init (0, 3, 0, 1)	
------------------------------------	--------------------------	--

Eternity 382:Attach_CeilingToControl	Static_Init (0, 3, 1, 2)	
--------------------------------------	--------------------------	--

Eternity 383:Attach_MirrorFloorToControl	Static_Init (0, 3, 0, 5)	
--	--------------------------	--

Eternity 384:Attach_MirrorCeilingToControl	Static_Init (0, 3, 0, 10)	
--	---------------------------	--

Eternity 401:ExtraDataSector	Static_Init (tag, 253)	
------------------------------	------------------------	--

Team_GivePoints

Warning: This feature is Skulltag specific, and is not compatible with ZDoom!
To see all of Skulltag's specific features, see Skulltag features.

153:Team_GivePoints (team, points, announce)

team: What team to give points to

points: How many points to give to the team

announce: Non-zero will have the play the announcer sound for "Team Scores"

Gives the specified team points. The following team values can be used:

TEAM_BLUE (0) – Sets the player to the blue team.

TEAM_RED (1) – Sets the player to the red team.

If announce is true the announcer will announce what team has scored and a message is displayed to all players.

Team_Score

SkulltagIcon.png Warning: This feature is Skulltag specific, and is not compatible with ZDoom!
To see all of Skulltag's specific features, see Skulltag features.

152:Team_Score (points, nogrin)

points: How many points to give to the team

nogrin: Assumed to be if non-zero, the player's mug does not grin

Gives the player who activated the special points for his/her team. According to the source nogrin is currently not used

Teleport

70:Teleport (tid, tag, nosourcefog)

tid: Thing ID of the destination spot
tag: Tag of the destination sector
nosourcefog: If non-zero, the teleport only spawns a teleport fog object at the destination but not the source of the teleportation
Teleports the activating thing to a new location. If tag is 0, it will use a random teleport destination out of those with the matching tid. This can be restricted to certain sectors if tag is non-0. If tid is 0 and tag is non-0, it will use the first teleport destination found in the first sector with the matching tag.

Conversions from linedef types
The following Doom map format types can be converted as Teleport:

Type	Conversion	Trigger
Doom 39:W1 Teleport	Teleport(0, tag)	Player Cross, Monsters Activate
Strife 39:W1 Teleport	Teleport(0, tag)	Player Cross, Monsters Activate
Doom 97:WR Teleport	Teleport(0, tag)	Player Cross, Repeatable, Monsters Activate
Strife 97:WR Teleport	Teleport(0, tag)	Player Cross, Repeatable, Monsters Activate
Doom 125:W1 Teleport	Monsters Only Teleport(0, tag)	Monster Cross
Strife 125:W1 Teleport	Monsters Only Teleport(0, tag)	Monster Cross
Doom 126:WR Teleport	Monsters Only Teleport(0, tag)	Monster Cross, Repeatable
Strife 126:WR Teleport	Monsters Only Teleport(0, tag)	Monster Cross, Repeatable
Boom 174:S1 Teleport	Teleport(0, tag)	Player Use, Monsters Activate
Boom 195:SR Teleport	Teleport(0, tag)	Player Use, Repeatable, Monsters Activate
Strife 231:WR Teleport (Silent Source)	Teleport(0, tag, 1)	Player Cross, Repeatable, Monsters Activate

Alias
In ZScript, this special is alternatively called TeleportSpecial. This alias was introduced to resolve a name conflict between the special and an actor function.

Teleport_EndGame

75:Teleport_EndGame ()

Ends the game.

Conversions from linedef types

The following Doom map format types can be converted as Teleport_EndGame:

Type	Conversion	Trigger
Strife 51:S1 Exit (Endgame)	Teleport_EndGame ()	Player Use
Strife 124:W1 Exit (Endgame)	Teleport_EndGame ()	Player Cross

Teleport_Line

215:Teleport_Line (thisid, destid, flip)

Error.gif
Warning: This special is not fully supported by the UDMF map format. Certain parameters, noted below, have been made obsolete by the ability to directly specify properties within the linedef structure. These parameters should be set to 0 when using the UDMF format.

thisid: Line ID of this line. Obsolete in UDMF.
destid: Line ID of destination line
flip: Set this to 1 to flip the teleported thing 180 degrees
Silently teleports a thing between two lines. A pair of lines with this special can share the same id (thisid and destid are the same) because a line will never teleport to itself. This special also behaves like Line_SetIdentification by setting the line ID of the any lines it is used on. Note: Cannot be used directly in ACS scripts but can be set to lines with SetLineSpecial.

Conversions from linedef types
The following Doom map format types can be converted as Teleport_Line:

Type	Conversion	Trigger
Boom 243:W1 Teleport Line	Teleport_Line (tag, tag, 0)	Player Cross, Monsters Activate
Boom 244:WR Teleport Line	Teleport_Line (tag, tag, 0)	Player Cross, Repeatable, Monsters Activate
Boom 262:W1 Teleport Line (Reversed)	Teleport_Line (tag, tag, 1)	Player Cross, Monsters Activate
Boom 263:WR Teleport Line (Reversed)	Teleport_Line (tag, tag, 1)	Player Cross, Repeatable, Monsters Activate
Boom 264:W1 Teleport Line Monsters Only (Reversed)	Teleport_Line (tag, tag, 1)	Monster Cross
Boom 265:WR Teleport Line Monsters Only (Reversed)	Teleport_Line (tag, tag, 1)	Monster Cross, Repeatable
Boom266:W1 Teleport Line Monsters Only	Teleport_Line (tag, tag, 0)	Monster Cross
Boom 267:WR Teleport Line Monsters Only	Teleport_Line (tag, tag, 0)	Monster Cross, Repeatable

Teleport_NewMap

74:Teleport_NewMap (map, pos, face)

map: Levelnum of the map to teleport to

pos: Corresponds to destination player start spot arg0

face: If TRUE (> 0) then the player will retain the direction they were facing from the last map

Teleports the player to a new map and to the player start spot whose arg0 matches pos.

Example:

Teleport_NewMap (2, 0, 1);

In Doom2 wads in Hexen format (Zdoom) the above line means teleport the player to map02 (Underhalls) and the player 1 start (0) keeping the direction they were facing on the previous map.

Teleport_NoFog

71:Teleport_NoFog (tid, useangle, tag, keepheight)

- tid: Thing ID of the destination spot
- useangle: specifies how the angle of the teleport exit is used
- tag: Tag of destination sector
- keepheight: if set, the thing will keep the same height relative to the floor

Teleports the activating thing to a new location, but without fog or a delay.

There are different modes how the teleport exit's angle is used:

- 0: Hexen-compatible: Do not change angle and velocity at all.
 - 1: Strife-compatible: Always use the teleport exit's angle
 - 2: Boom-compatible: Adjust relatively to the teleport exit's angle, but in the wrong direction. This replicates a Boom bug that never got fixed.
 - 3: Boom-fixed: Adjust relatively to the teleport exit's angle.
- If tag is 0, it will use a random teleport destination out of those with the matching tid. This can be restricted to certain sectors if tag is non-zero. If tid is 0 and the tag is non-zero, it will use the first teleport destination found in the first sector with the matching tag.

Conversions from linedef types

The following Doom map format types can be converted as Teleport_NoFog:

Type	Conversion	Trigger
Strife 185:WR Teleport (Silent)	Teleport_NoFog(0, 1, tag)	Player Cross, Repeatable, Monsters Activate
Boom 207:W1 Teleport Preserve Direction (Silent)	Teleport_NoFog(0, 2, tag, 1)	Player Cross, Monsters Activate
Boom 208:WR Teleport Preserve Direction (Silent)	Teleport_NoFog(0, 2, tag, 1)	Player Cross, Repeatable, Monsters Activate
Boom 209:S1 Teleport Preserve Direction (Silent)	Teleport_NoFog(0, 2, tag, 1)	Player Use, Monsters Activate
Boom 210:SR Teleport Preserve Direction (Silent)	Teleport_NoFog(0, 2, tag, 1)	Player Use, Repeatable, Monsters Activate
Boom 268:W1 Teleport Monsters Only (Silent)	Teleport_NoFog(0, 2, tag, 1)	Monster Cross
Boom 269:WR Teleport Monsters Only (Silent)	Teleport_NoFog(0, 2, tag, 1)	Monster Cross, Repeatable

Teleport_NoStop

154:Teleport_NoStop (tid, sectortag, nofog)

tid: Tid of the teleport spot

sectortag: If non-zero, a tagged sector with a non tagged teleport spot will be used instead

nofog: If non-zero, no teleport fog will be spawned for leaving an area only. For teleporting without fog entirely, use Teleport_NoFog.

Teleports an actor to a specified teleport spot (or alternatively a tagged sector) without losing momentum. sectortag is only required for Doom maps that are not in Hexen format.

Teleport_ZombieChanger

39:Teleport_ZombieChanger (tid, tag)

tid: Thing ID of the destination spot

tag: Tag of destination sector

Teleports the activating thing to a new location, but without fog or a delay. After teleporting, the teleported object switches to its pain state. If tag is 0, it will use a random teleport destination out of those with the matching tid. This can be restricted to certain sectors if tag non-zero. If tid is 0 and tag is non-zero, it will use the first teleport destination found in the first sector with the matching tag.

Conversions from linedef types

The following Doom map format types can be converted as Teleport_ZombieChanger:

Type	Conversion	Trigger
Strife 195:WR Teleport (Zombie Changer)	Teleport_ZombieChanger(0, tag)	Player Cross, Repeatable, Monsters Activate

TeleportGroup

77:TeleportGroup (tid, source, destination, movesource, fog)

Usage

Teleports the specified group of actors from one area to another while keeping their same positions relative to each other. Note that velocities are not altered to account for a change in angle.

Parameters

tid: The TID of the actor(s) to teleport. If 0, teleports the activator only.

source: The spot relative to which to teleport. This must be a teleport destination object.

destination: The destination spot relative to which to spawn, which must also be a teleport destination.

movesource: If 1, the source object teleports along with the other actors

fog: Set to 1 to use teleport fog, or 0 to teleport without.

TeleportInSector

78:TeleportInSector (tag, source_tid, dest_tid, fog, group_tid)

tag: Tag of the sector from which will be teleported.

source_tid: The spot relative to which to teleport.

dest_tid: The destination spot relative to which to spawn.

fog: Spawn with or without fog. (0 = no fog, 1 = fog)

group_tid: The TID of the thing(s) to teleport. If 0, teleports all actors in the sector

Teleport a group of actors in a sector. source_tid is used as a reference point so that they end up in the same position relative to dest_tid. group_tid can be used to not teleport all actors in the sector.

Note that the "destination spot" must be a Teleport Destination. Using a Map Spot for the destination will not work. Also, if the special is to be used for teleportation while the actor is in a moving elevator, use the destination with z-height and gravity. This is particularly true for use with elevators that use the Sector_Set3dFloor special for their floors and ceilings, since otherwise the actor risks ending up on the floor of the sector in which the elevator is rising/lowering.

TeleportOther

76:TeleportOther (tid, destination_tid, fog)

tid: TID of the thing to teleport.

destination: TID of the map spot to teleport to.

fog: Teleport with or without fog. (0 = no fog, 1 = fog)

Teleports a thing. The special's main usage is to teleport something else besides the activator.

Example

This script will teleport the player to the coordinates of the thing that is identified by tag 1 similarly to SetActorPosition, possibly telefragging if the thing will try to block.

```
#include "zcommon.acs"
```

```
function void SetActorPositionForced(int tag, int x, int y, int z, bool fog)
```

```
{
    if (tag == 0) tag = ActivatorTID();
    int t = UniqueTID();
    SpawnForced("TeleportDest2", x, y, z, t, 0);
    int t2 = tag;
    if (t2 == 0)
    {
        t2 = UniqueTID();
        Thing_ChangeTID(0, t2);
    }
}
```

```
TeleportOther(t2, t, fog);
```

```
if (t2 != tag)
    Thing_ChangeTID(t2, tag);
Thing_ChangeTID(t, 0);
}
```

```
script 1 ENTER
```

```
{
    SetActorPositionForced(0, GetActorX(1), GetActorY(1), GetActorZ(1), true);
}
```

Thing_Activate

130:Thing_Activate (tid)

tid: Thing ID of the thing to activate. 0 refers to the activator of the script.

Activates a thing so it can be used. In the case of a monster, removes the DORMANT flag from a monster so that it will be able to take damage and attack the player.

For a monster, which looks away from the player upon activation, there are two distinct reactions:

1. A monster, which has not heard gunfire or the player has not crossed the monster's line of sight before activation, will not immediately attack the player but remains as if still dormant. It will attack, however, as soon as it hears gunfire or the player crosses the monster's line of sight.
2. A monster, which has heard gunfire or the player has crossed the monster's line of sight before activation, will attack the player immediately upon activation.

See the example map below.

Examples

This script could be used to start a moving camera with a TID of 15.

```
script 13 ENTER
{
    Thing_Activate(15);
    ChangeCamera(15,0,0);
}
```

Thing_ChangeTID

176:Thing_ChangeTID (oldtid, newtid)

oldtid: The current TID of the thing whose TID will be changed.

newtid: The new TID that the thing will be changed to.

If oldtid is zero, then whatever activated the script will have its TID changed to newtid. If oldtid is non-zero, then everything with the TID oldtid will have their TID changed to newtid.

Examples

One of the primary uses for this special is to give a player a TID using a script like this:

```
script 100 enter
{
    Thing_ChangeTID (0, 1337+PlayerNumber());
}
```

This will give player 1 the TID 1337, player 2 the TID 1338, player 3 the TID 1339, and so on.

Use this script for multiplayer levels:

```
script 256 respawn
{
    Thing_ChangeTID(0, PlayerNumber()+256);
}
```

```
script 257 death
{
    Thing_ChangeTID(0, 0);
}
```

```
script 258 enter
{
    Thing_ChangeTID(0, PlayerNumber()+256);
}
```

This will give player 1 the TID 256, player 2 TID 257, etc. It will cause each player not to have a TID during death.

Use this script for multiplayer levels and players to retain their TID during death:

```
script 256 respawn
{
    Thing_ChangeTID(0, 0);
    Thing_ChangeTID(0, PlayerNumber()+256);
}
```

```
script 258 enter
{
    Thing_ChangeTID(0, PlayerNumber()+256);
}
```

This will give player 1 the TID 256, player 2 TID 257, etc. It will cause each player to keep their TID during death.

For multiplayer, it is very important to change the player's TID to 0 in DEATH or RESPAWN. Otherwise, there will be multiple players (corpses) using the same TID, which will eventually result in a crash.

Thing_Damage

119:Thing_Damage (tid, amount, mod)

Usage

Damages the specified actor.

Parameters

tid: TID of the thing you want to damage.

amount: The amount of damage the thing will receive.

mod: Means of death. Determines the obituary message that will appear if a player is killed. Relevant damage types for the means of death are found on the [Damage_types](#) page.

Similar to [DamageThing](#), but it hurts actors by TID rather than whoever activates it. Note that if TID is 0, this behaves similarly to [DamageThing](#), but using 0 as the amount will not force death.

There are some newer built-in damage types for which no MOD code exists and custom damage types are identified by names; neither can be used with this special. If you want to attach this special outside a script, e.g. to a line or a thing, but need to use a damage type that is recognized by the game by name, but is not listed above, set the line/thing special to run a script instead and use the [Thing_Damage2](#) ACS function in the script.

See [Damage types](#) for a list of means-of-death.

Examples

Script 1 ENTER

```
{  
    Thing_Damage(0, 20, MOD_RAILGUN);  
}
```

Thing_Deactivate

131:Thing_Deactivate (tid)

tid: Thing ID of the thing to deactivate. 0 refers to the activator of the script.

Sets the DORMANT flag of a monster so that it will not be able to take damage or attack the player. Does nothing when used on players.

Thing_Destroy

133:Thing_Destroy (tid, extreme, tag)

tid: Thing ID of the thing to destroy

extreme: Inflicts extreme damage (gib death)

tag: Only affects things in tagged sectors

Kills the specified thing. If you have a TID of 0, then it will kill all the monsters on the map, similar to the "kill monsters" cheat. If extreme is 1, the thing enters its extreme (gib) death sequence if it exists. If tag is not zero, the special will only kill monsters in the tagged sectors.

Note that this function does not properly handle killing players with armor or more than 100% health if extreme is 0.

Thing_Hate

177:Thing_Hate (hater, hatee, type)

hater: the TID of the hater (monster that will hate and attack)

hatee: the TID of the hatee (monster that is to be hated by the hater)

type: the type of hate that this will be:

0 — Just hate one specific actor

1 — Hate actors with given TID and attack players when shot

2 — Same as 1, but will go after enemies without seeing them first

3 — Hunt actors with given TID and also players

4 — Same as 3, but will go after monsters without seeing them first

5 — Hate actors with given TID and ignore player attacks

6 — Same as 5, but will go after enemies without seeing them first

Usage

This forces a thing to hate another thing. If set for monsters, it will cause that monster to go after the hatee rather than the player. Both the hatee and the hater must be alive and have at least 1 health for this to have any effect. Note that if the hater is friendly to the player, it will do nothing.

A TID of 0 refers to the activator of a script. As this is often the player, this means you can use Thing_Hate(tid, 0, 4) to make a monster go after a player without seeing him first.

The various types of hate have slight differences, described in detail here. By “distracted”, it is meant that the monster is shot by the player and it changes its target of attack. The player can harm the monster without changing its target, but it will attack the player after its current target is defeated.

Type 0

The monster will waken and attack the target. If distracted it will also attack the player. Assuming it wins and is not distracted, it will return to being a normal monster afterwards.

Type 1

The monster will attack the target upon sight. If distracted, it will also attack the player. Assuming it wins and is not distracted, it will return to sleep afterwards and ignore the player unless it is harmed.

Type 2

The monster will waken and attack the target. If distracted, it will also attack the player. Assuming it wins and is not distracted, it will return to sleep afterwards and ignore the player unless it is harmed.

Type 3

The monster will attack the target upon sight. If distracted, it will also attack the player. Assuming it wins and is not distracted, it will return to being a normal monster afterwards.

Type 4

The same as type 0.

Type 5

The monster will attack the target upon sight. It will never attack the player before or after.

Type 6

The monster will waken and attack the target. It will never attack the player before or after.

Although it seems a bit complicated, there are three basic types with a slight variation for each. These are: attack like a demon (3 and 4), attack like an ally to the player (5 and 6), and attack like an ally of the player unless friendly fire occurs (1 and 2). The slight variation is to waken the monster (even numbers) or not (odd numbers).

Examples

This script makes all monster with thing id of 100 waken and try to attack the player at the start of the map:

```
script 1 ENTER
{
```

```
    Thing_Hate(100, 0, 0);  
}
```

This script uses "ENTER" instead of "OPEN" because OPEN is run by the map, but ENTER is run by each player on entering the map. See script types.

As this script is very simple and only uses one command, Thing_Hate could be invoked with a switch, a line or a thing such an Actor enters sector thing rather than by a script such as this.

The following script runs a cutscene where some marines attack some demons. It takes three parameters, the TID of the marines, the TID of the demons, and the TID of a camera to show the action from.

```
script 1 (int marines, int demons, int cam)  
{  
    ChangeCamera(cam, 1, 0);  
    PrintBold(s:"The marines are storming\n  
        the demon's stronghold!");  
  
    Thing_Hate(marines, demons, 6);  
    Thing_Hate(demon, marines, 3);  
  
    Delay(35*10);  
  
    ChangeCamera(0, 1, 0);  
}
```

The first two lines set up the cutscene to a camera and explain what is happening. The Thing_Hate lines set up the battle. The marines are awakened and set to be allies of the player. The demons are idle, but will attack the marines or player on sight. This gives the illusion of the marines acting like other players. The last two lines let the action unfold a little then end the cutscene.

Note that groups of enemies can be made to fight using Thing_Hate in such a way.

Thing_Move

125:Thing_Move (tid, destid, nofog)

tid: tid of the thing to move

destid: tid of the thing to move to

nofog: do not spawn fog

Moves a thing to the location of another thing. If there are multiple things that have the given tids, only the ones with the highest thing numbers will be used. The destination will generally be a map spot or teleporter destination, but it can be anything.

If nofog is 0, a teleport flash is spawned at the source and destination spot.

Thing_Projectile

134:Thing_Projectile (tid, type, angle, speed, vspeed)

Spawns a projectile. Note that this special requires the actor being spawned to have a SpawnID. To spawn an actor without one, use the SpawnProjectile function instead.

tid: Thing ID of the map spot to spawn the projectile at

type: Type of projectile to spawn, from the list of spawn numbers.

When using this special on a linedef or a thing in UDMF, you can use the arg1str property to define a class name instead of the arg1 property to define a spawn number.

angle: Byte angle of the projectile

speed: Horizontal speed of the projectile in units per 8 tics

vspeed: Vertical speed of the projectile in units per 8 tics (up is positive)

Thing_ProjectileAimed

178:Thing_ProjectileAimed (tid, type, speed, target, newtid)

tid: Thing ID of the map spot to spawn the projectile at

type: Type of projectile to spawn

When using this special on a linedef or a thing in UDMF, you can use the arg1str property to define a class name instead of the arg1 property to define a spawn number.

speed: Speed of the projectile in units per 8 tics

target: Thing ID of the thing you want the projectile fired at

newtid: TID you want assigned to the spawned projectile (optional)

Spawns a projectile that is fired at a target which can be any thing on the map. Note that this special requires the actor being spawned to have a SpawnID. To spawn an actor without one, use the SpawnProjectile function instead.

Usage

tid is a number that makes a thing unique, the default is zero, and therefore a TID is usually set to something else, like 1.

In Doom Builder you have to right click on a thing, select the "Effects" tab and change the Thing Tag to something besides 0.

The type is actually from the list of Spawn_numbers. You can either use the number or the defined name (T_SOMETHING) for clarity.

speed is a number between 0-255.

target TID is the same idea as the TID above, to see an example of changing the players TID check out Thing_ChangeTID.

newtid is just a way to make the spawned thing unique if you need to refer to it later. It is just a number.

Thing_ProjectileGravity

136:Thing_ProjectileGravity (tid, type, angle, speed, vspeed)

Spawns a projectile and subjects it to gravity. Note that this special requires the actor being spawned to have a SpawnID. To spawn an actor without one, use the SpawnProjectile function instead.

tid: Thing ID of the map spot to spawn the projectile at

type: Type of projectile to spawn, from the list of Spawn numbers

When using this special on a linedef or a thing in UDMF, you can use the arg1str property to define a class name instead of the arg1 property to define a spawn number.

angle: Byte angle of the projectile

speed: Horizontal speed of the projectile in units per 8 tics

vspeed: Vertical speed of the projectile in units per 8 tics (up is positive)

Thing_ProjectileIntercept

175:Thing_ProjectileIntercept (tid, type, speed, target, newtid)

tid: Thing ID of the map spot to spawn the projectile at

type: Type of projectile to spawn

speed: Speed of the projectile in units per 8 tics

target: Thing ID of the thing you want the projectile fired at

newtid: TID you want assigned to the spawned projectile (optional)

Spawns a projectile, much like Thing_ProjectileAimed, but tries to intercept the target. Basically, it tries to predict where its target will be based on the target's current speed and direction. If the target changes speed/direction after the projectile is spawned, chances are the projectile will not hit. The main way it is used is to make a hassle for the player. By giving a player a TID (see Thing_ChangeTID), you can use this special to send the fireballs their way.

The higher the projectile's speed, the harder it is to dodge it, since there is less of a window of opportunity to change your speed/direction and get out of its way before it hits home.

Usage

tid is a number that makes a thing unique, the default is zero, and therefore a TID is usually set to something else, like 1. In Doom Builder you have to right click on a thing, select the Effects tab and change the Thing Tag to something besides 0.

The type is actually from the list of Spawn_numbers. You can either use the number or the defined name (T_SOMETHING) for clarity.

When using this special on a linedef or a thing in UDMF, you can use the arg1str property to define a class name instead of the arg1 property to define a spawn number.

speed is a number between 0 and 255.

target same TID idea as mentioned above.

newtid is just a way to make the spawned thing unique if you need to refer to it later. It is just a number.

Thing_Raise

17:Thing_Raise (tid, nocheck)

tid: The TID of the thing(s) you want to resurrect.

nocheck: If non-zero, no check for room before resurrecting is performed.

Resurrects the specified thing like an archvile does. If tid is 0, it tries to resurrect the activator. This special does not work on players.

Examples

This script will print a message on the screen and cause the tagged monsters to resurrect, though it will only do that on skill difficulty 3 or higher.

Script 1 (int tid)

```
{
  If(GameSkill() >= SKILL_HARD)
  {
    Print(s:"Prepare to die!!");
    Delay(52);
    Thing_Raise(tid);
  }
}
```

This zombie will keep coming back up each time it is dispatched normally without being gibbed.

ACTOR ResZombieMan : ZombieMan

```
{
  States
  {
    Death:
      POSS H 5
      POSS I 5 A_Scream
      POSS J 5 A_NoBlocking
      POSS K 5
      POSS L random(35, 105) // Wait between 1-3 seconds before rising
      POSS L -1 Thing_Raise(0)
      Stop
  }
}
```

Note that for the special to have an effect like in the above example, the state from which the special is called has to have an infinite duration (this is not the case if the CanRaise state flag is used).

Thing_Remove

132:Thing_Remove (tid)

tid: Thing ID of the thing to remove
Removes the specified thing from the map.

Examples

This script removes anything with a TID of 969, which so happens to be some mancubi in this example.

```
script 69 (void)
{
    Thing_Remove(969);

    print(s:"cgRemoved mancubi!");
}
```

Thing_SetConversation

79:Thing_SetConversation (tid, convid)

tid: TID of affected thing. If this is 0, the activator is used.

convid: The number of the conversation associated to the thing. If this is 0, the thing is set not to have any conversation. Sets the conversation of the thing with the same TID. This can also be used to remove a conversation from a thing, by setting it to 0.

Thing_SetGoal

229:Thing_SetGoal (tid, goal, delay, dontchasetarget)

tid: Thing ID of monster to send to a patrol point

goal: Thing ID of the destination patrol point

delay: Seconds before the monster starts moving toward the patrol point

dontchasetarget: If non-zero, even walk towards the goal if the monster has a valid target. It will still attack, though.

This special causes a monster to start following a path beginning with the specified patrol points. If tid is 0, Thing_SetGoal can be used as a monster's special to automatically start it walking toward a goal when it is spawned.

Thing_SetSpecial

127:Thing_SetSpecial (tid, special, arg0, arg1, arg2)

tid: TID of affected thing. If this is 0, the activator is used.

special: The number of the special that the thing will have.

arg0, arg1, arg2: The first three arguments for the special that the thing will have.

Sets the special of the thing with the same TID. For technical reasons, it can only affect the first three parameters; for a more flexible function see SetThingSpecial.

Thing_SetTranslation

180:Thing_SetTranslation (tid, translation)

tid: Thing ID of the actor to apply the translation to. If this is 0, the activator is used.

translation: Translation number (set up with CreateTranslation)

Assigns a translation to a specified thing. Use -1 for translation to change the translation to that currently being used by the activator of the script. So, if you have a scripted marine with a tid of 17, then to make him have the same color as the player you would do:

```
script 1 enter
```

```
{
```

```
    Thing_SetTranslation(17, -1);
```

```
}
```

Additionally, specifying TRANSLATION_ICE as the translation to use, allows the application of the ice translation.

Thing_Spawn

135:Thing_Spawn (tid, type, angle, newtid)

Spawns a thing using its spawn ID. For spawning things by name, use SpawnSpot instead.

tid: Thing ID of the map spot where to spawn the thing

type: Type of thing to spawn, from the list of spawn numbers

When using this special on a linedef or a thing in UDMF, you can use the arg1str property to define a class name instead of the arg1 property to define a spawn number.

angle: Byte angle for the thing to face

newtid: TID to give spawned thing

Thing_SpawnFacing

Jump to navigationJump to search

139:Thing_SpawnFacing (tid, type, nofog, newtid)

tid: TID of map spot to spawn things at.

type: Type of things to spawn.

nofog: TRUE if there should be no fog, FALSE otherwise.

newtid: TID to give to the newly spawned things.

This special is like Thing_Spawn and Thing_SpawnNoFog combined, except that the angle the newly spawned thing is facing will match the angle of the map spot it spawns at.

Note that this special requires the actor being spawned to have a SpawnID. To spawn an actor without one, use the SpawnSpotFacing function instead.

Usage

TID is a number that makes a thing unique, the default is zero, and therefore a TID is usually set to something else, like 1. In Doom Builder you have to right click on a thing, select the Effects tab and change the Thing Tag to something besides 0.

The type is actually from the list of Spawn numbers. You can either use the number or the defined name (T_SOMETHING) for clarity.

When using this special on a linedef or a thing in UDMF, you can use the arg1str property to define a class name instead of the arg1 property to define a spawn number.

New TID is just a way to make the spawned thing unique if you need to refer to it later. It is just a number.

Examples

This script spawns a Demon at points 1 to 4 with a delay in between:

```
script 1 (void)
{
    Thing_SpawnFacing(1, T_DEMON, FALSE, 0);

    delay(30);

    Thing_SpawnFacing(2, T_DEMON, FALSE, 0);

    delay(30);

    Thing_SpawnFacing(3, T_DEMON, FALSE, 0);

    delay(30);

    Thing_SpawnFacing(4, T_DEMON, FALSE, 0);
}
```

Thing_SpawnNoFog

137:Thing_SpawnNoFog (tid, type, angle, newtid)

tid: Thing ID of the map spot to spawn the thing at

type: Type of thing to spawn

angle: Byte angle for the thing to face

newtid: TID to give spawned thing

Spawns a thing without the teleporter fog. Note that this special requires the actor being spawned to have a SpawnID. To spawn an actor without one, use the SpawnSpot function instead.

Usage

tid is a number that makes a thing unique, the default is zero, and therefore, a TID is usually set to something else, like 1. In Doom Builder, you have to right-click on a thing, select the “Effects” tab and change the Thing Tag to something besides 0.

The type is actually from the list of spawn numbers. You can either use the number or the defined name (T_SOMETHING) for clarity.

When using this special on a linedef or a thing in UDMF, you can use the arg1str property to define a class name instead of the arg1 property to define a spawn number.

angle is a number between 0–255, representing a byte angle.

newtid is just a way to make the spawned thing unique if you need to refer to it later. It is just a number.

Thing_Stop

19:Thing_Stop (tid)

Usage

This stops the specified actor's current movement by setting its acceleration and speed to 0.

Example

This example freezes the player and prints a message to the player, as well as freezing him in place.

Script 1 (VOID)

```
{  
    Print(s:"STOP RIGHT THERE YOU FREAK!");  
  
    Thing_Stop(0);  
    SetPlayerProperty(0, 1, PROP_TOTALLYFROZEN);  
}
```

ThrustThing

72:ThrustThing (angle, force, nolimit, tid)

angle: Byte angle to thrust the thing

force: The force, in units per tic (1 second = 35 tics), to apply to the thing.

nolimit: Must be set to 1 if the thrust is larger than 30

tid: TID of the thing to thrust. If this is 0, the activator of the special is affected itself.

Thrusts the thing with the given tid in the direction specified by angle with the given force. Can be easily combined with ThrustThingZ.

ThrustThing can also be used to thrust based on actor angle.

```
ThrustThing(angle * 256 / 360, 0, 0, 0)
```

```
ThrustThing(angle * 256 / 360 + 64, 0, 0, 0)
```

The first line will thrust in the direction the actor is facing. Adding +64, +128, or +192 will thrust to the right, backwards, or left, respectively.

Examples

This script combines ThrustThing with ThrustThingZ to spawn an arachnotron, and make it jump ~500 map units into the air and ~200 map units to the east.

Script "Arachnotron jump" (void)

```
{
    SpawnSpotFacingForced("Arachnotron", 142, 143);
    Delay(1);
    ThrustThingZ(143, 115, 0, 0);
    Delay(18);
    ThrustThing(0, 10, 1, 143);
}
```

ThrustThingZ

128:ThrustThingZ (tid, force, up/down, set/add)

tid: Thing ID of the thing to thrust. If 0, this will thrust the activator (usually the player).

force: The force, in units per tic (1 second = 35 tics) divided by 4, to apply to the thing.

up/down: The direction to thrust the thing. (0 = up, 1 = down)

set/add: If 0, sets the thing's vertical speed to 0 and only then applies the force. If 1, adds the speed resulting from the special's force to the thing's current vertical speed.

Thrusts the thing vertically, given the specified force. It can either apply the force to the thing's current vertical speed or it can set the thing's vertical speed to 0 before applying the force. Can easily be combined with ThrustThing.

Transfer_CeilingLight

211:Transfer_CeilingLight (tag)

tag: Tag of affected sector
The amount of lighting on the tagged sector's ceiling will be the same as the lighting in the sector that this line faces.

Conversions from linedef types
The following Doom map format types can be converted as Transfer_CeilingLight:

Type	Conversion	Trigger
Boom 261:Transfer Ceiling Light	Transfer_CeilingLight (tag)	

Transfer_FloorLight

210:Transfer_FloorLight (tag)

tag: Tag of affected sector


The amount of lighting on the tagged sector's floor will be the same as the lighting in the sector that this line faces.

Conversions from linedef types

The following Doom map format types can be converted as Transfer_FloorLight:

Type	Conversion	Trigger
Boom 213:Transfer Floor Light		Transfer_FloorLight (tag)

tag: Tag of affected sector
flags: Defines the properties of the effect
This is Boom's 242 linedef. It is used to divide a sector into upper, lower, and middle regions. The drawn heights of the tagged sector's floor and ceiling come from the heights of the sector on the line's front side, used as the control sector. The line's texture names can be used to specify a special color map or palette blend to be used in those regions. Palette blends are of the form "AARRGGBB". You can use the testblend command to find a good blend from inside the game.

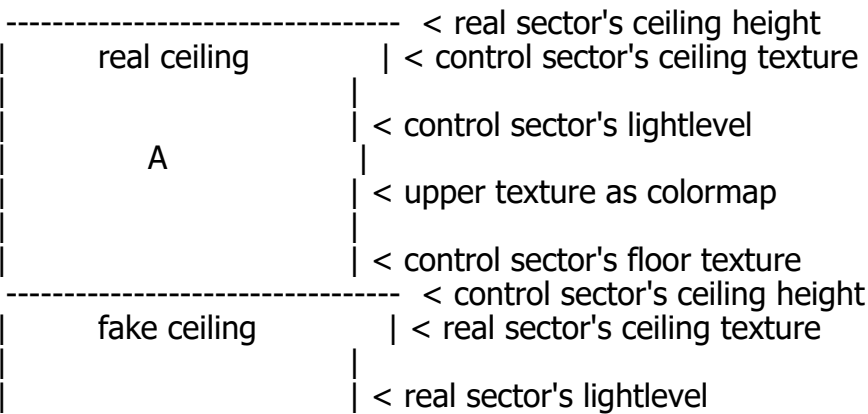
ZDoom supports the internal name "WATERMAP". If you place this on your linedef instead of a blend value, ZDoom will create a blend of the value "80004FA5", which is a nice misty underwater blue: .

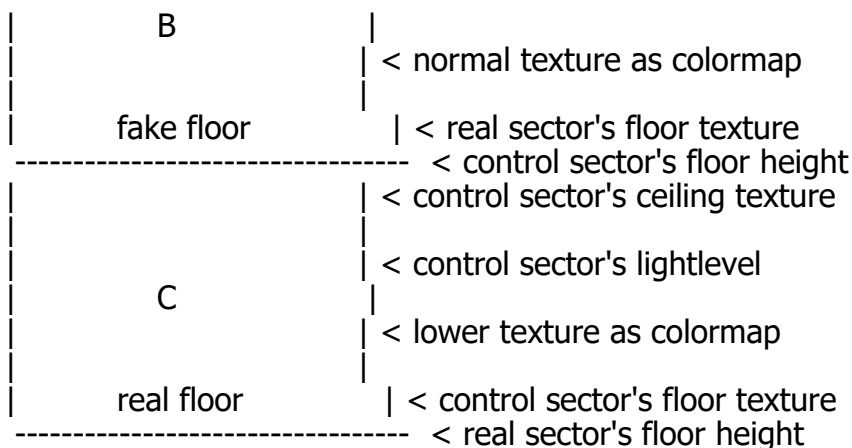
The flags parameter determines how the fake floor and ceiling are rendered and can have values such as:

- 0 — There will be situations when the sector's real ceiling and floor heights will be used instead of the fake heights (this is the way Boom does it).
 - 1 — The fake ceiling and floor heights will always be used.
 - 2 — Only the fake floor will be drawn; also allows the real ceiling to be drawn below the fake floor (useful for multi-sector underwater areas).
 - 4 — Improved texture control -
The real floor and ceiling will be drawn with the real sector's flats.
The fake floor and ceiling (from either side) will be drawn with the control sector's flats.
The real floor and ceiling will be drawn even when in the middle part, allowing lifts into and out of deep water to render correctly (not possible in Boom).
 - 8 — Make the target sector swimmable under the fake floor (no need for the WaterZone actor)
 - 16 — Do not draw the fake floor or ceiling; however, they can still be used in conjunction with the sector action things that correspond to them (useful for making the player fall down a deep hole or similar).
 - 32 — Do not transfer the control sector's lighting to the affected sector(s).
- These values can also be combined by adding them. For example a value of 6 would combine values 2 and 4, and a value of 29 would combine values 16, 8, 4 and 1.

The following description is an excerpt from boomref.txt:

This allows the tagged sector to have two levels — an actual floor and ceiling, and another floor or ceiling where more flats are rendered. Things will stand on the actual floor or hang from the actual ceiling, while this function provides another rendered floor and ceiling at the heights of the sector on the first sidedef of the linedef. Typical use is "deep water" that can be over the player's head. [Note: See the WaterZone thing (9045)]





Boom sectors controlled by a 242 linedef are partitioned into 3 spaces. The viewer's Z coordinate uniquely determine which space they are in.

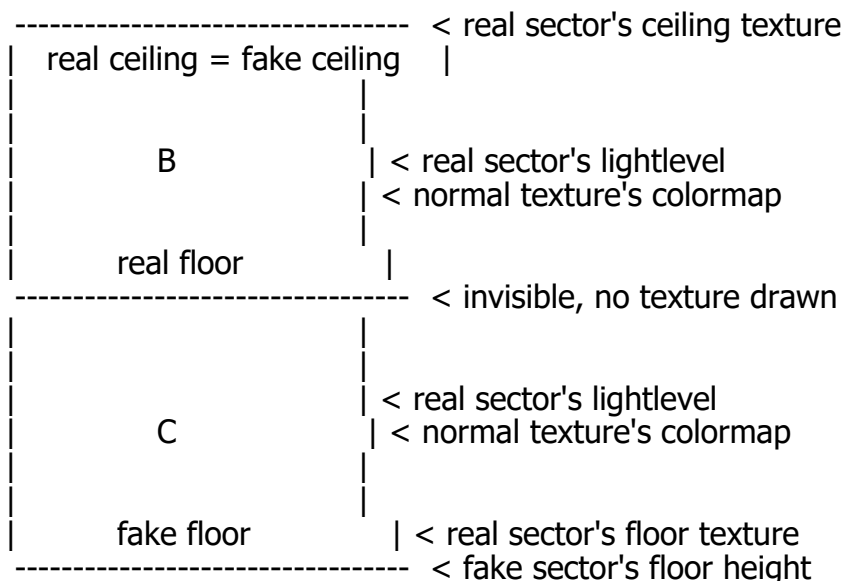
If they are in space B (normal space), then the floor and ceiling textures and lightlevel from the real sector are used, and the colormap from the 242 linedef's first sidedef's normal texture is used (COLORMAP is used if it is invalid or missing). The floor and ceiling are rendered at the control sector's heights.

If they are in space C ("underwater"), then the floor and ceiling textures and lightlevel from the control sector are used, and the lower texture in the 242 linedef's first sidedef is used as the colormap.

If they are in space A ("head over ceiling"), then the floor and ceiling textures and lightlevel from the control sector are used, and the upper texture in the 242 linedef's first sidedef is used as the colormap.

If only two of these adjacent partitions in z-space are used, such as underwater and normal space, one has complete control over floor textures, ceiling textures, light level, and colormaps, in each of the two partitions. The control sector determines the textures and lighting in the more "unusual" case (e.g. underwater).

It is also possible for the fake floor to extend below the real floor, in which case an invisible platform/stair effect is created. In that case, the picture looks like this (barring any ceiling effects too):



In this case, since the viewer is always at or above the fake floor, no colormap/lighting/texture changes occur — the fake floor just gets drawn at the control sector's height, but at the real sector's lighting and texture, while objects stand on the higher height of the real floor.

It is the viewer's position relative to the fake floor and/or fake ceiling which determines whether the control

sector's lighting and textures should be used, and which colormap should be used. If the viewer is always between the fake floor and fake ceiling, then no colormap, lighting, or texture changes occur, and the view just sees the real sector's textures and light level drawn at possibly different heights.

If the viewer is below the fake floor height set by the control sector, or is above the fake ceiling height set by the control sector, then the corresponding colormap is used (lower or upper texture name), and the textures and lighting are taken from the control sector rather than the real sector. They are still stacked vertically in standard order — the control sector's ceiling is always drawn above the viewer, and the control sector's floor is always drawn below the viewer.

The kaleidoscope effect only occurs when F_SKY1 is used as the control sector's floor or ceiling. If F_SKY1 is used as the control sector's ceiling texture, then under water, only the control sector's floor appears, but it "envelops" the viewer. Similarly, if F_SKY1 is used as the control sector's floor texture, then when the player's head is over a fake ceiling, the control sector's ceiling is used throughout.

F_SKY1 causes HOM when used as a fake ceiling between the viewer and normal space. Since there is no other good use for it, this kaleidoscope is an option turned on by F_SKY1. Note that this does not preclude the use of sky REAL ceilings over deep water — this is the control sector's ceiling, the one displayed when the viewer is underwater, not the real one.

A colormap has the same size and format as Doom's COLORMAP. Extra colormaps may be defined in Boom by adding them between C_START and C_END markers in WADs. Colormaps between C_START and C_END are automatically merged by Boom with any previously defined colormaps.

Ceiling bleeding may occur if required upper textures are not used.

Limitations

There is some limited capability to look into the lower or middle part (but not the upper part) from a non-Transfer_Heights sector - but you will never be able to see both lower and middle parts of different sectors at the same time. This will always create a hall of mirrors effect.

As a corollary of the above, you cannot look from a lower part of a Transfer_Heights sector into the middle part of another one. This is an inherent limitation of how this feature works.

A sector cannot have both transferred heights and 3D floors (you can but it will cause some rendering issues).

Sector damage on the control sector are not transferred to the target sectors.

Conversions from linedef types

The following Doom map format types can be converted as :

Type Conversion Trigger

Boom 242:Transfer Heights Transfer_Heights (tag)

Legacy 280:Swimmable Pool Transfer_Heights (tag, 12)

ZDoom 350:Fake Floor (real texture) Transfer_Heights (tag, 2)

ZDoom 351:Fake Floor (control texture) Transfer_Heights (tag, 6)

Transfer_WallLight

16:Transfer_WallLight (lineid, flags)

lineid: ID of the lines to transfer the light to

flags: Specifies what part of the line is affected

Transfers the light level from the line's front sector to the tagged line. The flags parameter specifies what exactly will happen. The following values can be set:

1: Transfer light level to the line's front side

2: Transfer light level to the line's back side

4: Ignore fake contrast setting (i.e. the light level is absolute). Fake contrast is on by default and can be specified in MAPINFO or disabled (with the "evenlighting" keyword) or modified (with the "smoothlighting" keyword)

TranslucentLine

208:TranslucentLine (lineid, amount, additive, moreflags)

Warning: This special is not fully supported by the UDMF map format. Certain parameters, noted below, have been made obsolete by the ability to directly specify properties within the linedef structure. These parameters should be set to 0 when using the UDMF format.

lineid: Line ID of lines to make translucent (0 for this line). Obsolete in UDMF.
amount: How translucent the line should be. Ranges from 0 to 255.
additive: Whether this translucent line should use additive translucency or not (0 = normal, 1 = additive)
moreflags: Sets extended lineflags. See Line_SetIdentification for details. Obsolete in UDMF. This parameter only has a function when used directly in a map.
Sets the amount of translucency for all matching lines (including itself). If lineid is 0, it only sets the translucency of the line it is on. Like Line_SetIdentification, this special sets the linedef's id. Amount controls how opaque the line is. 0 is nearly invisible. 255 is opaque. Intermediate values are somewhere in between.

You can also use this special in an ACS script to change the translucency for lines whose ids match lineid.

Conversions from linedef types
The following Doom map format types can be converted as TranslucentLine:

Type	Conversion	Trigger
Boom 260:Translucent Line	66%	TranslucentLine (lineid, 168)
Legacy 288:Opaque Line		TranslucentLine (lineid, 255, 0)
Legacy 284:Translucent Line	50%	TranslucentLine (lineid, 128, 0)
Legacy 285:Translucent Line	75%	TranslucentLine (lineid, 192, 0)
Legacy 286:Translucent Line	18%	TranslucentLine (lineid, 48, 0)
Legacy 287:Additive Line	50%	TranslucentLine (lineid, 128, 1)
EDGE 409:Translucent Line	80%	TranslucentLine (lineid, 204)
EDGE 410:Translucent Line	60%	TranslucentLine (lineid, 153)
EDGE 411:Translucent Line	40%	TranslucentLine (lineid, 101)
EDGE 412:Translucent Line	20%	TranslucentLine (lineid, 50)

Action functions

Jump to navigationJump to search

The following are all the code pointers supported by GZDoom's DECORATE and ZScript lumps as of the latest source code. These are placed at the end of an individual frame definition within the actor's state block. You may also use any action special in place of an action function. See also the category for a more exhaustive listing.

Functions may have a return type (e.g. return A_FunctionName). See anonymous functions for details.

Monster AI

- A_AlertMonsters
- A_Burst
- A_CentaurDefend
- A_Chase
- A_CheckForResurrection
- A_ClearLastHeard
- A_ClearSoundTarget
- A_ClearTarget
- A_DamageChildren
- A_DamageMaster
- A_DamageSelf
- A_DamageSiblings
- A_DamageTarget
- A_DamageTracer
- A_Die
- A_ExtChase (deprecated)
- A_FaceMaster
- A_FaceTarget
- A_FaceTracer
- A_FastChase
- A_KillChildren
- A_KillMaster
- A_KillSiblings
- A_KillTarget
- A_KillTracer
- A_Look
- A_Look2
- A_LookEx
- A_RaiseChildren
- A_RaiseMaster
- A_RaiseSelf
- A_RaiseSiblings
- A_RemoveChildren
- A_RemoveMaster
- A_RemoveSiblings
- A_RemoveTarget
- A_RemoveTracer
- A_Remove
- A_SentinelBob
- A_Teleport
- A_TurretLook
- A_VileChase
- A_Wander

Generic monster attacks

- A_BasicAttack
- A_BulletAttack
- A_ComboAttack (deprecated)
- A_CustomMissile (deprecated)
- A_CustomBulletAttack
- A_CustomRailgun
- A_CustomMeleeAttack
- A_CustomComboAttack
- A_Detonate
- A_Explode
- A_MeleeAttack (deprecated)
- A_MissileAttack (deprecated)
- A_MonsterRefire
- A_MonsterRail
- A_RadiusDamageSelf
- A_RadiusThrust
- A_SpawnProjectile
- A_ThrowGrenade
- A_WolfAttack

Freeze death functions

- A_FreezeDeath
- A_GenericFreezeDeath
- A_FreezeDeathChunks
- A_IceGuyDie

Sound functions

- A_StartSound
- A_StartSoundIfNotSame
- A_PlaySound (deprecated)
- A_PlaySoundEx (deprecated)
- A_PlayWeaponSound (deprecated)
- A_ActiveSound
- A_LoopActiveSound
- A_FLoopActiveSound
- A_StopSound
- A_StopSounds
- A_StopAllSounds
- A_StopSoundEx (deprecated)
- A_SoundPitch
- A_SoundVolume
- A_Pain
- A_Scream
- A_XScream
- A_PlayerScream
- A_VileStart
- A_BrainPain
- A_BrainAwake
- A_BFGSound

Print actions

- A_Print
- A_PrintBold
- A_Log

A_LogFloat
A_LogInt

Special actions

A_BossDeath
A_KeenDie
A_BrainDie
A_GetHurt
A_KlaxonBlare
A_CheckTerrain
A_SetBlend
A_CheckPlayerDone
A_PlayerSkinCheck
A_SkullPop
A_SprayDecal
A_Quake
A_QuakeEx
A_CopySpriteFrame
A_SetSpriteAngle
A_SetSpriteRotation
A_SpriteOffset
Spawn functions
A_TossGib
A_SpawnDebris
A_SpawnItem
A_SpawnItemEx
A_SpawnParticle
A_SpawnParticleEx (New from 4.9.0)

State jumps

A_CheckBlock
A_CheckCeiling
A_CheckFloor
A_CheckFlag (deprecated)
A_CheckLOF
A_CheckProximity
A_CheckRange
A_CheckSight
A_CheckSightOrRange
A_CheckSpecies
A_Jump
A_JumpIf
A_JumpIfArmorType
A_JumpIfCloser
A_JumpIfHealthLower
A_JumpIfHigherOrLower
A_JumpIfInventory
A_JumpIfInTargetInventory
A_JumpIfInTargetLOS
A_JumpIfMasterCloser
A_JumpIfNoAmmo
A_JumpIfTargetInLOS

A_JumpIfTargetInsideMeleeRange
A_JumpIfTargetOutsideMeleeRange
A_JumpIfTracerCloser

Status changes

A_ActiveAndUnblock
A_ChangeCountFlags
A_ChangeFlag (deprecated)
A_ChangeModel (New from 4.10.0)
A_ChangeVelocity
A_ClearShadow
A_CopyFriendliness
A_DeQueueCorpse
A_FadeIn
A_FadeOut
A_FadeTo
A_FaceMovementDirection
A_Fall
A_Gravity
A_HideThing
A_LowGravity
A_Morph
A_NoBlocking
A_NoGravity
A_QueueCorpse
A_RearrangePointers
A_ResetHealth
A_Respawn
A_ScaleVelocity
A_ScreamAndUnblock
A_SetAngle
A_SetArg
A_SetChaseThreshold
A_SetDamageType
A_SetFloat
A_SetFloatSpeed
A_SetFloatBobPhase
A_SetFloorClip
A_SetFriendly
A_SetGravity
A_SetHealth
A_SetInvulnerable
A_SetMass
A_SetMugshotState
A_SetPainThreshold
A_SetPitch
A_SetReflective
A_SetReflectiveInvulnerable
A_SetRenderStyle
A_SetRipperLevel
A_SetRipMin
A_SetRipMax
A_SetRoll

A_SetScale
A_SetShadow
A_SetShootable
A_SetSize
A_SetSolid
A_SetSpecial
A_SetSpecies
A_SetSpeed
A_SetTeleFog
A_SetTics
A_SetTranslucent
A_SetUserArray
A_SetUserArrayFloat
A_SetUserVar
A_SetUserVarFloat
A_SetViewAngle
A_SetViewPitch
A_SetViewRoll
A_SetTranslation
A_SetVisibleRotation
A_SwapTeleFog
A_TransferPointer
A_UnHideThing
A_UnsetFloat
A_UnsetFloorClip
A_UnsetInvulnerable
A_UnsetReflective
A_UnsetReflectiveInvulnerable
A_UnsetShootable
A_UnsetSolid
Dynamic lights
A_AttachLight
A_AttachLightDef
A_RemoveLight

Missile movement

A_SeekerMissile
A_Tracer
A_Tracer2
A_FaceTracer
A_Fire
A_BishopMissileWeave
A_CStaffMissileSlither
A_Weave
A_Warp
A_Countdown
A_CountdownArg
A_Stop

Inventory functions

A_DropInventory
A_DropItem

A_GiveInventory
A_GiveToChildren
A_GiveToSiblings
A_GiveToTarget
A_TakeInventory
A_TakeFromChildren
A_TakeFromSiblings
A_TakeFromTarget
A_SelectWeapon
A_SetInventory
A_RadiusGive

Weapon functions

A_WeaponReady
A_WeaponOffset
A_Lower
A_Raise
A_ReFire
A_ClearReFire
A_GunFlash
A_CheckReload
A_CheckForReload
A_CheckRailReload SkulltagIcon22.png (Skulltag only: not supported by ZDoom) (deprecated)
A_ResetReloadCounter
A_Light
A_Light0
A_Light1
A_Light2
A_LightInverse
A_ClearOverlays
A_Overlay
A_OverlayAlpha
A_OverlayFlags
A_OverlayOffset
A_OverlayPivot
A_OverlayPivotAlign
A_OverlayRenderstyle
A_OverlayRotate
A_OverlayScale
A_OverlayTranslation
A_OverlayVertexOffset
A_Recoil
A_ZoomFactor
A_SetCrosshair

Weapon attack functions

A_CustomPunch
A_FireBullets
A_FireCustomMissile (deprecated)
A_FireProjectile
A_FireAssaultGun
A_FireBFG

A_FireOldBFG
A_FirePistol
A_FireShotgun
A_FireShotgun2
A_FireCGun
A_FireMissile
A_FirePlasma
A_FireSTGrenade (deprecated)

A_Punch
A_RailAttack
A_Saw

Script functions
ACS_NamedExecute
ACS_NamedSuspend
ACS_NamedTerminate
ACS_NamedLockedExecute
ACS_NamedLockedExecuteDoor
ACS_NamedExecuteWithResult
ACS_NamedExecuteAlways

Original Doom/Strife monster attacks

A_PosAttack
A_SPosAttack
A_CPosAttack
A_CPosRefire
A_SpidRefire
A_TroopAttack
A_SargAttack
A_HeadAttack
A_BruisAttack
A_SkullAttack
A_BspiAttack
A_CyberAttack
A_PainAttack
A_DualPainAttack
A_PainDie
A_SkelFist
A_SkelMissile
A_FatAttack1
A_FatAttack2
A_FatAttack3
A_VileTarget
A_VileAttack
A_BrainSpit
A_SpawnFly
A_SpawnSound
A_BrainScream
A_BrainExplode
A_Mushroom
A_M_Saw
A_SentinelRefire
A_BetaSkullAttack

Miscellaneous functions for Doom

A_Hoof

A_Metal

A_BabyMetal

A_FatRaise

A_SkelWhoosh

A_StartFire

A_FireCrackle

A_BFGSpray

A_BarrelDestroy

A_AttachLight

bool A_AttachLight (Name lightid, int type, Color lightcolor, int radius1, int radius2 [, int flags [, Vector3 ofs [, double param [, double spoti [, double spoto [, double spotp]]]]]])

Note: this function is not fully supported by DECORATE. If used from DECORATE, all parameters starting from ofs should be omitted. Consider switching to ZScript to fully utilize this function.

Usage

Creates and attaches a dynamic light to the calling actor.

Can be removed with A_RemoveLight.

Parameters

lightid: an identifier for the light.

type: the type of the light, which can be one of the following:

DynamicLight.PointLight - PointLight. Default setting.

DynamicLight.PulseLight - PointLightPulse.

DynamicLight.FlickerLight - PointLightFlicker.

DynamicLight.RandomFlickerLight - PointLightFlickerRandom.

DynamicLight.SectorLight - SectorPointLight.

DynamicLight.DummyLight - (Need more info)

DynamicLight.ColorPulseLight - (Need more info)

DynamicLight.ColorFlickerLight - (Need more info)

DynamicLight.RandomColorFlickerLight - (Need more info)

lightcolor: the color of the light. This can be a color variable or a string name (such as "Red").

radius1: the primary radius of the light.

radius2: the secondary radius of the light, primarily for lights that flicker or pulse.

flags: this can be combined with the '|' separator. Default is 0.

DYNAMICLIGHT.LF_SUBTRACTIVE - Light becomes subtractive (darkening effect).

DYNAMICLIGHT.LF_ADDITIVE - Light becomes additive, making the colors whiter.

DYNAMICLIGHT.LF_DONTLIGHTSELF - Actor does not light itself.

DYNAMICLIGHT.LF_ATTENUATE - Light uses the angle attenuation formula, providing more realistic (if slightly dimmer) illumination on angled surfaces.

DYNAMICLIGHT.LF_NOSHADOWMAP - Light will not create shadow maps, which are realistic shadows created from map geometry obscuring other surfaces.

DYNAMICLIGHT.LF_DONTLIGHTACTORS - Light will not affect other actors.

DYNAMICLIGHT.LF_SPOT - Light is a spot light. Spot lights are in cones like a lamp, or a flash light.

DYNAMICLIGHT.LF_DONTLIGHTOTHERS - The light will not light other actors, only the actor that it is attached to.

DYNAMICLIGHT.LF_DONTLIGHTMAP - The light will not light up the level geometry, only actors.

ofs: the offset of the light as a vector. Default is (0, 0, 0).

param: if the light is a pulse light, this represents the interval to switch between the two radii in seconds. Otherwise, it is used for flicker lights (between 0 and 1) to represent the chance to be the first radius or the second. Default is 0.

spoti: Sets the inner angle of a spot light. Ignored if not a spot light. Default is 10.

spoto: Sets the outer angle of a spot light. Default is 25.

spotp: Sets the pitch of the spot light. Default is 0.

Return value

Returns true when the dynamic light has been spawned.

A_AttachLightDef

bool A_AttachLightDef (Name lightid, Name lightdef)

Usage

Attaches a dynamic light defined in GLDEFS to the calling actor.

Contrast to A_AttachLight which allows you to define, finely tune and control the light spawned, this function can be useful if you have a pre-existing definition you want to reuse multiple times.

Can be removed with A_RemoveLight.

Parameters

lightid: an identifier for the light.

lightdef: the dynamic light to attach, as defined in GLDEFS.

Return value

(Need more info)

Examples

// GLDEFS

flickerlight2 FLAMETHROWER_LIGHT_ATTACH

```
{
    color 1.0 0.6 0.0
    size 80
    secondarySize 96
    interval 1
    offset 0 40 0
}
```

// ZScript

...

States

{

Ready:

 // Call this before the loop

 FLMG A 0 Bright A_AttachLightDef('FlamethrowerLight', 'FLAMETHROWER_LIGHT_ATTACH');

 FLMG ABCD 2 Bright A_WeaponReady;

 // I call this beforehand... don't know if it's necessary

 FLMG D 0 Bright A_RemoveLight('FlamethrowerLight');

 Loop;

Deselect:

 // Remove the light when deselecting the weapon

 FLMG H 0 A_RemoveLight('FlamethrowerLight');

 FLMG HGFE 1 A_Lower;

 Wait;

Fire:

 // Remove the light when firing (other lights can be attached here too)

 FLMG B 0 Bright A_RemoveLight('FlamethrowerLight');

 ...

}

A_RemoveLight

bool A_RemoveLight (Name lightid)

Usage

Removes the dynamic light referenced by the specified identifier from the calling actor. Only dynamic lights attached by either A_AttachLight or A_AttachLightDef can be removed by this function, as they have the capability to assign an identifier to the attached light.

Parameters

lightid: the identifier for the light to remove.

Return value

(Need more info)

A_FreezeDeath

A_GenericFreezeDeath

(no parameters)

Starts the freeze sequence. This should only be used in the first state of the ice death sequence. A_GenericFreezeDeath additionally sets the ice palette translation and is useful if the sprites being used don't have the appropriate colors.

When used in conjunction with A_FreezeDeathChunks it is possible to recreate the frozen effect from Hexen where the frozen monster can be destroyed into chunks.

Examples

This would create the same freezing properties from Hexen:

```
actor FreezeDeathImp : DoomImp
{
    states
    {
        Ice:
            TROO V 5 A_FreezeDeath
            TROO V 1 A_FreezeDeathChunks
            wait
    }
}
```

A_FreezeDeathChunks

A_IceGuyDie

(no parameters)

Bursts the calling actor into chunks (IceChunk and for players also IceChunkHead) after a short random delay. The actor only bursts if it does not move. A_IceGuyDie immediately bursts the calling actor without any delay. The size box of the spawned shards will conform to the defined radius and height of the object.

If the calling actor has the BOSSDEATH flag, it also calls A_BossDeath.

Examples

This is an ice zombie, a frozen-solid zombie that continues to live. When killed, its death state triggers A_FreezeDeathChunks, bursting it into chunks of ice. A 100 frame delay was given to assure the zombie was not moving when this function was called (in case the corpse ends up sliding down a flight of stairs, off a ledge, etc.). A_NoBlocking was removed to prevent the monster from dropping 2 clips, since A_FreezeDeathChunks makes the monster drop its item upon death as well. Its XDeath state triggers A_IceGuyDie, causing it to instantly burst into chunks.

ACTOR IceZombie : ZombieMan

```
{
  Translation "Ice"
  States
  {
    Death:
      POSS H 5
      POSS I 5 A_Scream
      POSS J 5
      POSS K 5
      POSS L 100
      POSS L 10 A_FreezeDeathChunks
      Stop
    XDeath:
      POSS M 5 A_IceGuyDie
      Stop
  }
}
```

A_BasicAttack

A_BasicAttack (int meleedamage, string meleesound, string missiletype, float missileheight)

Parametrized version of A_ComboAttack. A_CustomComboAttack is more versatile.

Examples

Missile:

TROO EF 6 A_FaceTarget

TROO G 4 A_BasicAttack (5, "imp/melee", "TundraImpBall", 24)

goto See

A_BulletAttack

(no parameters)

Performs a hitscan attack. The amount of bullets fired is specified with the actor's Damage property. Additionally the attack sound is being played.

Examples

This is a shotgun zombie. It fires 5 bullets, according to its Damage property.

ACTOR ShotgunZombie : ZombieMan

```
{
  Damage 5
  States
  {
    Missile:
      POSS E 10 A_FaceTarget
      POSS F 8 A_BulletAttack
      POSS E 8
      Goto See
  }
}
```

A_CustomBulletAttack

void A_CustomBulletAttack (float horz_spread, float vert_spread, int numbullets, int damageperbullet [, string pufftype [, float range [, int flags [, int ptr [, string missile [, float Spawnheight [, float Spawnofs_xy]]]]]]]]))

Usage

A customizable hitscan attack for monsters. Fires a number of bullets with the specified damage and spread. The bullet puff and range can also be specified, as well as whether the actor uses its current target for aiming purposes.

Parameters

horz_spread: The horizontal spread, in degrees.

vert_spread: The vertical spread, in degrees.

numbullets: The number of bullets to fire.

damageperbullet: The amount of damage each bullet does. Unless the NORANDOM flag is set, this is multiplied by a random value between 1 and 3.

pufftype: The puff to spawn when a wall is hit. Default is "BulletPuff".

range: The maximum range of the bullets. Default is 0, which is interpreted as 2048.

flags:

CBAF_AIMFACING: If set, the attack will be fired in the direction the actor is currently facing, rather than at the actor's current target.

CBAF_NORANDOM: If set, the damage per bullet is not randomized.

CBAF_EXPLICITANGLE: If set, the horizontal and vertical spread are used as explicitly stated, instead of being used as a range for random spread.

CBAF_NOPITCH: If set, the vertical angle is not adjusted to aim at the target.

CBAF_NORANDOMPUFFZ: If set, the random z offset given to the puff when spawned is disabled.

CBAF_PUFFTARGET: Only works when missile is used. Sets the puff as the missile's target.

CBAF_PUFFMASTER: Only works when missile is used. Sets the puff as the missile's master.

CBAF_PUFFTRACER: Only works when missile is used. Sets the puff as the missile's tracer.

NOTE: The pointer flags will not work if the puff does not exist, i.e. spawning Blood instead of itself.

ptr: The actor to attack. This takes an actor pointer. Default is AAPTR_TARGET.

missile: The actor projectile to spawn. This actor faces the bullet puff and travels directly towards it. Default is none.

Spawnheight: Offsets how high up from the base of the actor missile spawns. Default is 32.

Spawnofs_xy: Offsets how far to the calling actor's right to spawn missile from (assuming one is viewing the actor from behind). Negative values spawn it to the left. Default is 0.

Examples

ACTOR Sniper : ShotgunGuy

```
{
  States
  {
    Missile:
      SPOS E 2 A_FaceTarget
      SPOS E 0 A_PlaySound("weapons/sshotf")
      SPOS F 3 Bright A_CustomBulletAttack(2, 2, 1, 20)
      SPOS E 5
      Goto See
  }
}
```

A_CustomComboAttack

A_CustomComboAttack (string missiletype, float spawnheight, int damage, string meleesound, string damagetype, bool bleed)

Usage

A customizable combo attack for monsters. The actor performs either a melee or missile attack depending on the distance to the target. A melee attack will be used if the target is within the actor's specified melee range. Otherwise, a projectile is generated instead.

Parameters

missiletype: The class name of the missile to spawn.

spawnheight: The height of the spawned missile.

damage: The melee damage dealt to the target.

meleesound: The sound played when the melee attack is used.

damagetype: The type of damage dealt if a melee attack is used. Defaults to "melee".

bleed: Controls whether the target bleeds on a successful hit by the melee attack. Defaults to true.

Examples

This example actually replicates the Baron of Hell's attack (A_BruisAttack) using this function:

Melee:

Missile:

```
BOSS EF 8 A_FaceTarget
```

```
BOSS G 8 A_CustomComboAttack("BaronBall", 32, 10 * random(1, 8), "baron/melee")
```

```
Goto See
```

A_CustomMeleeAttack

A_CustomMeleeAttack [(int damage [, str meleesound [, str missound [, str damagetype [, bool bleed]]]])]

Usage

A customizable melee attack for monsters. Causes the actor to execute a melee attack with the specified parameters.

The function calls A_FaceTarget and checks if the caller's target is within melee range, damaging it if it is and playing meleesound. If the target is outside melee range, however, missound sound is played and no damage is dealt to the target. If there is no target, the function does nothing when it is called.

Both meleesound and missound are played on the weapon channel (CHAN_WEAPON).

Parameters

damage: The amount of damage to perform. This can be an exact value or an expression. Default is 0.

meleesound: The sound to make when the melee successfully hits. Default is "", for no sound.

missound: The sound to make when the melee attack misses. Default is "", for no sound.

damagetype: The type of damage to do. Default is "Melee".

bleed: If true, target bleeds when hit. This only produces blood decals on nearby walls. Default is true.

Examples

This is the revenant's melee sequence:

Melee:

```
SKEL G 1 A_FaceTarget
SKEL G 6 A_SkelWhoosh
SKEL H 6 A_FaceTarget
SKEL I 6 A_SkelFist
goto See
```

The same sequence could be simulated this way:

Melee:

```
SKEL G 1 A_FaceTarget
SKEL G 6 A_SkelWhoosh
SKEL H 6 A_FaceTarget
SKEL I 6 A_CustomMeleeAttack(random(1, 10) * 6, "skeleton/melee") // Does not make any sound when missing
```

A_CustomRailgun

```
void A_CustomRailgun (int damage [, int spawnofs_xy [, color color1 [, color color2 [, int flags [, int aim [, double maxdiff [, class<Actor> pufftype [, double spread_xy [, double spread_z [, double range [, int duration [, double sparsity [, double driftspeed [, class<Actor> spawnclass [, double spawnofs_z [, int spiraloffset [, int limit [, double veleffect]]]]]]]]]]]]))
```

Usage

A customizable railgun attack for monsters.

Parameters

damage: The damage to inflict; this can be a fixed value or an expression.

`spawnofs_xy`: The horizontal offset (from the actor's center) where the railgun will emerge from. Negative values shift the beam to the actor's left, positive values shift it right. Default is 0.

color1: The color of the particles that form the spiral "ring" of the beam. This can be in the form of a RRGGBB string or a string holding a color defined in the X11R6RGB lump. If the string is invalid, the particles will be black. None (without quotes) in DECORATE or "" in ZScript will make the ring invisible. A value of 0, which is treated specially, draws the the particles in one of four shades of blue, picked randomly. Default is 0.

color2: The color of the particles that form the central "core" of the beam. This can be in the form of a RRGGBB string or a string holding a color defined in the X11R6RGB lump. If the string is invalid, the particles will be black. None (without quotes) in DECORATE or "" in ZScript will make the core invisible. A value of 0, which is treated specially, draws the particles in one of three shades of gray, picked randomly. Default is 0.

flags: The following flags can be combined by using the | character between the constant names:

RGF_SILENT — Silent: The railgun will not play an attack sound when firing.

RGF_NOPIERCING — Not piercing: The railgun will stop at the first enemy hit, rather than passing through.

RGF_EXPLICITANGLE — Explicit angle: The spread parameters are taken as explicit angles rather than maximum random amplitude.

RGF_FULLBRIGHT — Full bright: Rail particles will be rendered at maximum brightness, ignoring sector lighting.

RGF_CENTERZ — Center z: Prevent the railgun from using the actor's attack z-offset (Player.AttackZOffset for players or a hard-coded +8 for everything else).

RGF_NORANDOMPUFFZ — No random puff Z: Disables the random z-offset of spawned puffs.

aim determines which aiming mode to use:

0: The monster shoots in the direction it is looking (default).

1: The monster aims at its target.

2: The monster aims at and takes the target's horizontal velocity into account.

maxdiff: This is used to make the rail more jagged, or lightning-like, with higher numbers. Default is 0 (straight).

pufftype: The puff actor to use. By default, the puff will only spawn in rare circumstances (e.g. when hitting a dormant monster) unless the puff actor has the ALWAYSPUFF flag set. Even if not shown, the selected puff will still be used for applying custom damagetypes and other properties. Default is "BulletPuff".

`spread_xy`: Maximum angle of random horizontal spread. Default is 0.

`spread_z`: Maximum angle of random vertical spread. Default is 0.

range: Maximum distance (in map units, as fixed-point) the rail shot will travel before vanishing. Default is 0, which uses the default value of 8192 as the range.

duration: Lifetime of spawned particles, in tics. Default is 0, which uses the default value of 35 as the duration.

sparsity: Distance between individual particles. Implemented as a multiplier. Default is 1.0.

driftspeed: Speed at which particles "drift" away from their initial spawn point. Implemented as a multiplier. Default is 1.0.

spawnclass: Actor to spawn in place of trail particles. If non-null, the specified actor will be spaced sparsity units apart instead of the usual trail. It will also inherit the pitch of the shooter and track the owner, allowing for explosive trails to not hurt the owner. Particle-specific properties such as duration, driftspeed, and rail color are ignored in such a case. Default is "None".

spawnofs_z: The vertical offset (from the actor's center) where the railgun will emerge from. Negative values

shift the beam down, positive values shift it up. Default is 0.

spiraloffset: the angle from which the outer ring starts spiraling. Default is 270.

limit: Sets the maximum number of actors to pierce through, if they are applicable for damaging. Default is 0 (no limit is set).

veleffect: Customizes the inaccuracy induced by a moving target. Default is 3.

Examples

This arachnotron uses a rail attack which deals 65 points of damage and has a white core and a blue ring.

ACTOR RailArachnotron : Arachnotron

```
{
  States
  {
    Missile:
      BSPI A 17 Bright A_FaceTarget
      BSPI G 4 Bright A_CustomRailgun(65, 0, "Blue", "White") // Equivalent to A_CustomRailgun(65, 0,
"0000FF", "FFFFFF")
      BSPI H 4 Bright
      Goto See
    }
  }
}
```

A_Detonate

(no parameters)

Performs an explosive (radius) attack. The amount of damage and the attack's radius are specified with the actor's Damage property.

Examples

This is a gamma demon. It releases a wave of damaging radiation when it dies. When killed, its death state triggers A_Detonate, which according to its Damage property, releases a wave of radius damage within 100 units, dealing 100 damage at the center of the blast.

ACTOR GammaDemon : Demon

```
{
  Damage 100
  States
  {
    Death:
      SARG I 8
      SARG J 8 A_Scream
      SARG K 4 A_Detonate
      SARG L 4 A_NoBlocking
      SARG M 4
      SARG N -1
      Stop
  }
}
```

A_Explode

```
int A_Explode [(int damage [, int distance [, int flags [, bool alert [, int fulldamagedistance [, int nails [, int naildamage [, class<Actor> pufftype [, name damagetype]]]]]]]]]]]
```

Usage

Performs an explosive (radius) attack.

Parameters

damage: How much damage is inflicted at the center of the explosion. If this is set to a value that is less than 0, the ExplosionDamage property is used for the damage. Default is -1.

distance: The area covered by the damage (damage inflicted drop linearly with distance). Note that a radius larger than 32767 extends beyond ZDoom's map space limitations, and will overflow with undesired results (such as becoming too small and damaging nothing). If damage is less than 0, the ExplosionRadius property is used for the radius. A radius value of 0 or less uses the same value which damage uses. Default is -1.

flags: Allows the alteration of the function's behavior. This parameter is ignored in favor of using the DontHurtShooter property if damage is less than 0. The following flags can be combined by using the | character between the constant names:

XF_HURTSOURCE — Hurts the source: if set, the source can be damaged by the explosion. Note that the source is not necessarily the calling actor. This flag is set by default. By using other flags or, if none of the other flags is desired, passing 0 to flags i.e. using 0 in place of the flag, this flag can be cleared.

XF_NOTMISSILE — Not a missile: if set, the calling actor is considered to be the source. By default, the calling actor is assumed to be a projectile, and the source is therefore considered to be the calling actor's target.

XF_EXPLICITDAMAGETYPE — The damagetype parameter will never change to the actor's damage type.

XF_NOSPLASH — No splash: if set, the explosion does not create any terrain splashes.

XF_THRUSTZ — Apply vertical thrust: if set, the attack pushes the victim vertically, in addition to horizontally. Normally, vertical thrust is applied without the need of this flag, but it could be disabled by setting the compat_explode1 compatibility flag. This flag overrides the compatibility flag.

alert: Whether or not the explosion rings the alarm. This parameter is always set to false if damage is less than 0, regardless of what is passed to it. Default is false.

fulldamagedistance: The area within which full damage is inflicted. Default is 0.

nails: The number of horizontal hitscan attacks performed in a ring, originating from the actor's center. A value of 30 emulates the A_NailBomb codepointer from SMMU, while still allowing to modify all other parameters from A_Explode. Default is 0.

naildamage: The amount of damage inflicted by the nail attack, if any. Default is 10.

pufftype: The name of the puff to use for the nail attack. If nothing is supplied, BulletPuff is used. Default is "BulletPuff".

damagetype: The damage type to use for the damage of this function rather than the actor's own damage type. If left as is without XF_EXPLICITDAMAGETYPE, will use the actor's damage type instead. Default is 'None'.

Return value

Returns the number of actors damaged. Actors that absorb the damage completely are not counted.

Examples

actor MyGrenade : Grenade

{

Speed 17

BounceFactor 0.4

BounceCount 6

States

{

Death:

QEX1 A 0 A_PlaySound("Weapon/GenericExplode", CHAN_WEAPON)

QEX1 A 5 A_Explode(100, 128)


```
        QEX1 BCDE 5
      Stop
    }
  }
```

Whenever this rocket explodes, it logs the number of actors which got damaged by its splash effect.

```
actor SmartRocket : Rocket
{
  States
  {
    Death:
      MISL B 8 Bright A_LogInt(A_Explode)
      Goto Super::Death+1
  }
}
```

A_MonsterRail

(no parameters)

Performs a railgun attack. The amount of damage this attack inflicts is specified with the calling actor's Damage property.

Examples

This is a railgun zombie. It fires a railgun shot that deals 15 damage. The attack sound was edited to give a railgun sound effect to the monster when fired, instead of the zombie's default pistol shot sound.

ACTOR RailgunZombie : ZombieMan

```
{
  Damage 15
  AttackSound "weapons/rbeam"
  States
  {
    Missile:
      POSS E 10 A_FaceTarget
      POSS F 8 A_MonsterRail
      POSS E 8
      Goto See
  }
}
```

A_MonsterRefire

state A_MonsterRefire (int chancecontinue, str "abortstate")

A_CPosRefire

(no parameters)

A_CrusaderRefire

(no parameters)

A_SentinelRefire

(no parameters)

A_SpidRefire

(no parameters)

Calls A_FaceTarget and then checks whether the monster should abort its attack sequence and go back to abortstate. If the target is out of sight or dead, has a chancecontinue/256 chance to not jump to the abort state.

The monster-specific functions use the following parameters:

A_CPosRefire: 40, "See"

A_CrusaderRefire: 0, "See"

A_SpidRefire: 10, "See"

A_SentinelRefire: 30, "See"

All these functions jump to the "See" state if the attack is to be aborted. The loop has to be explicitly coded in the actor definition. A_SentinelRefire also has a 10/256 chance of aborting the attack even if the target is still in sight.

A_CrusaderRefire is restricted to Crusader and derived classes.

Example

ACTOR SuperZombie : ZombieMan replaces ZombieMan

```
{
    States
    {
        Missile:
            POSS E 10 A_FaceTarget
            POSS FE 2 Bright A_PosAttack
            POSS F 1 A_MonsterRefire(130, "See") // About 50% chance to jump to "See" if target is out of sight.
            Goto Missile+1 // Looping back to the attack state to allow the actual refire!
    }
}
```

A_RadiusDamageSelf

```
void A_RadiusDamageSelf [(int damage [, double distance [, int flags [, class<Actor> flashtype]]]])]
```

Usage

Performs an explosive (radius) attack, much like A_Explode, that damages the calling actor's target only, which normally for projectiles, is their shooter.

While this function can be used by any actor, its usage should generally be restricted to projectiles only.

Parameters

damage: how much damage is inflicted at the center of the explosion. Default is 128.

distance: the area covered by the damage (damage inflicted drops linearly with distance). Default is 128.

flags: customizes the behavior of the function:

RDSF_BFGDAMAGE use the BFG9000 tracers' formula for damage. This works by picking and adding a random value from 1 to 8 a number of times which equals to the calculated damage over distance.

Default is 0.

flashtype: the actor to spawn at the target's position when damage is inflicted. This actor may have the PUFFGETSOWNER, FOILINVUL and FOILBUDDHA flags. In addition, its DamageType property is used as the damage type of the attack. If this actor is not provided, the function defaults to BFGSplash as the damage type to inflict. Default is null(ZScript only)/"None"(DECORATE only).

Examples

```
class OddRocket : Rocket
{
    States
    {
        Death:
            MISL B 8 Bright A_RadiusDamageSelf(72, 192.0, RDSF_BFGDAMAGE, "BFGExtra");
            MISL C 6 Bright;
            MISL D 4 Bright;
            Stop;
    }
}
```

A_RadiusThrust

`void A_RadiusThrust [(int force [, int distance [, int flags [, int fullthrustdistance [, name species]]]])]`

Usage

Makes the calling actor thrust away actors nearby from it. This is similar to A_Explode but without the damage. Although, affected actors could take damage if they collide with a wall or each other, akin to A_Blast.

Parameters

force: how powerful the blast is. The velocity of the blast is determined as: $\text{force} / (2 * \text{mass})$, so a force of 40000 pushes away an actor of 1000 mass (the mass of a baron of hell) with a velocity of 20 units/tic (the speed of a rocket) at the very center of the blast. Negative values push actors towards the center of the source. Default is 128.

distance: how far the blast extends. At the center, actors take the full force of the blast. At the outer edge, this many units away, actors are not pushed at all. The value passed to force is used to determine the distance if this parameter is 0 or less. Default is -1.

flags: the following flags can be combined by using the | character between the constant names:

RTF_AFFECTSOURCE — Affect source: if this flag is set, the shooter of the projectile is affected. This is set by default.

RTF_NOIMPACTDAMAGE — No impact damage: if this flag is set, actors thrust away do not cause melee damage on impact.

RTF_NOTMISSILE — Not a missile: if set, the calling actor is considered to be the source. By default, the calling actor is assumed to be a projectile, and the source is therefore considered to be the calling actor's target.

RTF_THRUSTZ — Apply thrust to vertical velocity as well as horizontal. By default, the function does not apply Z velocity at all. This flag overrides the compat_explode1 compatibility flag.

fullthrustdistance: the radius that actors will be subject to the full force of the blast. Default is 0.

species: the actor species to thrust. Only actors whose species matches this are thrust. Default is "None".

Examples

The following example shows a projectile that blows objects away with a force of 500 within a 64 unit radius affecting the shooter:

```
class WindyProjectile : Actor
```

```
{
    Default
    {
        Height 4;
        Radius 2;
        Projectile;
    }

    States
    {
        Spawn:
            WIND ABCDEFGHIJK 2 A_RadiusThrust(500, 64, RTF_AFFECTSOURCE);
            WIND LMNO 2;
            Stop;
    }
}
```

A_SpawnProjectile

Actor A_SpawnProjectile (class<Actor> missiletype [, double spawnheight [, double spawnofs_xy [, double angle [, int flags [, double pitch [, int ptr]]]]]])

Usage

A customizable projectile attack for non-player actors, typically used by monsters to launch a projectile at their target.

This serves as a replacement for A_CustomMissile which has a pitch miscalculation in it.

Parameters

missiletype: The class name of the projectile to fire.

spawnheight: Raises the projectile spawn point on the actor by this amount in units. Default is 32.

spawnofs_xy: Moves the projectile spawn point to the right if positive, left if negative. Default is 0.

angle: Adds this much offset to the actor's angle, which is aimed at the target first. Default is 0.

flags: Customizes the behavior of the function. Multiple flags can be combined by using the bitwise OR operator (|) between the constant names:

CMF_AIMOFFSET — Aim mode 2; the projectile is aimed parallel to a projectile with a spawn height of 32 and an xy-offset of 0. This can be useful if you want to have an actor shoot multiple projectiles at once. This aim mode requires a target to be present in order to spawn the projectile.

CMF_AIMDIRECTION — Aim mode 3; the projectile is not aimed at a target. Instead, it is shot in the specified direction with the specified pitch. This implies the CMF_ABSOLUTEPIITCH flag. This aim mode does not require a target to be present in order to spawn the projectile.

If neither of the above is used, then the function defaults to aim mode 1. With this mode, the projectile is aimed directly at the target. This aim mode requires a target to be present in order to spawn the projectile.

CMF_TRACKOWNER — If a projectile is fired by another projectile, this flag can be used to ensure the secondary projectile knows who its real owner is. This behavior would have been the default, but it was not in the 2.0.x branch of ZDoom, allowing erroneous behavior which has been used by some mods, so fixing this would break compatibility.

CMF_CHECKTARGETDEAD — If the target is not present and the chosen aim mode is not 3, no projectile is spawned. In such a case, this flag makes the calling actor abort its attack sequence by entering the See state, if it exists. Unlike other actor types, monsters need their health to be above 0 in order to successfully enter the state.

CMF_ABSOLUTEPIITCH — Use the pitch parameter as an absolute value, ignoring the calculated aim pitch. This is implied by the CMF_AIMDIRECTION flag. The main difference here is that a target is required, where as the CMF_AIMDIRECTION flag does not require a target.

CMF_OFFSETPIITCH — Use the pitch parameter as an offset to the calculated aim pitch.

CMF_SAVEPIITCH — The pitch used for firing the projectile is saved as the projectile's own pitch (requires the CMF_AIMDIRECTION, CMF_ABSOLUTEPIITCH or CMF_OFFSETPIITCH flag).

CMF_ABSOLUTEANGLE — Use the angle parameter as an absolute value, ignoring the calculated aim angle. The calling actor's angle is still factored in, however.

Default is 0.

pitch: Offsets the projectile's aim vertically by this amount. Positive values aim down, while negative values aim up. This is only used if at least one of the flags CMF_AIMDIRECTION, CMF_ABSOLUTEPIITCH and CMF_OFFSETPIITCH is used. Default is 0.

ptr: The actor to fire the projectile at. This takes an actor pointer. Default is AAPTR_TARGET.

Return value

A pointer to the projectile if the spawn is successful, otherwise the return value is null.

Examples

Missile:

```
POSS E 10 A_FaceTarget
```

```
POSS F 8 A_SpawnProjectile("NormalBullet", 48)
```

```
POSS E 8
```

```
Goto See
```


A_ThrowGrenade

bool A_ThrowGrenade (string spawntype [float zheight [, float xyvel [, float zvel [, bool useammo]]]])

Spawns a projectile and throws it like a grenade. This function can be used for monsters, inventory items and weapons. useammo only has an effect when used with a weapon.

The parameters are:

spawntype: Type of the projectile to be spawned.

zheight: Height at which the projectile is spawned. The function adds 35 to this, so that passing 0 means at the center of the throwing actor.

xyvel, zvel: Speed with which the object is thrown.

Examples

This is a grenade-throwing zombie. The attack sound was changed so that it would not use the default pistol firing sound for throwing a grenade.

ACTOR GrenadierZombie : ZombieMan

```
{
  AttackSound ""
  States
  {
    Missile:
      POSS E 10 A_FaceTarget
      POSS F 8 A_ThrowGrenade("Grenade", 20, 8, 4)
      POSS E 8
      Goto See
  }
}
```


A_WolfAttack

`A_WolfAttack (int flags[, sound whattoplay[, float snipe[, int maxdamage[, int blocksize[, int pointblank[, int longrange[, float runspeed, [class pufftype]]]]]]]])`

This codepointer emulates the behavior of the enemy guns in Wolfenstein 3D. It diverges greatly from normal hitscan attacks on several points:

There is no puff (unless the flag `WAF_USEPUFF` is given).

There is also no damage thrust.

Distance computation is orthogonal, like explosions. (So a target at `[128, 127]` relatively to the shooter will not be further away than a target at `[128, 0]`, no matter what trigonometry says.)

The attack is less likely to be successful at longer range.

The attack is less likely to be successful if the attacker is in front of the target, as the target will be able to "dodge" the incoming bullet.

The attack is less likely to be successful if the target is a player running.

Damage is decreased with range, too: halved at medium range, and halved again (so, quartered) at long range. This can result in 0 damage.

The parameters are all optional. They consist of:

flags can be set to 0 for default behavior, or a combination of the following flags:

`WAF_NORANDOM` — the damage parameter is used as an absolute value instead of the upper bound on a random roll.

`WAF_USEPUFF` — spawns a puff as normal Doom engine behavior. It will be used for damage type.

`whattoplay` is the attack sound. It defaults to "weapons/pistol".

`snipe` corresponds to the factor by which the effective distance is multiplied. The lower it is, the more precise the enemy is. Hans Grosse and the blue SS in Wolf 3D have a snipe factor of $2/3$. It defaults to 1.0.

`maxdamage` corresponds (unless the `WAF_NORANDOM` flag is used) to the theoretical maximum damage inflicted at point blank range, and is used as a modulo for the random number generator. The default value of 64 corresponds to Wolf's double-bitshift ($255 \gg 2 == 63$; $255 \% 64 == 63$).

`blocksize` corresponds to the size of a "block" emulating the Wolf 3D squares. The default value is 128, corresponding to the translation of the Wolf levels done by Id Software in the secret levels of Doom II. This value is used by the next two parameters.

`pointblank` corresponds to the number of squares below which the damage isn't divided.

`longrange` corresponds to the number of squares beyond which the damage is further divided.

`runspeed` is the combined velocity at which the target must move if it wants to be harder to aim. The default value of 160.0 is roughly based on the Doom player walk and run speeds. This is compared to the target's $velx^2 + vely^2 + velz^2$. Note that because monster AI movements work without using velocities, this only has an effect against players (and theoretically, missiles).

`pufftype` is only meaningful if the `WAF_USEPUFF` flag is used. The puff specified will be used for damage type and other effects. The default puff is `BulletPuff`.

A_DropInventory

A_DropInventory (string itemtype [, int amount])

Usage

Drops an item from the caller's inventory to the ground.

Parameters

itemtype: the desired item to drop. The item must be present in the caller's inventory in order to be dropped.

amount: specifies the number of samples the dropped item contains. The function cannot drop more than what is available in the inventory. Default is -1, which means one sample.

A_DropItem

Actor A_DropItem (class<Actor> item [, int dropamount [, int chance]])

Usage

The calling actor drops the specified item. This works in a similar way to the DropItem actor property.

Parameters

item: The item to drop. This can be any valid actor class, not just inventory.

dropamount: The inventory amount the dropped item contains. This is only meaningful with actors inheriting from the Inventory class. If dropamount is greater than 0, the amount of inventory gained from picking up the item equals exactly to that number. Otherwise, the amount gotten is the same as the amount defined by the item itself, except for ammo items, which in this case, the amount is determined by the current skill level's DropAmmoFactor. Default is -1.

chance: The probability of the drop. The item is never dropped if this is -1 or less, while it is always dropped if this is 255 or greater. Default is 256.

Return value

Returns a pointer to item if it was dropped successfully, otherwise it returns null.

A_GiveInventory

bool A_GiveInventory (string type [, int amount [, pointer giveto]])

Usage

Adds amount items of type type to the calling actor's inventory. This function will not add more items than can be carried. This is only really useful for weapons and custom inventory.

If type is an item derived from Health, then the amount received is the result of multiplying amount by the item's amount, e.g. if type is Medikit and amount is 2, then the amount of health received would be 50 points, not 2.

Parameters

type: the item to give. This should be a valid inventory item.

amount: the number of samples of this item to give. Default is 0, which is interpreted as 1.

giveto: the actor to give the item to. This is an actor pointer selector. Default is AAPTR_DEFAULT, which corresponds to the caller of the function.

Return value

The function returns true if the item is successfully received, otherwise it returns false.

Examples

Actor BigBoost : CustomInventory 10492

```
{
  Inventory.PickupMessage "Energy Boost!!!"
  Inventory.PickupSound "misc/p_pkup"
+COUNTITEM
States
{
  Spawn:
    AWI3 A -1
    Stop
  Pickup:
    TNT1 A 0 A_GiveInventory ("Soulsphere", 2)
    TNT1 A 0 A_GiveInventory ("BFG9000")
    Stop
}
```

This actor uses the CustomInventory's special behavior to give two inventory items with one pickup using A_GiveInventory.

A_GiveToChildren

int A_GiveToChildren (string type [, int count])

Usage

Adds count items of type type to the calling actor's children's inventory. This function will not add more items than can be carried.

An actor's children are actors which are spawned by said actor through A_SpawnItemEx with the SXF_SETMASTER flag passed to the function, or by pointer manipulation with functions like A_RearrangePointers or A_TransferPointer.

If type is an item derived from Health, then the amount received is the result of multiplying count by the item's amount, e.g. if type is Medikit and count is 2, then the amount of health received would be 50 points, not 2.

Parameters

type: the item to give. This should be a valid inventory item.

count: the number of samples of this item to give. Default is 0, which is interpreted as 1.

Return value

The function returns the total number of actors which successfully received the item.

A_GiveToSiblings

int A_GiveToSiblings (string type [, int count])

Usage

Adds count items of type type to the calling actor's siblings' inventory. This function will not add more items than can be carried.

An actor's siblings are actors which share a common master actor with said actor. These siblings are considered the children of the master. This relationship is formed when those children actors are spawned by the master actor through A_SpawnItemEx with the SXF_SETMASTER flag passed to the function, or if a child spawns another child with SXF_TRANSFERPOINTERS without the SXF_SETMASTER flag. This is needed because SXF_SETMASTER will otherwise cause the child of the child to be its child instead of a sibling, as the calling actor with that flag makes it the master instead of its own master.

If type is an item derived from Health, then the amount received is the result of multiplying count by the item's amount, e.g. if type is Medikit and count is 2, then the amount of health received would be 50 points, not 2.

Parameters

type: the item to give. This should be a valid inventory item.

count: the number of samples of this item to give. Default is 0, which is interpreted as 1.

Return value

The function returns the total number of actors which successfully received the item.

A_GiveToTarget

bool A_GiveToTarget (string type [, int amount [, pointer forward_ptr]])

Usage

Tries to add count items of type type to the calling actor's current target's inventory. This function will not add more items than can be carried. Note that when an actor dies its target is automatically set to the killer. Projectiles also target the actor which shot them when they are fired.

If type is an item derived from Health, then the amount received is the result of multiplying count by the item's amount, e.g. if type is Medikit and count is 2, then the amount of health received would be 50 points, not 2.

Parameters

type: the item to give. This should be a valid inventory item.

amount: the number of samples of this item to give. Default is 0, which is interpreted as 1.

forward_ptr: the actor to give the item to, with the calling actor's target being the context, here, as opposed to the caller itself. This is an actor pointer selector. Default is AAPTR_DEFAULT, which corresponds to the calling actor's target.

Return value

The function returns true if the item is successfully received, otherwise it returns false.

Examples

This function can be used to reward players with items for killing certain monsters.

Death:

```
CYBR H 10 A_GiveToTarget("Credits", 1000)
CYBR I 10 A_Scream
CYBR JKL 10
CYBR M 10 A_NoBlocking
CYBR NO 10
CYBR P 30
CYBR P -1 A_BossDeath
stop
```

If this imp dies, it gives a partial invisibility sphere to its target's (killer's) master.

ACTOR BlurSphereDoomImp : DoomImp

```
{
    States
    {
        Death:
            TROO A 0 A_GiveToTarget("BlurSphere", 1, AAPTR_MASTER)
            Goto Super::Death

        XDeath:
            TROO A 0 A_GiveToTarget("BlurSphere", 1, AAPTR_MASTER)
            Goto Super::XDeath
    }
}
```

A_RadiusGive

int A_RadiusGive (str item, float distance, int flags [, int amount [, str filter [, str species [, float mindist [, int limit]]]])

Usage

Gives an item to all eligible actors within range. Note that the receiving actor's center point must be within the radius specified by the distance parameter, otherwise it won't receive the item.

Note that for very large distances, attempting to give to specific things map-wide may fail. A good range for the entirety of the map would be 16383. A dummy actor can warp to (0,0,0) coordinates to ensure the radius does not overflow from being too close to the map boundaries. Experimentation may be required.

Since A_RadiusGive returns an int, CustomInventory actors can set success or failure of an item's reception. This allows for accurate measuring of actors having successfully received the item or not.

Parameters

item is the item to give.

distance is the radius of range. A value of 0 or less causes the function to do nothing, and does not imply infinite range.

flags determines which actors are eligible of getting the item:

RGF_GIVESELF: The calling actor is eligible.

Note: This always succeed at giving the calling actor the item if specified, and is not subject to filtering, nor types of eligible actors.

RGF_PLAYERS: Any player actor is eligible.

RGF_MONSTERS: Any monster, be it friend or foe, is eligible.

RGF_OBJECTS: Any shootable or vulnerable object is eligible. This does not include players and monsters.

RGF_VOODOO: Any voodoo doll is eligible.

RGF_CORPSES: Any corpse is eligible.

RGF_MISSILES: Missile actors are eligible.

RGF_ITEMS: Inventory items are eligible.

RGF_KILLED: Monsters which are truly dead may receive the item. Killed enemies may not always have the corpse flag, and corpse flagged monster may not always be killed (such as the DONTCORPSE flag). This should be kept in mind when using this flag for things like body removal, as they may not have time to drop their items or execute specials. This may prevent maps like Doom II's MAP07 from working their specials, so caution is advised.

Note: At least one of the above flags must be specified for the giving to have any effect.

RGF_NOTARGET: The calling actor's target may not get the item.

RGF_NOTRACER: The calling actor's tracer may not get the item.

RGF_NOMASTER: The calling actor's master may not get the item.

RGF_INCLUSIVE: By default, if a same actor occupies more than one of the calling actor's pointer fields, such as target and tracer, specifying either RGF_NOTARGET or RGF_NOTRACER is enough to prevent them from receiving the item. This flag changes the behavior around to be inverse, which would require both RGF_NOTARGET and RGF_NOTRACER to prevent them from getting the item. This flag is only useful with

RGF_NO(TARGET/MASTER/TRACER).

RGF_EXFILTER: Inverts the actor filter from a whitelisted actor to a blacklist; the actor type specified will not receive the item, while everyone else will.

RGF_EXSPECIES: Inverts the species filter from a whitelisted species to a blacklist; the species specified will not receive the item, while all other species will.

RGF_CUBE: Use a cube for the range check rather than a circle.

RGF_NOSIGHT: The actor is given the item regardless of whether it is in the line of sight of the caller or not.

amount is how much of this item will be given to actors. If item is a health item, the amount of health to be given is the health item's amount multiplied by this parameter. Default is 0 (which is interpreted as 1).

filter determines which actors may receive this item.

species determines which particular species may receive this item.

mindist actor must be this far away to receive the item. This must be less than distance. Default is 0.

limit will stop the function from performing after successfully delivering the item this amount of times to actors within range. The actors which receive the items are random -- they are not found closest or furthest away from the calling actor.

Return value

The function returns the total number of actors which successfully received the item.

Examples

The following example shows a projectile that gives health to any ally players that come near to it. Could be useful in cooperative.

ACTOR HealingPlasma : PlasmaBall

```
{
+RIPPER
Damage (0)
Translation "192:207=168:176", "240:247=177:184" // blue -> red
States
{
Spawn:
    PLSS AAB 3 Bright A_RadiusGive("Health", 96, RGF_PLAYERS, 5)
    Loop
}
```

This barrel gives any player standing in its vicinity one point of energy cells every six tics. However, the barrel has a limited amount to give, and once it runs out of cells to give, it harmlessly explodes and disappears.

ACTOR CellsBarrel : ExplosiveBarrel

```
{
    var int user_cellinstock; // See user variables

    States
    {
    Spawn:
        BAR1 A 6 NoDelay A_SetUserVar("user_cellinstock", 80)
        BAR1 B 6

    GiveCells:
        BAR1 AB 6
        { // See anonymous functions
            A_SetUserVar("user_cellinstock", user_cellinstock - A_RadiusGive("Cell", 128.0, RGF_PLAYERS));

            If(user_cellinstock <= 0)
            {
                return state("Explode");
            }

            return state("");
        }
        Loop

    Explode:
        BEXP A 5 Bright
        BEXP B 5 Bright A_Scream
        BEXP CD 5 Bright
```

BEXP E 10 Bright
Stop

}
}

A_SelectWeapon

bool A_SelectWeapon (class<Weapon> whichweapon [, int flags])

Usage

Selects the specified weapon type as the calling player's current weapon.

Parameters

whichweapon: the class name of the weapon to switch to. Unless the SWF_SELECTPRIORITY flag is passed, this has to be a valid weapon and the weapon must be present in the player's inventory for the switch to happen. The function will be carried out whether or not the weapon to switch to has ammo.

flags: Default is 0.

SWF_SELECTPRIORITY â€” if the specified weapon is either invalid or does not exist in the inventory and this flag is set, the function switches to the highest priority weapon the player has, in the same vein as running out of ammo.

Return value

The function returns true if the weapon switch happens, otherwise it returns false.

Examples

This berserk pack uses A_SelectWeapon to change to a new unarmed attack which replaces the fist.

```
actor NewBerserk : Berserk
{
    States
    {
        Pickup:
            TNT1 A 0 A_GiveInventory("PowerStrength")
            TNT1 A 0 HealThing(100)
            TNT1 A 0 A_SelectWeapon("KoolFist")
            Stop
    }
}
```

This rather odd item, when picked up, tries to switch the player's weapon to the BFG9000. If the weapon exists in the player's inventory, the switch happens and the player receives 40 points of energy cells. If, however, the switch fails, i.e the player does not have the weapon, the player gets damage by half of his or her health.

```
actor OddArtifact : Megasphere
{
    Inventory.PickupMessage "Oddsphere!"

    States
    {
        Pickup:
            TNT1 A 0
            { // See anonymous functions.
                if (A_SelectWeapon("BFG9000"))
                {
                    A_GiveInventory("Cell", 40);
                }
                else
                {
                    A_DamageSelf(health / 2);
                }
            }
    }
}
```

```
}  
  Stop  
}  
}
```

A_SetInventory

bool A_SetInventory (string type, int count [, pointer ptr [, bool beyondMax]])

Usage

Behaves similarly to A_GiveInventory with a few key exceptions.

The amount is specifically set, not added or subtracted. As such, setting it to 0 will remove the item completely. As such, ignores health properties and skill levels. Use A_GiveInventory for standard giving.

Parameters

type: the item to give. This should be a valid inventory item.

count: the amount to set.

ptr: the actor to give the item to. This is an actor pointer selector. Default is AAPTR_DEFAULT, which corresponds to the caller of the function.

beyondMax: Unlike A_GiveInventory, this function respects the MaxAmount property at all times. Setting this to true will ignore the maximum value. Default is false.

Return value

The function returns true if the actor has their inventory changed. Thus, if they already have the amount specified, it returns false.

A_TakeFromChildren

int A_TakeFromChildren (string type [, int count])

Usage

Removes count items of type type from the calling actor's children's inventory. The minimum amount of item of a type in an inventory is zero, removing a greater amount than what a child actually possesses will not result in a negative amount.

An actor's children are actors which are spawned by said actor through A_SpawnItemEx with the SXF_SETMASTER flag passed to the function.

Parameters

type: the item to take. This should be a valid inventory item.

count: the number of samples of this item to take. If this is 0 or a value which is equal to or greater than the number of samples in the inventory, the item is cleared from the inventory unless it has the INVENTORY.KEEPDEPLETED flag set, and in which case, its amount is merely reduced to 0. Default is 0.

Return value

The function returns the total number of child actors which have one or more samples of type (before attempting removal) in their inventory.

A_TakeFromSiblings

int A_TakeFromSiblings (string type [, int count])

Usage

Removes count items of type type from the calling actor's siblings' inventory. The minimum amount of item of a type in an inventory is zero, removing a greater amount than what a sibling actually possesses will not result in a negative amount.

An actor's siblings are actors which share a common master actor with said actor. These siblings are considered the children of the master. This relationship is formed when those children actors are spawned by the master actor through A_SpawnItemEx with the SXF_SETMASTER flag passed to the function.

Parameters

type: the item to take. This should be a valid inventory item.

count: the number of samples of this item to take. If this is 0 or a value which is equal to or greater than the number of samples in the inventory, the item is cleared from the inventory unless it has the INVENTORY.KEEPDEPLETED flag set, and in which case, its amount is merely reduced to 0. Default is 0.

Return value

The function returns the total number of sibling actors which have one or more samples of type (before attempting removal) in their inventory.

A_TakeFromTarget

bool A_TakeFromTarget (string type [, int amount [, int flags [, pointer forward_ptr]]])

Usage

Removes count items of type type from the inventory of the actor's current target. The minimum amount of item of a type in an inventory is zero, removing a greater amount than what the target actually possesses will not result in a negative amount.

Parameters

type: The inventory item to take.

amount: The amount to take. If this is 0 or a value which is equal to or greater than the number of samples in the inventory, the item is cleared from the inventory unless it has the INVENTORY.KEEPDEPLETED flag set, and in which case, its amount is merely reduced to 0. Default is 0.

flags: There is at the moment only one flag:

TIF_NOTAKEINFINITE — If this flag is set, nothing is taken if the type is an Ammo and the player benefits from infinite ammo (either from a powerup or a cheat).

forward_ptr: The actor to take the item from. This is an actor pointer, which can be any of the following:

AAPTR_DEFAULT — The calling actor's target (default value).

AAPTR_NULL — No actor at all.

AAPTR_TARGET — The target of the calling actor's target, if any.

AAPTR_MASTER — The master of the calling actor's target, if any.

AAPTR_TRACER — The tracer of the calling actor's target, if any.

Remember that the nature of these pointers depend on the actor type and is not always intuitive.

Return value

The function returns true if the number of samples of type in the pointed-to actor's inventory (before attempting removal) is greater than zero, otherwise it returns false.

Examples

This demon tries to run up to you and steal your ammo!

ACTOR BanditDemon : Demon

```
{
    States
    {
        Melee:
            SARG E 8 A_FaceTarget
            SARG E 0 A_TakeFromTarget("Clip", 10)
            SARG E 0 A_TakeFromTarget("Shell", 4)
            SARG E 0 A_TakeFromTarget("RocketAmmo", 1)
            SARG E 0 A_TakeFromTarget("Cell", 20)
            SARG E 18 A_ChangeFlag("FRIGHTENED", TRUE) // Demon makes a run for it.
            Goto See
    }
}
```


A_TakeInventory

bool A_TakeInventory (string type [, int amount [, int flags [, pointer giveto]]])

Usage

Removes amount items of type type from the calling or pointed to actor's inventory. The minimum amount of item of a type in an inventory is zero, removing a greater amount than what the actor actually possesses will not result in a negative amount.

Parameters

type: The inventory item to take.

amount: The amount to take. If this is 0 or a value which is equal to or greater than the number of samples in the inventory, the item is cleared from the inventory unless it has the INVENTORY.KEEPDEPLETED flag set, and in which case, its amount is merely reduced to 0. Default is 0.

flags: There is at the moment only one flag:

TIF_NOTAKEINFINITE — If this flag is set, nothing is taken if the type is an Ammo and the player benefits from infinite ammo (either from a powerup or a cheat).

giveto: The actor to take the item from. This is an actor pointer, which can be any of the following:

AAPTR_DEFAULT — The calling actor itself (default value).

AAPTR_NULL — No actor at all.

AAPTR_TARGET — The calling actor's target, if any (equivalent to using A_TakeFromTarget).

AAPTR_MASTER — The calling actor's master, if any.

AAPTR_TRACER — The calling actor's tracer, if any.

Remember that the nature of these pointers depend on the actor type and is not always intuitive.

Return value

The function returns true if the number of samples of type in the pointed-to actor's inventory (before attempting removal) is greater than zero, otherwise it returns false.

Examples

This is a working example of a forgetful imp that uses inventory for timed checks. A_TakeInventory is used to reset the inventory based timer.

ACTOR ForgetfulImp : DoomImp

```
{
  States
  {
    See:
      TROO AABCCDD 3 A_Chase
      TROO A 0 A_GiveInventory("Forgettimer", 1) // A dummy inventory for tracking how long the imp has
      been searching.
      TROO A 0 A_JumpIfInventory("Forgettimer", 20, "Forget") // Jump to the Forget state when the timer
      reaches 20.
      Loop
      Melee:
      Missile:
        TROO E 0 A_TakeInventory("Forgettimer", 255) // Reset the timer.
        TROO EF 8 A_FaceTarget
        TROO G 6 A_TroopAttack
        Goto See
      Forget:
        TROO A 0 A_TakeInventory("Forgettimer", 255)
        TROO A 3 A_ClearTarget
        Goto Spawn
  }
}
```


A_BabyMetal

(no parameters)

Plays the sound "baby/walk" and calls A_Chase.

Examples

This example is taken from Doom's Arachnotron

See:

BSPI A 20

BSPI A 3 A_BabyMetal

BSPI ABBCC 3 A_Chase

BSPI D 3 A_BabyMetal

BSPI DEEFF 3 A_Chase

goto See+1

A_BarrelDestroy

(no parameters)

Removes the calling actor unless the game is in a multiplayer mode and the sv_barrelrespawn CVAR is set.

This function is used by Doom's barrel to selectively respawn in multiplayer games.

A_BFGSpray

```
void A_BFGSpray [(class<Actor> spraytype [, int numrays [, int damagecnt [, double ang [, double distance [, double vrange [, int defdamage [, int flags]]]]]]]]]
```

Usage

Performs the secondary, "tracer" attack of the BFG9000. Note that this function will create unpredictable effects if used on a non-missile.

The tracer actors (spraytype) which are spawned by the function could have custom damage types applied to them, much like other weapon puffs. Note that the EXTREMEDEATH flag still has no effect on this, however, so modders must use DamageType "Extreme" for it to work properly. If PUFFGETSOWNER is used, however, it will use the calling actor's owner as the inflictor -- meaning if the owner has EXTREMEDEATH, then it will actually enact that flag. Specifically, if a BFG ball shot from a player calls A_BFGSpray and the sprays have the PUFFGETSOWNER flag, it relies upon the player's actor to determine causing extreme death or not. A tracer with the MTHRUSPECIES flag set prevents this function from having any effect on actors with the same species as the shooter's own.

Parameters

spraytype: The actor to spawn at the position of each shootable actor that is hit. Default is "BFGExtra".

numrays: Spawns this many of the specified actor across an angle of 90 degrees with the target in the center. Default is 40, i.e. one tracer every 2.25°.

damagecnt: The count of iterations to calculate the damage for original BFG formula; it will add N random values from 1 to 8 together based on this value. Default is 15.

ang: Determines the field of view. Any actors within this angle are subject to being sprayed and damaged. Default is 90 (which is the player's FOV).

distance: Determines how far the function should search for monsters from the calling actor. Default is 1024.

vrange: Determines maximum vertical angle to autoaim at, in degrees. Default is 32.

defdamage: If greater than 0, the tracers deal this exact amount of damage and damagecnt is ignored. Default is 0.

flags: The following flags can be combined by usgin the bit-wise OR operator (|):

BFGF_MISSILEORIGIN " If set, the tracers are emitted from the projectile rather than the shooter.

BFGF_HURTSOURCE " If set, the shooter can be struck by their own projectile's tracers. This only has an effect if BFGF_MISSILEORIGIN is also set.

Examples

Because of the unique behavior of the BFG, this function has very limited use in custom projects. The only built-in class that makes use of it is the BFGBall, which is fired by the BFG, and triggers the function upon exploding.

Death:

```
BFE1 AB 8 Bright
BFE1 C 8 Bright A_BFGSpray
BFE1 DEF 8 Bright
```

Stop

ZScript definition

```
extend class Actor
```

```
{
    //
    // A_BFGSpray
    // Spawn a BFG explosion on every monster in view
    //
    void A_BFGSpray(class<Actor> spraytype = "BFGExtra", int numrays = 40, int damagecnt = 15, double ang = 90,
double distance = 16*64, double vrange = 32, int defdamage = 0, int flags = 0)
    {
        int damage;
        FTranslatedLineTarget t;
```

```

// validate parameters
if (spraytype == null) spraytype = "BFGExtra";
if (numrays <= 0) numrays = 40;
if (damagecnt <= 0) damagecnt = 15;
if (ang == 0) ang = 90.;
if (distance <= 0) distance = 16 * 64;
if (vrangle == 0) vrangle = 32.;

// [RH] Don't crash if no target
if (!target) return;

// [XA] Set the originator of the rays to the projectile (self) if
// the new flag is set, else set it to the player (target)
Actor originator = (flags & BFGF_MISSILEORIGIN) ? self : target;

// offset angles from its attack ang
for (int i = 0; i < numrays; i++)
{
    double an = angle - ang / 2 + ang / numrays*i;

    originator.AimLineAttack(an, distance, t, vrangle);

    if (t.linetrans != null)
    {
        Actor spray = Spawn(spraytype, t.linetrans.pos + (0, 0, t.linetrans.Height / 4),
ALLOW_REPLACE);

        int dmgFlags = 0;
        Name dmgType = 'BFGSplash';

        if (spray != null)
        {
            if ((spray.bMThruSpecies && target.GetSpecies() == t.linetrans.GetSpecies()) ||
                (!(flags & BFGF_HURTSOURCE) && target == t.linetrans)) // [XA] Don't hit oneself
            {
                spray.Destroy(); // [MC] Remove it because technically, the spray isn't trying to "hit"
                continue;
            }
            if (spray.bPuffGetsOwner) spray.target = target;
            if (spray.bFoilInvul) dmgFlags |= DMG_FOILINVUL;
            if (spray.bFoilBuddha) dmgFlags |= DMG_FOILBUDDHA;
            dmgType = spray.DamageType;
        }

        if (defdamage == 0)
        {
            damage = 0;
            for (int j = 0; j < damagecnt; ++j)
                damage += Random[BFGSpray](1, 8);
        }
        else

```

unless we say so.

them.

```
{  
    // if this is used, damagecnt will be ignored  
    damage = defdamage;  
}
```

```
    int newdam = t.linetarget.DamageMobj(originator, target, damage, dmgType,  
dmgFlags|DMG_USEANGLE, t.angleFromSource);  
    t.TraceBleed(newdam > 0 ? newdam : damage, self);  
}  
}  
}
```

A_FatRaise

(no parameters)

Plays the sound "fatso/raiseguns" and calls A_FaceTarget.

Examples

This example is taken from Doom's Mancubus.

Missile:

```
FATT G 20 A_FatRaise
FATT H 10 bright A_FatAttack1 // See FatShot
FATT IG 5 A_FaceTarget
FATT H 10 bright A_FatAttack2
FATT IG 5 A_FaceTarget
FATT H 10 bright A_FatAttack3
FATT IG 5 A_FaceTarget
goto See
```


A_FireCrackle

(no parameters)

Plays the sound "vile/firecrkl" and calls A_Fire.

Examples

This example is taken from Doom's Archvile Fire

Spawn:

FIRE A 2 bright A_StartFire

FIRE BAB 2 bright A_Fire

FIRE C 2 bright A_FireCrackle

FIRE BCBCDCDCDEDED 2 bright A_Fire

FIRE E 2 bright A_FireCrackle

FIRE FEFEFGHGHGH 2 bright A_Fire

Stop

A_Hoof

(no parameters)

Plays the sound "cyber/hoof" on the BODY channel and calls A_Chase.

Examples

This example is taken from Doom's Cyberdemon.

See:

CYBR A 3 A_Hoof

CYBR ABBCC 3 A_Chase

CYBR D 3 A_Metal

CYBR D 3

loop

A_Metal

(no parameters)

Plays the sound "spider/walk" and calls A_Chase.

Examples

This example is taken from Doom's Cyberdemon.

See:

CYBR A 3 A_Hoof

CYBR ABBCC 3 A_Chase

CYBR D 3 A_Metal

CYBR D 3

loop

A_SkelWhoosh

(no parameters)

Plays the sound "skeleton/swing" and calls A_FaceTarget.

Examples

This example is taken from Doom's Revenant.

Melee:

SKEL G 0 A_FaceTarget

SKEL G 6 A_SkelWhoosh

SKEL H 6 A_FaceTarget

SKEL I 6 A_SkelFist

Goto See

A_StartFire

(no parameters)

Plays the sound "vile/firestr" and calls A_Fire.

A_Countdown

void A_Countdown ()

Warning: this function should only be used with missile-type actors. Using it with other types is considered undefined behavior, and may result in unwanted side effects.

Usage

Counts ReactionTime down until it reaches 0 and then destroys the calling actor.

Examples

This revenant missile follows the target until its animation has played 25 times, upon which it then explodes.

```
actor RevenantTracer2 : RevenantTracer
```

```
{
```

```
    ReactionTime 25
```

```
    States
```

```
    {
```

```
    Spawn:
```

```
        FATB AB 2 Bright A_Tracer
```

```
        FATB A 0 A_Countdown
```

```
        Loop
```

```
    }
```

```
}
```

A_CountdownArg

A_CountdownArg (int arg[, str state])

Note: Jump functions perform differently inside of anonymous functions.

This function counts one of the actor's args down until it reaches 0 and then changes its state, possibly killing it.

arg: what argument to decrement. Using a zero-based index, valid values are in the 0â€“4 range. Any other will result in the function doing nothing.

state: what state to set the calling actor to once the countdown is reached. Default is "Death". If such state is not defined the actor is simply removed without other effects. This only affects non-missile actors without the SHOOTABLE flag. The method used to destroy the calling actor depends on its actor flags.

If it has the MISSILE flag the missile explodes.

Otherwise, if it has the SHOOTABLE flag, the actor is dealt an amount of damage equal to its current health.

In all other cases the actor is placed in its "Death" state, or in state if one is provided without other effects (means it does not necessarily kills or removes the calling actor).

Examples

This dispenser will throw some junk after being hit. If it has no more or damaged too much it explodes.

Wound:

DISP A 10 A_SpawnItemEx("DispenserJunk", random(-16, 16), random(-16, 16), 16)

TNT1 A 0 A_CountdownArg(0)

loop

Death: // reuses Rocket's explosion sprites.

MISL B 8 bright A_Scream

MISL C 6 bright

MISL D 4 bright

stop

A_FaceTarget

```
void A_FaceTarget [(double max_turn [, double max_pitch [, double ang_offset [, double pitch_offset [, int flags [, double z_ofs]]]]]]]  
void A_FaceTracer [(double max_turn [, double max_pitch [, double ang_offset [, double pitch_offset [, int flags [, double z_ofs]]]]]]]  
void A_FaceMaster [(double max_turn [, double max_pitch [, double ang_offset [, double pitch_offset [, int flags [, double z_ofs]]]]]]]
```

Usage

Change the calling actor's angle to face a specific actor, depending on the chosen function.

A_FaceTarget changes the angle to face the calling actor's target.

A_FaceTracer changes the angle to face the calling actor's tracer.

A_FaceMaster changes the angle to face the calling actor's master.

Parameters

max_turn: The maximum turn angle; the calling actor cannot turn by more than said angle, however the SHADOW flag has no effect in such case. A value of 0 is interpreted as unlimited angle. Default is 0.

max_pitch: If specified to a value no greater than 180, then the calling actor's pitch is adjusted up to said value to face the actor. A value of 0 is interpreted as unlimited angle; and technically a pitch change will never be greater than 180 degrees. It will also aim at the actor's feet when set to 0. Default is 270, which means its disabled.

ang_offset: Specifies the amount of degrees to offset the actor's angle by. Positive values turn it left, while negative values turn it right. This is factored in after max_turn is performed. Due to limitations, distance is not factored in. Default is 0.

pitch_offset: Adjusts the pitch by this many degrees after max_pitch has been taken into account. Default is 0.

flags: Customizes the behavior of the function. Multiple flags can be combined by using the bitwise OR operator (|) between the constant names:

FAF_BOTTOM "Aim for the bottom of the actor, otherwise known as the raw Z position. Whenever max_pitch is taken into account, it will aim towards the actor's feet + 32 units above. This flag disables adding that 32 units.

FAF_MIDDLE "Aim for the middle of the actor (z position + height / 2).

FAF_TOP "Aim for the top of the actor (z position + height).

Note that all of these flags are taken into account first before anything else.

Default is 0.

z_ofs: Offsets the z position distance of the actor to face by this amount. Unlike pitch_offset, this takes into account how far away the actor is at all times. Default is 0.

Examples

Almost every monster uses A_FaceTarget before it shoots at its target.

Missile:

```
TROO EF 8 A_FaceTarget  
TROO G 6 A_TroopAttack // See DoomImpBall  
Goto See
```

This lost soul homes in on its target by calling A_FaceTarget and A_SkullAttack repeatedly.

actor HomingLostSoul : LostSoul

```
{  
    States  
    {  
        Missile:  
            SKUL C 10 Bright A_FaceTarget  
            SKUL D 4 Bright A_SkullAttack  
            SKUL CD 4 Bright
```


Goto Missile+1

}
}

A_Fire

A_Fire [(float height)]

Places the calling actor directly in front of its tracer, with a z offset of height. If height is not specified, it defaults to 0.

Examples

The Arch-Vile from Doom uses this function in its fire attack's Decorate code to keep the fire placed on its target:

```
ACTOR ArchvileFire
{
    Game Doom
    SpawnID 98
    RenderStyle Add
    Alpha 1
    +NOBLOCKMAP
    +NOGRAVITY
    States
    {
    Spawn:
        FIRE A 2 bright A_StartFire
        FIRE BAB 2 bright A_Fire
        FIRE C 2 bright A_FireCrackle
        FIRE BCBCDCDCDEDED 2 bright A_Fire
        FIRE E 2 bright A_FireCrackle
        FIRE FEFEFGHGHGH 2 bright A_Fire
        Stop
    }
}
```

A_SeekerMissile

A_SeekerMissile (angle threshold, angle maxturnangle, int flags = 0, int chance = 50, int distance = 10)

Seeker missile handling. threshold and maxturnangle determine how 'aggressive' the missile will home in on its tracer. The larger the values the more precise it is.

threshold

Specifies the angle inside which the missile will home in directly on its tracer. If the angle toward the target is larger than threshold it will change its movement angle only partially towards the tracer.

maxturnangle

The maximum change of movement direction that will be performed in one move. maxturnangle should be larger than threshold. Both angles are specified in degrees and must be in the range [0, 90].

flags

The following flags can be combined by using the | character between the constant names. The default value is 0 for no flags.

SMF_LOOK "Look for targets: If this flag is passed, the missile will try to acquire a target if it does not already have one. The SCREENSEEKER flag on the projectile can restrict this search for potential targets that are in the shooter's field of vision.

SMF_PRECISE "Precise trajectory: If this flag is passed, the missile will chase its target in true 3D, readjusting its vertical velocity with each call, making it harder to lead it to crash on a ground or ceiling.

SMF_CURSPEED "Use current speed: If this flag is passed, the missile will keep its current velocity rather than use a vector calculated from its default Speed value.

chance

If the SMF_LOOK flag is used, this is the chance (out of 256) that the missile will try acquiring a target if it doesn't already have one. The default value is 50.

distance

If the SMF_LOOK flag is used, this is the maximum distance (in blocks of 128 map units) at which targets are sought. The default value is 10 (i.e. roughly 1080 map units); values much larger should be avoided or the function might become too much resource intensive.

The range at which a player-fired seeker missile will first acquire a tracer can be set using MaxTargetRange actor property. Note that the value to be set is in a 128-map-unit block. For instance, a value of 4 means a range of 512 map units ($4 * 64 = 512$). Behind the scenes, this function uses the BLOCKMAP to perform this search, so the actual maximum range is not always exact (it depends on how close the player is to a blockmap boundary); if exact unit precision is desired, it may be necessary to do an additional distance check manually after acquiring the tracer.

It is also worth mentioning that NOAUTOAIM should be disabled if SMF_LOOK isn't being used, otherwise it will not home when fired from a weapon.

Examples

This is an example for a magic missile that homes in on its tracer. It has a low maxturnangle which gives it a laggy turning effect.

ACTOR MagicMissile

```
{
  Projectile
  +RANDOMIZE
  +SEEKERMISSILE
  Height 16
  Radius 8
  Speed 10
```

```
Damage (random(13, 17))
RenderStyle "Add"
Alpha 0.8
States
{
Spawn:
    MMIS B 2 Bright A_SeekerMissile(0, 2) // MMIS is the sprite used for the missile.
    Loop
Death:
    MMIS CDE 5 Bright
    Stop
}
}
```

A_Stop

(no parameters)

This stops the actor's current movement by setting its acceleration and speed to 0. This can also be used on players, monsters and anything else that is moving vertically or horizontally. Additionally when used on a player, it'll jump them to their idle state if it exists, otherwise their spawn state.

Examples

This Lost Soul flies through the air to attack but stops after a short distance. Without A_Stop it would return to its See state while still flying towards its target.

ACTOR SoulStopper : LostSoul

```
{
  States
  {
    Missile:
      SKUL C 10 bright A_FaceTarget
      SKUL D 4 bright A_SkullAttack
      SKUL CDCDCDC 4 Bright
      SKUL C 0 A_Stop
      Goto See
  }
}
```

A_Tracer

(no parameters)

The seeking function for the Revenant's missile. This seeks very aggressively and spawns both a puff and a smoke behind the missile. This only works for missiles with the SEEKERMISSILE flag.

Note: This function is written so that it only performs its action if the global time counter is a multiple of 4, so it may have some side effects — for instance, Revenant rockets, which call this function every two tics, only home in if they were spawned during an even tic; if they were spawned during an odd tic the function will never be called during a tic that is a multiple of 4, and the tracer will not home in. The frequency at which A_Tracer is called is thus very important for its behavior:

If the interval is odd, the homing behavior will happen every $\text{interval} \times 4$ tics.

If it is a multiple of four, it will happen every interval tics, but only if it was spawned during a tic that is a multiple of four.

If it is an even number that is not a multiple of four, it will happen every $\text{interval} \times 2$ tics only if it was spawned during an even tic.

See examples below.

Examples

The original example is RevenantTracer. It only homes every four tics if it were spawned during an even tic. The following table underlines at which gametic the function will have an effect, depending on the gametic-modulo-four at which the RevenantTracer actor was spawned:

0: 02 04 06 08 10 12 14 16 etc.

1: 03 05 07 09 11 13 15 17 etc.

2: 04 06 08 10 12 14 16 18 etc.

3: 05 07 09 11 13 15 17 19 etc.

actor RevenantTracer

```
{
  // See RevenantTracer for the full definition!
  States
  {
    Spawn:
      FATB AB 2 bright A_Tracer // See RevenantTracerSmoke
    loop
  }
}
```

This "Tracer1" variant will home in very aggressively every four tics, no matter when it was spawned.

0: 01 02 03 04 05 06 07 08 09 10 11 12 etc.

1: 02 03 04 05 06 07 08 09 10 11 12 13 etc.

2: 03 04 05 06 07 08 09 10 11 12 13 14 etc.

3: 04 05 06 07 08 09 10 11 12 13 14 15 etc.

Actor Tracer1 : RevenantTracer

```
{
  States
  {
    Spawn:
      FATB AB 1 bright A_Tracer
    loop
  }
}
```

This "Tracer3" variant will home in on the target every time, but only steer towards it once every 12 tics.

0: 03 06 09 12 15 18 21 24 27 30 33 36 etc.

1: 04 07 10 13 16 19 22 25 28 31 34 37 etc.
2: 05 08 11 14 17 20 23 26 29 32 35 38 etc.
3: 06 09 12 15 18 21 24 27 30 33 36 39 etc.

Actor Tracer3 : RevenantTracer

```
{
  States
  {
    Spawn:
      FATB AB 3 bright A_Tracer
      loop
  }
}
```

This "Tracer4" variant will only home in if it was spawned during a gametic that is a multiple of 4, as shown by the table below:

0: 04 08 12 16 20 etc.
1: 05 09 13 17 21 etc.
2: 06 10 14 18 22 etc.
3: 07 11 15 19 23 etc.

Actor Tracer4 : RevenantTracer

```
{
  States
  {
    Spawn:
      FATB AB 4 bright A_Tracer
      loop
  }
}
```

This "Tracer5" variant will home in every twenty tics.

0: 05 10 15 20 25 30 35 40 etc.
1: 06 11 16 21 26 31 36 41 etc.
2: 07 12 17 22 27 32 37 42 etc.
3: 08 13 18 23 28 33 38 43 etc.

Actor Tracer5 : RevenantTracer

```
{
  States
  {
    Spawn:
      FATB AB 5 bright A_Tracer
      loop
  }
}
```

Likewise, a "Tracer6" variant would home every 12 rounds only if it were spawned during an even tic; a "Tracer7" would home every 28 tics, a "Tracer8" variant would home every eight rounds only if it were spawned during a tic that is a multiple of four, and so on.

A_Tracer2

(no parameters)

This is Strife's seeker missile function. It is mostly identical with A_Tracer but there are a few notable differences:

This function always seeks. There is no randomization involved.

This function does not spawn puffs behind the missile.

The seeking angle is slightly larger than for A_Tracer.

This only works for missiles with the SEEKERMISSILE flag.

A_Warp

state, bool A_Warp (int ptr_destination [, float xofs [, float yofs [, float zofs [, float angle [, int flags [, state success_state [, float heightoffset [, float radiusoffset [, float pitch]]]]]]]]))

Usage

This command is considered an extension of A_Fire, with more versatility and flexible actor controls. Unlike A_Fire, it does not require an actor to be visible like the Archvile's fire attack, and these can be used with actor pointers.

Parameters

ptr_destination: Moves the calling actor to the specified actor pointer. See actor pointers for more information.

xofs: Specifies how far forward/backward in front/behind of the warped-to actor the warping actor will appear. Positive numbers are further forward, and negative numbers will go backward.

yofs: Specifies how far to the side the warping actor will appear to the warped-to. Positive numbers are further to the right, while negative numbers spawn them more to the left.

zofs: Specifies how far upward the warping actor will appear to the warped-to actor. Positive numbers are further upward, while negative means under the actor and possibly into the ground.

angle: Specifies an offset from the warped-to actor's angle to set the warping actor's angle to. Default (zero) is to just use the warped-to actor's angle exactly.

flags:

Flags that affect the behavior:

WARPF_ABSOLUTEOFFSET — By default the xofs and yofs parameters apply the warped-to actor's angle; this flag disables this behaviour.

WARPF_ABSOLUTEANGLE — Use the angle parameter as an absolute angle, instead of an offset.

WARPF_ABSOLUTEPOSITION — Treat x, y and z offset parameters as absolute coordinates for the warping actor's position, instead of being relative to the warped-to actor's. This flag overrides

WARPF_ABSOLUTEOFFSET, but can still add the z offset of floorz when used with WARPF_TOFLOOR.

WARPF_USECALLERANGLE — Use the warping actor's angle instead of the warped-to actor's. The angle parameter will instead add itself to the warping's angle and cause it to orbit around the warped-to actor's. The greater the angle parameter, the faster it will orbit.

WARPF_NOCHECKPOSITION — Blindly accept any resultant position, instead of checking for the warp being a valid movement.

WARPF_STOP — Reduce warping actor's velocity to 0 when the move is completed, similar to A_Stop.

WARPF_TOFLOOR — Makes the zofs parameter relative to the floor, rather than relative to the warped-to actor.

WARPF_TESTONLY — Does not perform any warping, but still jumps to the success state if it is able to warp. This flag ignores all other flags and properties, including but not limited to pitch and angle changes, with the exception of WARPF_NOCHECKPOSITION which always jumps to the success state.

WARPF_BOB — Makes the warping actor's floatbob phase follow the warped-to actor's.

WARPF_MOVEPTR - The target is set to warp instead of the calling actor. The calling actor is still responsible for performing a state jump and determining success, however.

WARPF_USETID — If set, the first parameter changes to accept a tid instead of an actor pointer.

WARPF_COPYVELOCITY - Copies the warped-to actor's velocity exactly, regardless of how the warping actor is moving.

WARPF_COPYPITCH - Copies the warped-to actor's pitch, and then adds the pitch parameter to it.

Flags to customize appearance of the warp:

WARPF_INTERPOLATE — Keeps the warping actor's default interpolation data.

WARPF_WARPINTERPOLATION — Modify interpolation data with the vector PlayerNewPosition - PlayerOldPosition.

WARPF_COPYINTERPOLATION — Copies the warped-to actor's interpolation data, allowing the warping actor to mimic it.

success_state: An optional state to jump to if the warp succeeds.

heightoffset: Adds the warped-to actor's height multiplied by heightoffset to the zofs parameter. Default is 0.

radiusoffset: Adds the warped-to actor's radius multiplied by radiusoffset to the xofs and yofs parameters.

Default is 0.

pitch: With WARPF_COPYPITCH, this parameter works exactly as angle, but with pitch instead. Without WARPF_COPYPITCH, this parameter merely adds to the warping actor's pitch.

Note: If the actor being warped to does not exist or has stopped existing, and the success state is defined, it will always fail to jump. Even if success state is left undefined, though, it will fail to move at all. The success state can be useful for getting rid of orbiting/external actors when the main actor is no longer being used.

Note: Jump functions perform differently inside of anonymous functions.

Examples

The following example is a replication of A_Fire:

```
A_Warp(AAPTR_TRACER, 24, 0, 0, 0, WARPF_NOCHECKPOSITION|WARPF_INTERPOLATE)
```

A useful application of this function is to have "orbiters". This example baron has a projectile circulating it all the time.

```
ACTOR SpecialBaronOfHell : BaronOfHell
```

```
{
  States
  {
    Spawn:
      BOSS A 0 NoDelay A_CustomMissile("OrbitBall", 32, 0, 0, CMF_AIMDIRECTION)
    Idle:
      Goto Super::Spawn
  }
}
```

```
ACTOR OrbitBall
```

```
{
  RenderStyle "Add"
  Translation "0:255=%[0,0,0]:[0.93,2,0.87]" // Makes it green-colored
  +MISSILE
  +NOINTERACTION

  var int user_angle; // See user variables

  States
  {
    Spawn:
      BAL1 A 1 Bright NoDelay A_Warp(AAPTR_TARGET, 32, 0, 32, user_angle,
      WARPF_ABSOLUTEANGLE|WARPF_NOCHECKPOSITION|WARPF_INTERPOLATE)
      TNT1 A 0 A_SetUserVar("user_angle", user_angle + 8)
    Loop
  }
}
```

A_Weave

A_Weave (int horzspeed, int vertspeed, float horzdist, float vertdist)

A generalized version of A_BishopMissileWeave and A_CStaffMissileSlither. You have to call this function repeatedly to have a permanent effect.

A_BishopMissileWeave is equivalent to A_Weave(2, 2, 2.0, 1.0); A_CStaffMissileSlither is equivalent to A_Weave(3, 0, 1.0, 0.0).

Parameters

A weave in ZDoom can be understood as the sum of two waves, a horizontal and a vertical one.

horzspeed: the speed at which the projectile weaves horizontally; that is to say, it affects the frequency of the horizontal wave.

vertspeed: same but for the vertical wave.

horzdist: the maximum distance to which the projectile will horizontally stray from its linear trajectory; that is to say, the semi-amplitude of the horizontal wave.

vertdist: same but for the vertical wave.

In addition to these parameters, the actor's WeaveIndexXY and WeaveIndexZ properties as well as the frequency at which A_Weave is called are both instrumental in determining the exact behavior of projectile.

A_AlertMonsters

A_AlertMonsters [(float maxrange [, int flags])]

Usage

This can be used on weapons, projectiles and monsters:

For monsters it alerts all monsters in hearing range of the current monster's target.

For projectiles it alerts all monsters in hearing range of the projectile's shooter.

For weapons it alerts all monsters in hearing range of the player holding the weapon.

This function does nothing on monsters which already have a target.

Parameters

maxrange: The maximum distance from the calling actor at which monsters can still be alerted. Note that this is a simple distance check to the alerter. The "sound" still travels the normal distance. Default is 0 (unlimited range)

flags: The following flags can be combined by using the | character between the constant names:

AMF_TARGETEMITTER â€” Causes alerted monsters to chase the actor which emitted the alert. Note that the alerter must be alive and has the SHOOTABLE flag to be chased (this is standard ZDoom behavior, not a fault of the feature).

AMF_TARGETNONPLAYER â€” Allows a monster to alert others to its own target when that target is not a player.

Note that AMF_TARGETEMITTER implies that non-players can be targeted, so there is no need to use both at once.

AMF_EMITFROMTARGET â€” Causes the alert to be emitted from the actor that is being warned about, instead of the calling actor. Note that this is inherently pointless when using AMF_TARGETEMITTER.

Note: The maxrange will be foiled if compat_soundtarget is true, since it makes monsters rely purely on whether sound was heard in the sector in which they are, rather than what the monster itself heard.

Examples

The StrifeZap1 actor class uses this function to wake up enemies when the projectile explodes.

Death:

```
ZAP1 A 3 A_AlertMonsters
```

```
ZAP1 BCDEF 3
```

```
ZAP1 DCB 2
```

```
ZAP1 A 1
```

```
Stop
```

A_Burst
Jump to navigationJump to search
A_Burst (string classname)

Spawns a random numbers of actors of type classname, positions them all around the calling actor, and then destroys the calling actor, removing it instantly from the game. The amount spawned is relative to the calling actor's size, so a very large actor will spawn many more classnames than a much smaller actor. An actor with radius 20 and height 64 spawns, on average, around 40 things.

If the calling actor has the BOSSDEATH flag, it also calls A_BossDeath.

Examples
This would make a modified grenade that would split in its death state into an actor called "MiniGrenade"

```
Actor ClusterBomb : Grenade
{
    States
    {
    Death:
        MISL B 8 Bright A_Burst ("MiniGrenade")
        MISL C 6 Bright
        MISL D 4 Bright
        Stop
    }
}
```

A_CentaurDefend
Jump to navigationJump to search
A_CentaurDefend

(no parameters)

Usage

Calls A_FaceTarget. If the target is within melee range of the calling actor, there is a 12.5% chance that the calling actor clears its REFLECTIVE and INVULNERABLE flags and goes into its Melee state.

Examples

ACTOR MirrorImp : DoomImp

```
{
  States
  {
    Pain:
      TROO H 6 A_Pain
      TROO H 6 A_SetReflectiveInvulnerable
      TROO FFF 15 A_CentaurDefend
      TROO F 1 A_UnsetReflectiveInvulnerable
      Goto See
    Melee:
      TROO EF 4 A_FaceTarget
      TROO G 3 A_CustomMeleeAttack(50, "skeleton/melee", "none")
      Goto See
  }
}
```

This is a modified Imp using the same guard stance as the centaur when it enters the pain state. When the imp is hurt, it will stop moving and become invulnerable and reflective for a short time. But if the player is too close to it, A_CentaurDefend has a chance to make the imp switch to its melee state and deliver a dangerously powerful blow to the player.

A_CentaurDefend works together with A_SetReflectiveInvulnerable and A_UnsetReflectiveInvulnerable, and would not make much sense without the other two in the correct order.

A_Chase

```
void A_Chase [(statelabel melee [, statelabel missile [, int flags]])]
```

Usage

This is the standard monster walking function which has to be used in the walking frames of a monster. Typically, it is used in an actor's "See" state or a custom equivalent. When called, actors usually change their directions to a strict 45 degree angle to give the effect of pursuit. This angle changes based on which direction the target is, no matter if the calling actor can see it or not.

Parameters

Note: passing "_a_chase_default" to both the melee and missile parameters, puts the function in its default-behavior mode. In this mode, the monster jumps to Melee and Missile states when performing melee and ranged attacks, respectively, does not strafe randomly, plays its active sound, and moves twice as fast in Nightmare! difficulty setting or equivalent. Additionally, the flags parameter is ignored.

melee: The state to jump to when the monster decides to perform a melee attack. If this is null(ZScript only)/0 or "" (empty string)(DECORATE only), the monster will not be able to enter the melee state while chasing. Default is "_a_chase_default".

missile: The state to jump to when the monster decides to perform a ranged attack. If this is null(ZScript only)/0 or "" (empty string)(DECORATE only), the monster will not be able to enter the ranged state while chasing. Default is "_a_chase_default".

flags: Determines how the monster will behave when the function is called. Flags can be combined by using the | character between the constant names. Default is 0. Because of the special way in which A_Chase handles its state jumps, the preceding state parameters must be specified or these flags will be ignored. CHF_FASTCHASE – Actor will randomly attempt to strafe left or right around the target, like the "player bosses" in Hexen. Note that the calling actor will ignore their MaxDropOffHeight property when strafing, and so may strafe off cliffs or into pits from which they can not escape. Setting the NODROPOFF flag will prevent this from happening.

CHF_NOPLAYACTIVE – Actor will not play active sounds.

CHF_NIGHTMAREFAST – Actor will move twice as fast in Nightmare! difficulty setting or equivalent.

CHF_RESURRECT – Actor will enter the "Heal" state (if defined) upon encountering a revivable corpse, like the Arch-Vile.

CHF_DONTMOVE – Actor will not move.

CHF_NORANDOMTURN – Actor will not change its chasing angle to its target if its beyond a certain degree until something blocks the way (obstacle or wall).

CHF_NODIRECTIONTURN – Actor will not turn its angle to face the direction of travel.

CHF_NOPOSTATTACKTURN – Actor will not turn itself to its target after attacking.

CHF_STOPIFBLOCKED – Actor cannot turn away from obstacles blocking it. It will simply not move, but can still angle itself.

CHF_DONTIDLE – Hostile actors will not jump to their Spawn or Idle state once they no longer have a target. Allowing you to define custom states for them to jump to, when they have no target.

CHF_DONTTURN – Implies CHF_NORANDOMTURN, CHF_NOPOSTATTACKTURN and CHF_STOPIFBLOCKED.

Notes

This function is not subject to the same rules as other jump functions are in anonymous functions, and cannot be used inside if/else statements. A_Chase will perform its jump, no matter what circumstances apply.

Because this function and its variant have the capacity to put the actor into an attack state, they should not be used during the first tic of one of those states, as it may result in infinite recursion, which will usually manifest itself in the engine as a crash.

Examples

Here is an example of a new monster which uses the optional range attack as well as a melee attack if it is close to its target, like the Revenant.

```
class Scurymonster : Actor
{
    Default
```

```

{
    Monster;
    Height 20; // Default variables.
    Radius 16; // Default variables.
    Speed 10;
}

States
{
    Spawn:
        MONS A 10 A_Look;
        Loop;
    See:
        MONS ABCD 4 A_Chase("Meleeattack", "Cowattack");
        Loop;
    Cowattack:
        MONS E 5 A_SpawnProjectile("Cowmissile"); // Not included in-game.
        Goto See;
    Meleeattack:
        MONS F 5 A_CustomMeleeAttack(5);
        Goto See;
}
}

```


A_CheckForResurrection

bool A_CheckForResurrection [(State state [, Sound snd])]

Note: starting with GZDoom 4.7.0, this function is no longer fully supported by DECORATE. If used from DECORATE, it is to be called without passing any parameters to it. Consider switching to ZScript to fully utilize this function.

Usage

Checks for a revivable corpse. If the calling actor comes into contact with one, it resurrects the actor in question, provided the actor being revived can actually be resurrected. The outcome of CanResurrect is another determining factor in the success or failure of an actor's resurrection.

Parameters

state: a pointer to the state to enter. Passing null defaults to the Heal state. If the Heal state does not exist, no state is entered. Default is null.

snd: the sound to play. The sound is played by the resurrected actor on the body (CHAN_BODY) channel with idle (ATTN_IDLE) attenuation. Passing 0 or an invalid sound name defaults to "vile/raise" as the sound to play. Default is 0.

Return value

The function returns true if the actor is resurrected successfully, otherwise it returns false.

Examples

This mysterious barrel wanders around. If it stumbles upon a corpse, it revives it, then it explodes.

```
class SentientBarrel : ExplosiveBarrel
{
    Default
    {
        Speed 6;
    }

    States
    {
        Spawn:
            BAR1 AB 6
            {
                if (!A_CheckForResurrection(FindState("Rise"), "grunt/pain"))
                {
                    A_Wander();
                }
            }
        Loop;

        Rise:
            BAR1 ABABABABABABABABABAB 2;
            BAR1 A -1 A_Die;
            Stop;
    }
}
```

A_ClearLastHeard
A_ClearLastHeard

(no parameters)

Makes the calling actor forget about the last thing it heard. This does not prevent it from hearing new things after the call, nor does it cause it to lose its target if it already has one (see A_ClearTarget for this).

This function is ineffective when `compat_soundtarget` is active.

A_ClearSoundTarget

A_ClearSoundTarget

(no parameters)

Clears the sound target for all actors in this actor's sector. Anything not actively chasing a target will be unalerted by this.

To make an individual actor forget its target (without affecting other actors), see A_ClearTarget.

A_ClearTarget

A_ClearTarget

(no parameters)

Usage

Makes the calling actor forget about its target, sound target, and last target. You can call this before jumping back to a monster's idle states to make a monster "give up" trying to chase after its target after a while (for ex. if it's too far away), after which it will resume searching for targets or doing whatever else is specified in its spawn state.

Examples

This is a working example of an imp after a while completely forgets what he was doing and goes back to dancing on the spot.

Actor ForgetfullImp : DoomImp

```
{
  States
  {
    See:
      TROO AABBCDD 3 A_Chase
      TROO A 0 A_GiveInventory("Forgettimer", 1) // A dummy inventory for tracking how long the imp has been
searching.
      TROO A 0 A_JumpIfInventory("Forgettimer", 20, "Forget") // Jump to the Forget state when the timer reaches 20
      Loop
    Melee:
    Missile:
      TROO E 0 A_TakeInventory("Forgettimer", 20) // Reset the timer.
      TROO EF 8 A_FaceTarget
      TROO G 6 A_TroopAttack
      Goto See
    Forget:
      TROO A 0 A_TakeInventory("Forgettimer", 20)
      TROO A 3 A_ClearTarget
      Goto Spawn
  }
}
```

A_DamageChildren

```
void A_DamageChildren (int amount [, name damagetype [, int flags [, class<Actor> filter [, name species [, int src [, int inflict]]]]]])
```

Usage

Damages the calling actor's child actors by the specified amount. Negative amounts heal them, instead. This function cannot damage player actors which are under the effect of god2, and they still survive with one point of health if under the effect of buddha2.

For the calling actor to be considered to have a "child" actor, the child actor's master field should point to it.

Parameters

amount: the amount of damage to inflict. Negative amounts heal. An amount of one million or higher is treated specially, and results in killing the actor regardless of health or any damage modifiers that may be in effect.

damagetype: the type of damage to inflict. Default is "None".

flags: the following flags can be combined using the | character between the constants names:

DMSS_FOILINVUL — the actor is damaged even if it is invulnerable. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_FOILBUDDHA — if the damage is enough to kill the actor, it dies even if it is under the effect of buddha. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_NOPROTECT — damage bypasses the damage-modifying capability of items in the actor's inventory. A protection powerup is an example of such items.

DMSS_NOFACTOR — damage bypasses the the actor's damage factors.

DMSS_AFFECTARMOR — damage does not bypass the damage-absorbing capability of items in the actor's inventory. An armor is an example of such items.

DMSS_KILL — inflicts an amount of damage which is equal to the sum of the actor's current health and amount, killing the actor under normal conditions. This flag bypasses the actor's damage factors and the damage-absorbing capability (but not the damage-modifying one) of items in the actor's inventory.

DMSS_EXFILTER — inverts the case of the class name filter; the actor is only damaged if its class name does not match the value passed to filter.

DMSS_EXSPECIES — inverts the case of the species filter; the actor is only damaged if its species does not match the value passed to species.

DMSS_EITHER — the actor is damaged if either its class name or species matches the values passed to filter and species, respectively.

DMSS_INFLICTORDMGTYPE — ignores the specified damage type, and instead, uses the damage type of the actor doing the damage (inflictor).

filter: the actor class to damage. The actor is only damaged if its class name matches the specified class.

Default is null(ZScript only)/"None"(DECORATE only).

species: the actor species to damage. The actor is only damaged if its species matches the specified species. Default is "None".

src: the actor responsible for the damage (source). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without a source. Default is AAPTR_DEFAULT, which is the calling actor itself.

inflict: the actor doing the damage (inflictor). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without an inflictor (note that doing so renders both DMSS_FOILINVUL and DMSS_FOILBUDDHA ineffective). Default is AAPTR_DEFAULT, which is the calling actor itself.

Examples

This "leader" imp spawns regular imps as a form of attack. Upon its death, all of its spawns die.

```
actor LeaderImp : DoomImp
```

```
{
    Health 100
    Mass 1000
    PainChance 255
```

States

{

Missile:

TROO EF 8 A_FaceTarget

TROO G 6 A_SpawnItemEx("DoomImp", 50, 50, 60, 0, 0, 0, 0, SXF_SETMASTER)

Goto See

Death:

TROO I 8 A_DamageChildren(0, "None", DMSS_KILL)

TROO J 8 A_Scream

TROO K 6

TROO L 6 A_NoBlocking

TROO M -1

Stop

}

}

A_DamageMaster

```
void A_DamageMaster (int amount [, name damagetype [, int flags [, class<Actor> filter [, name species [, int src [, int inflict]]]]]])
```

Usage

Damages the calling actor's master actor by the specified amount. Negative amounts heal it, instead. This function cannot damage player actors which are under the effect of god2, and they still survive with one point of health if under the effect of buddha2.

Parameters

amount: the amount of damage to inflict. Negative amounts heal. An amount of one million or higher is treated specially, and results in killing the actor regardless of health or any damage modifiers that may be in effect.

damagetype: the type of damage to inflict. Default is "None".

flags: the following flags can be combined using the | character between the constants names:

DMSS_FOILINVUL — the actor is damaged even if it is invulnerable. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_FOILBUDDHA — if the damage is enough to kill the actor, it dies even if it is under the effect of buddha. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_NOPROTECT — damage bypasses the damage-modifying capability of items in the actor's inventory. A protection powerup is an example of such items.

DMSS_NOFACTOR — damage bypasses the the actor's damage factors.

DMSS_AFFECTARMOR — damage does not bypass the damage-absorbing capability of items in the actor's inventory. An armor is an example of such items.

DMSS_KILL — inflicts an amount of damage which is equal to the sum of the actor's current health and amount, killing the actor under normal conditions. This flag bypasses the actor's damage factors and the damage-absorbing capability (but not the damage-modifying one) of items in the actor's inventory.

DMSS_EXFILTER — inverts the case of the class name filter; the actor is only damaged if its class name does not match the value passed to filter.

DMSS_EXSPECIES — inverts the case of the species filter; the actor is only damaged if its species does not match the value passed to species.

DMSS_EITHER — the actor is damaged if either its class name or species matches the values passed to filter and species, respectively.

DMSS_INFLICTORDMGTYPE — ignores the specified damage type, and instead, uses the damage type of the actor doing the damage (inflictor).

filter: the actor class to damage. The actor is only damaged if its class name matches the specified class. Default is null(ZScript only)/"None"(DECORATE only).

species: the actor species to damage. The actor is only damaged if its species matches the specified species. Default is "None".

src: the actor responsible for the damage (source). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without a source. Default is AAPTR_DEFAULT, which is the calling actor itself.

inflict: the actor doing the damage (inflictor). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without an inflictor (note that doing so renders both DMSS_FOILINVUL and DMSS_FOILBUDDHA ineffective). Default is AAPTR_DEFAULT, which is the calling actor itself.

Examples

This "leader" imp spawns clones of itself as a form of attack. Upon a clone's death, it gets damaged.

```
actor LeaderImp : DoomImp
```

```
{
    Health 100
    Mass 1000
    PainChance 255
```

```
    States
```

```
{
Missile:
    TROO EF 8 A_FaceTarget
    TROO G 6 A_SpawnItemEx("SoldierImp", 50, 50, 60, 0, 0, 0, 0, SXF_SETMASTER)
    Goto See
}
```

```
actor SoldierImp : DoomImp
{
    States
    {
    Death:
        TROO I 8 A_DamageMaster(25)
        TROO J 8 A_Scream
        TROO K 6
        TROO L 6 A_NoBlocking
        TROO M -1
        Stop
    }
}
```


A_DamageSelf

```
void A_DamageSelf (int amount [, name damagetype [, int flags [, class<Actor> filter [, name species [, int src [, int inflict]]]]]])
```

Usage

Damages the calling actor by the specified amount. Negative amounts heal it, instead. This function cannot damage player actors which are under the effect of god2, and they still survive with one point of health if under the effect of buddha2.

Parameters

amount: the amount of damage to inflict. Negative amounts heal. An amount of one million or higher is treated specially, and results in killing the actor regardless of health or any damage modifiers that may be in effect.

damagetype: the type of damage to inflict. Default is "None".

flags: the following flags can be combined using the | character between the constants names:

DMSS_FOILINVUL — the actor is damaged even if it is invulnerable. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_FOILBUDDHA — if the damage is enough to kill the actor, it dies even if it is under the effect of buddha. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_NOPROTECT — damage bypasses the damage-modifying capability of items in the actor's inventory. A protection powerup is an example of such items.

DMSS_NOFACTOR — damage bypasses the the actor's damage factors.

DMSS_AFFECTARMOR — damage does not bypass the damage-absorbing capability of items in the actor's inventory. An armor is an example of such items.

DMSS_KILL — inflicts an amount of damage which is equal to the sum of the actor's current health and amount, killing the actor under normal conditions. This flag bypasses the actor's damage factors and the damage-absorbing capability (but not the damage-modifying one) of items in the actor's inventory.

DMSS_EXFILTER — inverts the case of the class name filter; the actor is only damaged if its class name does not match the value passed to filter.

DMSS_EXSPECIES — inverts the case of the species filter; the actor is only damaged if its species does not match the value passed to species.

DMSS_EITHER — the actor is damaged if either its class name or species matches the values passed to filter and species, respectively.

DMSS_INFLECTORDMGTYPE — ignores the specified damage type, and instead, uses the damage type of the actor doing the damage (inflictor).

filter: the actor class to damage. The actor is only damaged if its class name matches the specified class. Default is null(ZScript only)/"None"(DECORATE only).

species: the actor species to damage. The actor is only damaged if its species matches the specified species. Default is "None".

src: the actor responsible for the damage (source). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without a source. Default is AAPTR_DEFAULT, which is the calling actor itself.

inflict: the actor doing the damage (inflictor). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without an inflictor (note that doing so renders both DMSS_FOILINVUL and DMSS_FOILBUDDHA ineffective). Default is AAPTR_DEFAULT, which is the calling actor itself.

Examples

This example creates a new item that when used increases movement and attack speed at the cost of 25 health, similar to the stimpacks in Starcraft.

```
actor StarcraftStimpack : CustomInventory
```

```
{
    +INVENTORY.INVBAR
    Tag "Stimpack"
    Inventory.Amount 1
    Inventory.MaxAmount 25
}
```

```
Inventory.PickupMessage "You found some stimpacks"  
Inventory.PickupSound "misc/i_pkup"  
Inventory.UseSound "misc/p_pkup"
```

```
States
```

```
{  
Spawn:  
    SSTM A -1  
    Stop
```

```
Use:  
    TNT1 A 0  
    {  
        if (health > 25) // Make sure it does not kill the player  
        {  
            A_DamageSelf(25);  
            A_GiveInventory("StimpackMoveGiver");  
            A_GiveInventory("StimpackAttackGiver");  
            return true;  
        }  
        return false;  
    }  
    Stop  
}
```

```
actor PowerStimpackMove : PowerSpeed  
{  
    +POWERSPEED.NOTRAIL  
}
```

```
actor StimpackMoveGiver : PowerupGiver  
{  
    Powerup.Duration -15  
    Powerup.Type "PowerStimpackMove"  
    Inventory.MaxAmount 0  
    +INVENTORY.AUTOACTIVATE  
}
```

```
actor StimpackAttackGiver : PowerupGiver  
{  
    Powerup.Duration -15  
    Powerup.Type "PowerDoubleFiringSpeed"  
    Inventory.MaxAmount 0  
    INVENTORY.AUTOACTIVATE  
}
```

A_DamageSiblings

```
void A_DamageSiblings (int amount [, name damagetype [, int flags [, class<Actor> filter [, name species [, int src [, int inflict]]]]]])
```

Usage

Damages the calling actor's sibling actors by the specified amount. Negative amounts heal them, instead. This function cannot damage player actors which are under the effect of god2, and they still survive with one point of health if under the effect of buddha2.

For the calling actor to be considered to have a "sibling" actor, its and the sibling actor's master fields should point to the same actor.

Parameters

amount: the amount of damage to inflict. Negative amounts heal. An amount of one million or higher is treated specially, and results in killing the actor regardless of health or any damage modifiers that may be in effect.

damagetype: the type of damage to inflict. Default is "None".

flags: the following flags can be combined using the | character between the constants names:

DMSS_FOILINVUL — the actor is damaged even if it is invulnerable. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_FOILBUDDHA — if the damage is enough to kill the actor, it dies even if it is under the effect of buddha. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_NOPROTECT — damage bypasses the damage-modifying capability of items in the actor's inventory. A protection powerup is an example of such items.

DMSS_NOFACTOR — damage bypasses the the actor's damage factors.

DMSS_AFFECTARMOR — damage does not bypass the damage-absorbing capability of items in the actor's inventory. An armor is an example of such items.

DMSS_KILL — inflicts an amount of damage which is equal to the sum of the actor's current health and amount, killing the actor under normal conditions. This flag bypasses the actor's damage factors and the damage-absorbing capability (but not the damage-modifying one) of items in the actor's inventory.

DMSS_EXFILTER — inverts the case of the class name filter; the actor is only damaged if its class name does not match the value passed to filter.

DMSS_EXSPECIES — inverts the case of the species filter; the actor is only damaged if its species does not match the value passed to species.

DMSS_EITHER — the actor is damaged if either its class name or species matches the values passed to filter and species, respectively.

DMSS_INFLECTORDMGTYPE — ignores the specified damage type, and instead, uses the damage type of the actor doing the damage (inflictor).

filter: the actor class to damage. The actor is only damaged if its class name matches the specified class. Default is null(ZScript only)/"None"(DECORATE only).

species: the actor species to damage. The actor is only damaged if its species matches the specified species. Default is "None".

src: the actor responsible for the damage (source). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without a source. Default is AAPTR_DEFAULT, which is the calling actor itself.

inflict: the actor doing the damage (inflictor). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without an inflictor (note that doing so renders both DMSS_FOILINVUL and DMSS_FOILBUDDHA ineffective). Default is AAPTR_DEFAULT, which is the calling actor itself.

Examples

Assuming this monster has alive siblings, they are damaged by it each time it is stunned by an attack with the "Curse" damage type.

```
actor SoldierImp : DoomImp
```

```
{  
    States
```

```
{
Pain.Curse:
TROO H 2
TROO H 2
{
    A_Pain();
    A_DamageSiblings(15);
}
Goto See
}
}
```

A_DamageTarget

```
void A_DamageTarget (int amount [, name damagetype [, int flags [, class<Actor> filter [, name species [, int src [, int inflict]]]]]])
```

Usage

Damages the calling actor's target actor by the specified amount. Negative amounts heal it, instead. This function cannot damage player actors which are under the effect of god2, and they still survive with one point of health if under the effect of buddha2.

Parameters

amount: the amount of damage to inflict. Negative amounts heal. An amount of one million or higher is treated specially, and results in killing the actor regardless of health or any damage modifiers that may be in effect.

damagetype: the type of damage to inflict. Default is "None".

flags: the following flags can be combined using the | character between the constants names:

DMSS_FOILINVUL — the actor is damaged even if it is invulnerable. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_FOILBUDDHA — if the damage is enough to kill the actor, it dies even if it is under the effect of buddha. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_NOPROTECT — damage bypasses the damage-modifying capability of items in the actor's inventory. A protection powerup is an example of such items.

DMSS_NOFACTOR — damage bypasses the the actor's damage factors.

DMSS_AFFECTARMOR — damage does not bypass the damage-absorbing capability of items in the actor's inventory. An armor is an example of such items.

DMSS_KILL — inflicts an amount of damage which is equal to the sum of the actor's current health and amount, killing the actor under normal conditions. This flag bypasses the actor's damage factors and the damage-absorbing capability (but not the damage-modifying one) of items in the actor's inventory.

DMSS_EXFILTER — inverts the case of the class name filter; the actor is only damaged if its class name does not match the value passed to filter.

DMSS_EXSPECIES — inverts the case of the species filter; the actor is only damaged if its species does not match the value passed to species.

DMSS_EITHER — the actor is damaged if either its class name or species matches the values passed to filter and species, respectively.

DMSS_INFLECTORDMGTYPE — ignores the specified damage type, and instead, uses the damage type of the actor doing the damage (inflictor).

filter: the actor class to damage. The actor is only damaged if its class name matches the specified class. Default is null(ZScript only)/"None"(DECORATE only).

species: the actor species to damage. The actor is only damaged if its species matches the specified species. Default is "None".

src: the actor responsible for the damage (source). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without a source. Default is AAPTR_DEFAULT, which is the calling actor itself.

inflict: the actor doing the damage (inflictor). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without an inflictor (note that doing so renders both DMSS_FOILINVUL and DMSS_FOILBUDDHA ineffective). Default is AAPTR_DEFAULT, which is the calling actor itself.

Examples

Taking advantage of the fact that a projectile's target actor is its shooter, this rather weird rocket damages its shooter, instead of performing a radius attack, if it hits a wall, ceiling or floor.

```
class OddRocket : Rocket
```

```
{  
    States
```

```
{
```

```
    Crash:
```

```
    Death.Extreme:
```

Goto Super::Death; // Use the parent class's own Death state sequence.

Death:

MISL B 8 Bright A_DamageTarget(10, flags: DMSS_AFFECTARMOR, src: AAPTR_TARGET);

MISL C 6 Bright;

MISL D 4 Bright;

Stop;

}

}

A_DamageTracer

Jump to navigationJump to search

```
void A_DamageTracer (int amount [, name damagetype [, int flags [, class<Actor> filter [, name species [, int src [, int inflict]]]]]])
```

Usage

Damages the calling actor's tracer actor by the specified amount. Negative amounts heal it, instead. This function cannot damage player actors which are under the effect of god2, and they still survive with one point of health if under the effect of buddha2.

Parameters

amount: the amount of damage to inflict. Negative amounts heal. An amount of one million or higher is treated specially, and results in killing the actor regardless of health or any damage modifiers that may be in effect.

damagetype: the type of damage to inflict. Default is "None".

flags: the following flags can be combined using the | character between the constants names:

DMSS_FOILINVUL — the actor is damaged even if it is invulnerable. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_FOILBUDDHA — if the damage is enough to kill the actor, it dies even if it is under the effect of buddha. For this flag to work, an inflictor is required (see inflict below). This flag is ignored if the actor is a player.

DMSS_NOPROTECT — damage bypasses the damage-modifying capability of items in the actor's inventory. A protection powerup is an example of such items.

DMSS_NOFACTOR — damage bypasses the the actor's damage factors.

DMSS_AFFECTARMOR — damage does not bypass the damage-absorbing capability of items in the actor's inventory. An armor is an example of such items.

DMSS_KILL — inflicts an amount of damage which is equal to the sum of the actor's current health and amount, killing the actor under normal conditions. This flag bypasses the actor's damage factors and the damage-absorbing capability (but not the damage-modifying one) of items in the actor's inventory.

DMSS_EXFILTER — inverts the case of the class name filter; the actor is only damaged if its class name does not match the value passed to filter.

DMSS_EXSPECIES — inverts the case of the species filter; the actor is only damaged if its species does not match the value passed to species.

DMSS_EITHER — the actor is damaged if either its class name or species matches the values passed to filter and species, respectively.

DMSS_INFLICTORDMGTYPE — ignores the specified damage type, and instead, uses the damage type of the actor doing the damage (inflictor).

filter: the actor class to damage. The actor is only damaged if its class name matches the specified class. Default is null(ZScript only)/"None"(DECORATE only).

species: the actor species to damage. The actor is only damaged if its species matches the specified species. Default is "None".

src: the actor responsible for the damage (source). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without a source. Default is AAPTR_DEFAULT, which is the calling actor itself.

inflict: the actor doing the damage (inflictor). This is an actor pointer. If this is set to AAPTR_NULL, the actor is damaged without an inflictor (note that doing so renders both DMSS_FOILINVUL and DMSS_FOILBUDDHA ineffective). Default is AAPTR_DEFAULT, which is the calling actor itself.

Examples

This revenant's missile causes damage to the actor it seeks (tracer) every second while in-flight.

```
actor CurseRevenant : Revenant
```

```
{
    HitObituary "%o was punched by a cursed revenant."
    Obituary "%o was killed by the presence of a cursed revenant's fireball."
```

States

```
{
```

Missile:

SKEL J 0 Bright A_FaceTarget

SKEL J 10 Bright A_FaceTarget

SKEL K 10 A_SpawnProjectile("CurseTracer", 40)

SKEL K 10 A_FaceTarget

Goto See

}

}

actor CurseTracer : RevenantTracer

{

States

{

Spawn:

FATB ABABABAB 4 Bright A_SeekerMissile(45, 90, SMF_PRECISE)

FATB A 3 Bright A_DamageTracer(1, "None", DMSS_NOPROTECT)

Loop

}

}

A_Die

A_Die [(str DamageType)]

Kills the calling actor if it is not already dead. This has only an effect if the actor has the SHOOTABLE or VULNERABLE flag set.

If DamageType is provided, it determines what type of damage is inflicted upon the actor to kill it. This could be used to determine which Death state the monster uses as it dies.

Examples

This imp's days are numbered... It will walk around and fire at you normally. But all the while walking on borrowed time, slowly gaining the inventory "DeathTimer" with every action it takes. When the timer reaches 20, the imp will simulate having a violent heart-attack, then trigger A_Die, forcing it to enter its death state, regardless what its health previously was. For added effect, A_Jump is used in its see state giving it a small chance to cringe with chest-pain once in a while.

ACTOR DyingImp : DoomImp

```
{
  States
  {
    See:
      TROO AABBCDD 3 A_Chase
      TROO A 0 A_GiveInventory("DeathTimer", 1) // A dummy inventory for tracking how long the imp has been
searching.
      TROO A 0 A_JumpIfInventory("DeathTimer", 20, "HeartAttack") // Jump to the HeartAttack state when the timer
reaches 20
      TROO A 0 A_Jump(45, "Pain")
    Loop
  Melee:
  Missile:
      TROO EF 8 A_FaceTarget
      TROO G 6 A_TroopAttack
      TROO A 0 A_GiveInventory("DeathTimer", 1) // Attacking the player only speeds up the process
      TROO A 0 A_JumpIfInventory("DeathTimer", 20, "HeartAttack")
      Goto See
  Pain:
      TROO H 2
      TROO H 2 A_Pain
      Goto See
  HeartAttack:
      TROO H 2
      TROO HHHH 20 A_Pain
      TROO H 2 A_Die
  }
}
```

This is the inventory item that the monster uses.

ACTOR DeathTimer : Inventory

```
{
  Inventory.MaxAmount 20
  Inventory.Amount 1
}
```


A_FaceTarget

```
void A_FaceTarget [(double max_turn [, double max_pitch [, double ang_offset [, double pitch_offset [, int flags [,
double z_ofs]]]]]]
void A_FaceTracer [(double max_turn [, double max_pitch [, double ang_offset [, double pitch_offset [, int flags [,
double z_ofs]]]]]]
void A_FaceMaster [(double max_turn [, double max_pitch [, double ang_offset [, double pitch_offset [, int flags [,
double z_ofs]]]]]]]
```

Usage

Change the calling actor's angle to face a specific actor, depending on the chosen function.

A_FaceTarget changes the angle to face the calling actor's target.

A_FaceTracer changes the angle to face the calling actor's tracer.

A_FaceMaster changes the angle to face the calling actor's master.

Parameters

max_turn: The maximum turn angle; the calling actor cannot turn by more than said angle, however the SHADOW flag has no effect in such case. A value of 0 is interpreted as unlimited angle. Default is 0.

max_pitch: If specified to a value no greater than 180, then the calling actor's pitch is adjusted up to said value to face the actor. A value of 0 is interpreted as unlimited angle; and technically a pitch change will never be greater than 180 degrees. It will also aim at the actor's feet when set to 0. Default is 270, which means its disabled.

ang_offset: Specifies the amount of degrees to offset the actor's angle by. Positive values turn it left, while negative values turn it right. This is factored in after max_turn is performed. Due to limitations, distance is not factored in. Default is 0.

pitch_offset: Adjusts the pitch by this many degrees after max_pitch has been taken into account. Default is 0.

flags: Customizes the behavior of the function. Multiple flags can be combined by using the bitwise OR operator (|) between the constant names:

FAF_BOTTOM "Aim for the bottom of the actor, otherwise known as the raw Z position. Whenever max_pitch is taken into account, it will aim towards the actor's feet + 32 units above. This flag disables adding that 32 units.

FAF_MIDDLE "Aim for the middle of the actor (z position + height / 2).

FAF_TOP "Aim for the top of the actor (z position + height).

Note that all of these flags are taken into account first before anything else.

Default is 0.

z_ofs: Offsets the z position distance of the actor to face by this amount. Unlike pitch_offset, this takes into account how far away the actor is at all times. Default is 0.

Examples

Almost every monster uses A_FaceTarget before it shoots at its target.

Missile:

```
TROO EF 8 A_FaceTarget
TROO G 6 A_TroopAttack // See DoomImpBall
Goto See
```

This lost soul homes in on its target by calling A_FaceTarget and A_SkullAttack repeatedly.

actor HomingLostSoul : LostSoul

```
{
    States
    {
        Missile:
            SKUL C 10 Bright A_FaceTarget
            SKUL D 4 Bright A_SkullAttack
            SKUL CD 4 Bright
```

Goto Missile+1

}
}

A_FastChase

A_FastChase

(no parameters)

Note: This function has been superseded by A_Chase, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

This function makes the calling actor advance on its target while randomly attempting to strafe left or right. The actor may enter an attack state (if it has one) at any time. As this function maintains a certain distance between the calling actor and its target, it is not always advisable to use it for actors who only possess a melee attack.

The three "player bosses" in Hexen use this function to obtain their distinctive movement.

Note that the calling actor will ignore their MaxDropOffHeight property when strafing, and so may strafe off cliffs or into pits from which they cannot escape. Setting the NODROPOFF flag will prevent this from happening.

Examples

Here is an example of a cacodemon which tries to dodge attacks and strafe around the player.

ACTOR Smartcaco : Cacodemon

```
{
  States
  {
    See:
      HEAD A 2 A_FastChase
    Loop
  }
}
```

A_KillChildren

A_KillChildren [(string damagetype [, int flags [, string filter [, string species [, int src [, int inflict]]]]))]

Usage

Called by monsters to kill all monsters spawned by the caller or by a projectile spawned by the caller of this function. Currently the only function that sets the necessary information is A_SpawnItemEx. Optionally, a damagetype parameter can be given. Note that damage inflicted by this codepointer is not affected by damage factors. Monsters with the INVULNERABLE flag are unaffected, but monsters with the NODAMAGE flag can still enter their pain state based upon damagetype.

Monsters spawned with A_SpawnProjectile are not affected by this. A_SpawnProjectile was never designed to spawn monsters.

Parameters

damagetype - If the actor dies, the actor will enter a death state based on damagetype if present (or pain state if using NODAMAGE).

flags - The following flags can be combined using the | character between name constants:

KILLS_FOILINVUL - Kills monsters and/or missiles that have the INVULNERABLE flag.

KILLS_KILLMISSILES - Causes missiles that are affected to enter their death state. Note that this follows the INVULNERABLE, BUDDHA and NODAMAGE rules for monsters.

KILLS_NOMONSTERS - Don't target monsters with this function. Alone, it makes the function do nothing, but can be combined with KILLS_KILLMISSILES to only affect missiles.

KILLS_FOILBUDDHA — the buddha effect is ignored when attempting to kill the actor.

KILLS_EXFILTER — inverts the case of the class name filter; the calling actor's children are only killed if their class name does not match the value passed to filter.

KILLS_EXSPECIES — inverts the case of the species filter; the calling actor's children are only killed if their species does not match the value passed to species.

KILLS_EITHER — the calling actor's children are killed if either of their class name or species matches the values passed to filter and species, respectively.

filter: the actor class to kill. The calling actor's children are only killed if their class name matches the specified filter class. Default is "None". If left to "None", it will not apply a filter.

species: the actor species to kill. The calling actor's children are only killed if their species matches the specified species filter. Default is "None". If left to "None", it will not apply a filter.

src: Indicates the actor pointer responsible for dealing the damage. A monster dealing the damage should use AAPTR_DEFAULT, and missiles should use AAPTR_TARGET (so monsters can identify missiles belonging to their owners and give proper credit for the kill). Default is AAPTR_DEFAULT.

inflict: The actor doing the actual damage. By changing this, the actor's flags upon the pointed actor are taken into account instead of the calling actor's own.

(Note: Unlike monsters, missiles were never meant to have damagetype specific death states and will ignore it.)

Examples

The following is a variant of the doom imp, a monster that spawns clones of itself to attack you. When killed, it will trigger A_KillChildren, causing all of the spawned clones of it to die simultaneously.

```
ACTOR VoodooLeaderImp : DoomImp
```

```
{
    Game Doom
    SpawnID 5
    Health 100
    Mass 1000
    PainChance 255
    States
    {
        Missile:
```

TROO EF 8 A_FaceTarget

TROO G 6 A_SpawnItemEx("SoldierImp", 50, 50, 60, 0, 0, 0, 0, SXF_SETMASTER)

Goto See

Pain:

TROO H 2

TROO H 1 A_Pain

TROO H 1 A_DamageChildren(1, "Voodoo")

Goto See

Death:

TROO I 8

TROO J 8 A_Scream

TROO K 6 A_KillChildren

TROO L 6 A_NoBlocking

TROO M -1

Stop

XDeath:

TROO N 5

TROO O 5 A_XScream

TROO P 5 A_KillChildren

TROO Q 5 A_NoBlocking

TROO RST 5

TROO U -1

Stop

}

}

A_KillMaster

Jump to navigationJump to search

A_KillMaster [(string damagetype [, int flags [, string filter [, string species [, int src [, int inflict]]]]]]]

Usage

Called by monsters to kill the monster that spawned this one. Currently the only function that sets the necessary information is A_SpawnItemEx. Optionally, a damagetype parameter can be given. Note that damage inflicted by this codepointer is not affected by damage factors.

Monsters spawned with A_SpawnProjectile are not affected by this. A_SpawnProjectile was never designed to spawn monsters.

Parameters

damagetype - If the actor dies, the actor will enter a death state based on damagetype if present (or pain state if using NODAMAGE).

flags - The following flags can be combined using the | character between name constants:

KILS_FOILINVUL - Kills monsters and/or missiles that have the INVULNERABLE flag.

KILS_KILLMISSILES - Causes missiles that are affected to enter their death state. Note that this follows the INVULNERABLE and NODAMAGE rules for monsters.

KILS_NOMONSTERS - Don't target monsters with this function. Alone, it makes the function do nothing, but can be combined with KILS_KILLMISSILES to only affect missiles.

KILS_FOILBUDDHA — the buddha effect is ignored when attempting to kill the actor.

KILS_EXFILTER — inverts the case of the class name filter; the calling actor's master is only killed if its class name does not match the value passed to filter.

KILS_EXSPECIES — inverts the case of the species filter; the calling actor's master is only killed if its species does not match the value passed to species.

KILS_EITHER — the calling actor's master is killed if either of its class name or species matches the values passed to filter and species, respectively.

filter: the actor class to kill. The calling actor's master is only killed if its class name matches the specified filter class. Default is "None".

species: the actor species to kill. The calling actor's master is only killed if its species matches the specified species filter. Default is "None".

src: Indicates the actor pointer responsible for dealing the damage. A monster dealing the damage should use AAPTR_DEFAULT, and missiles should use AAPTR_TARGET (so monsters can identify missiles belonging to their owners and give proper credit for the kill). Default is AAPTR_DEFAULT.

inflict: The actor doing the actual damage. By changing this, the actor's flags upon the pointed actor are taken into account instead of the calling actor's own.

Examples

The following is a variant of the doom imp, a monster spawned by a different monster with the master/child flag. If it gets killed with an attack with the "Curse" damage type, its death state will trigger A_KillMaster, killing the monster who spawned it.

ACTOR SoldierImp : DoomImp

```
{
  States
  {
    Pain.Voodoo:
      TROO H 2 A_Pain
      TROO H 50
      Goto See
    Pain.Curse:
      TROO H 2 A_Pain
      TROO H 2 A_DamageSiblings(15)
      Goto See
    Death:
      TROO I 8 A_DamageMaster(-25)
```


TROO J 8 A_Scream
TROO K 6
TROO L 6 A_NoBlocking
TROO M -1
Stop

Death.Curse:

TROO I 8 A_KillMaster
TROO J 8 A_Scream
TROO K 6
TROO L 6 A_NoBlocking
TROO M -1
Stop

}

}

A_KillSiblings

Jump to navigationJump to search

A_KillSiblings [(string damagetype [, int flags [, string filter [, string species [, int src [, int inflict]]]])]

Usage

Called by monsters to kill the monsters that were spawned by the same master (other than the caller). Currently the only function that sets the necessary information is A_SpawnItemEx. Optionally, a damagetype parameter can be given. Note that damage inflicted by this codepointer is not affected by damage factors.

Monsters spawned with A_SpawnProjectile are not affected by this. A_SpawnProjectile was never designed to spawn monsters.

Parameters

damagetype - If the actor dies, the actor will enter a death state based on damagetype if present (or pain state if using NODAMAGE).

flags - The following flags can be combined using the | character between name constants:

KILLS_FOILINVUL - Kills monsters and/or missiles that have the INVULNERABLE flag.

KILLS_KILLMISSILES - Causes missiles that are affected to enter their death state. Note that this follows the INVULNERABLE and NODAMAGE rules for monsters.

KILLS_NOMONSTERS - Don't target monsters with this function. Alone, it makes the function do nothing, but can be combined with KILLS_KILLMISSILES to only affect missiles.

KILLS_FOILBUDDHA — the buddha effect is ignored when attempting to kill the actor.

KILLS_EXFILTER — inverts the case of the class name filter; the calling actor's siblings are only killed if their class name does not match the value passed to filter.

KILLS_EXSPECIES — inverts the case of the species filter; the calling actor's siblings are only killed if their species does not match the value passed to species.

KILLS_EITHER — the calling actor's siblings are killed if either of their class name or species matches the values passed to filter and species, respectively.

filter: the actor class to kill. The calling actor's siblings are only killed if their class name matches the specified filter class. Default is "None".

species: the actor species to kill. The calling actor's siblings are only killed if their species matches the specified species filter. Default is "None".

src: Indicates the actor pointer responsible for dealing the damage. A monster dealing the damage should use AAPTR_DEFAULT, and missiles should use AAPTR_TARGET (so monsters can identify missiles belonging to their owners and give proper credit for the kill). Default is AAPTR_DEFAULT.

inflict: The actor doing the actual damage. By changing this, the actor's flags upon the pointed actor are taken into account instead of the calling actor's own.

Examples

The following is a variant of the doom imp, a monster spawned by a different monster using the master/child flag. If it gets killed with an attack with the "Curse" damage type, its death state triggers A_KillSiblings, killing all of the other monsters spawned by the same master.

ACTOR SoldierImp : DoomImp

```
{
  States
  {
    Pain.Voodoo:
      TROO H 2 A_Pain
      TROO H 50
      Goto See
    Pain.Curse:
      TROO H 2 A_Pain
      TROO H 2 A_DamageSiblings(15)
      Goto See
    Death:
      TROO I 8 A_DamageMaster(-25)
```

TROO J 8 A_Scream
TROO K 6
TROO L 6 A_NoBlocking
TROO M -1
Stop

Death.Curse:

TROO I 8 A_KillMaster
TROO J 8 A_Scream
TROO K 6 A_KillSiblings
TROO L 6 A_NoBlocking
TROO M -1
Stop

}

}

A_KillTarget

A_KillTarget [(string damagetype [, int flags [, string filter [, string species [, int src [, int inflict]]]]]]]

Usage

Kills the calling actor's target. Optionally, a damage type can be given. Note that damage inflicted by this codepointer is not affected by damage factors.

Parameters

damagetype: if the actor dies, the actor will enter a death state based on the specified damage type if present (or pain state if using NODAMAGE). Default is "None"

flags: the following flags can be combined using the | character between name constants:

KILLS_FOILINVUL - kills monsters and/or missiles that have the INVULNERABLE flag.

KILLS_KILLMISSILES - causes missiles that are affected to enter their death state. Note that this follows the INVULNERABLE and NODAMAGE rules for monsters.

KILLS_NOMONSTERS - do not target monsters with this function. Alone, it makes the function do nothing, but can be combined with KILLS_KILLMISSILES to only affect missiles.

KILLS_FOILBUDDHA — the buddha effect is ignored when attempting to kill the actor.

KILLS_EXFILTER — inverts the case of the class name filter; the calling actor's target is only killed if its class name does not match the value passed to filter.

KILLS_EXSPECIES — inverts the case of the species filter; the calling actor's target is only killed if its species does not match the value passed to species.

KILLS_EITHER — the calling actor's target is killed if either of its class name or species matches the values passed to filter and species, respectively.

filter: the actor class to kill. The calling actor's target is only killed if its class name matches the specified filter class. Default is "None".

species: the actor species to kill. The calling actor's target is only killed if its species matches the specified species filter. Default is "None".

src: Indicates the actor pointer responsible for dealing the damage. A monster dealing the damage should use AAPTR_DEFAULT, and missiles should use AAPTR_TARGET (so monsters can identify missiles belonging to their owners and give proper credit for the kill). Default is AAPTR_DEFAULT.

inflict: The actor doing the actual damage. By changing this, the actor's flags upon the pointed actor are taken into account instead of the calling actor's own.

Examples

This cyberdemon kills target by Psi-wave with chance less than 1%:

Actor PsiCyber4ZDoomWiki: Cyberdemon replaces Cyberdemon

```
{
States
{
Missile:
CYBR E 0 A_Jump( 1, "PsiKill" )
CYBR E 6 A_FaceTarget
CYBR F 12 A_CyberAttack
CYBR E 12 A_FaceTarget
CYBR F 12 A_CyberAttack
CYBR E 12 A_FaceTarget
CYBR F 12 A_CyberAttack
Goto See
PsiKill:
CYBR E 0 A_PlaySound( "cyber/sight", CHAN_VOICE )
CYBR E 12 Bright A_FaceTarget
CYBR E 12 A_FaceTarget
CYBR E 8 Bright A_FaceTarget
CYBR E 8 A_FaceTarget
CYBR E 4 Bright A_FaceTarget
}
```

```
CYBR E 4 A_FaceTarget  
CYBR E 2 Bright A_FaceTarget  
CYBR E 2 A_FaceTarget  
CYBR F 20 A_KillTarget( "PsiDmg" )  
Goto See
```

```
}  
}  
}
```

A_KillTracer

A_KillTracer [(string damagetype [, int flags [, string filter [, string species [, int src [, int inflict]]]]]]]

Usage

Kills the calling actor's tracer. Optionally, a damage type can be given. Note that damage inflicted by this codepointer is not affected by damage factors.

Parameters

damagetype: if the actor dies, the actor will enter a death state based on the specified damage type if present (or pain state if using NODAMAGE). Default is "None".

flags: the following flags can be combined using the | character between name constants:

KILLS_FOILINVUL - kills monsters and/or missiles that have the INVULNERABLE flag.

KILLS_KILLMISSILES - causes missiles that are affected to enter their death state. Note that this follows the INVULNERABLE and NODAMAGE rules for monsters.

KILLS_NOMONSTERS - do not target monsters with this function. Alone, it makes the function do nothing, but can be combined with KILLS_KILLMISSILES to only affect missiles.

KILLS_FOILBUDDHA — the buddha effect is ignored when attempting to kill the actor.

KILLS_EXFILTER — inverts the case of the class name filter; the calling actor's tracer is only killed if its class name does not match the value passed to filter.

KILLS_EXSPECIES — inverts the case of the species filter; the calling actor's tracer is only killed if its species does not match the value passed to species.

KILLS_EITHER — the calling actor's tracer is killed if either of its class name or species matches the values passed to filter and species, respectively.

filter: the actor class to kill. The calling actor's tracer is only killed if its class name matches the specified filter class. Default is "None".

species: the actor species to kill. The calling actor's tracer is only killed if its species matches the specified species filter. Default is "None".

src: Indicates the actor pointer responsible for dealing the damage. A monster dealing the damage should use AAPTR_DEFAULT, and missiles should use AAPTR_TARGET (so monsters can identify missiles belonging to their owners and give proper credit for the kill). Default is AAPTR_DEFAULT.

inflict: The actor doing the actual damage. By changing this, the actor's flags upon the pointed actor are taken into account instead of the calling actor's own.

Examples

This is a modified revenant, whos projectile will instantly freeze the player to death if it flies for around 5 seconds, using A_KillTracer, with the "Ice" damage type.

ACTOR Petriphant : Revenant

```
{
  Translation "Ice"
  HitObituary "%o was punched by a Petriphant."
  Obituary "%o was petrified by a Petriphant's fireball."
  States
  {
    Missile:
      SKEL J 0 Bright A_FaceTarget
      SKEL J 10 Bright A_FaceTarget
      SKEL K 10 A_SpawnProjectile ("StoneTracer", 40)
      SKEL K 10 A_FaceTarget
      Goto See
  }
}
```

ACTOR StoneTracer : RevenantTracer

```
{
```

Translation "Ice"

DamageType Ice

States

{

Spawn:

FATB ABABABABA 4 Bright A_SeekerMissile (45, 90, SMF_PRECISE)

FATB ABABABABA 4 Bright A_SeekerMissile (45, 90, SMF_PRECISE)

FATB ABABABABA 4 Bright A_SeekerMissile (45, 90, SMF_PRECISE)

FATB ABABABABA 4 Bright A_SeekerMissile (45, 90, SMF_PRECISE)

FATB ABABABABA 4 Bright A_SeekerMissile (45, 90, SMF_PRECISE)

FATB A 3 Bright A_KillTracer ("Ice")

Loop

Death:

FBXP A 8 Bright

FBXP B 6 Bright

FBXP C 4 Bright

Stop

}

}

A_Look

(no parameters)

Note: This function has been superseded by A_LookEx, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

Looks for players or other attackable actors in the game. If it finds a target, it enters its "See" state.

Examples

Almost every monster in Doom uses A_Look. Here is code taken from the Imp monster.

Spawn:

```
TROO AB 10 A_Look
```

loop

A_Look is called on every animation frame in the "Spawn" state to check for players.

A_Look2

(no parameters)

Looks for players or other attackable actors in the game. This function is similar to A_Look but it only reacts to sound. Just seeing the player will not wake up a monster using this. This function must be used in the idle states of a monster.

The three states after this function call are reserved. You must include these states because the function jumps to the states following the call. A_Look2 calls them randomly to animate actors using this function. The third of these states is only called when the STANDSTILL flag is not set.

If you do not need that, A_TurretLook is the better solution.

Examples

Some of Strife's actors use this to check for gunfire. Here is code taken from the Peasant.

Spawn:

```
PEAS A 10 A_Look2
```

```
Loop
```

As you can see it is very similar to the A_Look example.

A_LookEx

A_LookEx (int flags, float minseedist, float maxseedist, float maxheardist, double fov, state label)

Usage

This is a customizable version of A_Look, allowing for enemies which can only see or hear a target under given conditions. Just like with A_Look, if you set this on a friendly monster, it will make the monster enter the specified state even if it does not have a target. The STANDSTILL actor flag can be used (and is pretty much required) with this function to prevent this behavior.

Note: In order for A_LookEx to function correctly, the actor calling it must have either the seestate, or a "See" state defined.

Parameters

flags: Controls miscellaneous aspects of the monster's behavior while looking for targets. Use | to use multiple flags at once.

LOF_NOSIGHTCHECK — do not process sight checks (makes monster blind).

LOF_NOSOUNDCHECK — do not process sound target checks (makes monster deaf).

Note: this is not the same as using AMBUSH: When the AMBUSH flag is specified for a monster, if the player makes a sound in the same room (and in range of the monster if maxheardist is set), the monster will react to the player if they walk within the monster's current line of sight, regardless of the direction it is facing (FOV is ignored). This flag will make the monster completely deaf, preventing it from reacting to the player at all, regardless of the sounds they make.

LOF_DONTCHASEGOAL — do not leave to chase after a goal from this state, even if the actor has one. This can be used to prevent a monster from breaking from its idle animations to walk over to a goal.

LOF_NOSEESOUND — do not play the actor's sight sound if it wakes up from this state.

LOF_FULLVOLSEESOUND — Play the see sound globally at full volume (like a boss).

LOF_NOJUMP — acquire a target, but do not jump to any other state, allows to check the target and jump manually into the See state based on conditions without interrupting the idle animation.

minseedist: the minimum sight distance of the monster in map units. If this is greater than 0, the monster will not see a player who is closer than this. If this is set, the monster also will not wake up if touched by the player (so it can be set smaller than the radius to prevent the monster from reacting if the player comes up from behind).

maxseedist: the maximum sight distance of the monster, in map units. It will not see any players farther away than this. 0 means unlimited (as in Doom). Note that friendly monsters have a hard cap of 1280 map units for performance reasons.

maxheardist: the maximum hearing range of the monster, it will not react to sounds from players farther away than this. 0 means unlimited (as in Doom).

fov: the field of view of the monster. A narrower FOV means the player has to be more centered in front of the monster for it to notice the player. Setting this to 0 uses the old default of 180 degrees, while smaller values make the FOV narrower and larger values wider. 360 means look in all directions and is the same as LOOKALLAROUND with an FOV of 0.

label: The state the monster will jump to if it acquires a target. Default is "See".

Examples

The actor in the following example is totally deaf, and can not see targets more than 256 map units away. It jumps to the custom state "WakeUp" upon acquiring a valid target.

```
actor ImpairedZombie : ZombieMan
{
    states
    {
        Spawn:
            POSS A 1 A_LookEx(LOF_NOSOUNDCHECK, 0, 256, 0, 0, "WakeUp")
        loop
        WakeUp:
            POSS A 1 A_AlertMonsters
```

```
    goto See
  }
}
```

The following actor, in addition to being totally deaf, has a limited field of vision (30 degrees) and will not detect targets closer than 32 map units to it. Its seesound is played at full volume upon waking up.

```
actor SecuritySoul : LostSoul
{
  SeeSound "misc/alarm"
  states
  {
    Spawn:
      SKUL A 1 A_LookEx(LOF_NOSOUNDCHECK | LOF_FULLVOLSEESOUND, 32, 0, 0, 30, "See")
      loop
  }
}
```

Note that in both these cases, the actor will regain full hearing and vision upon waking up, like a regular Doom monster. To check FOV while currently chasing a target, see `A_JumpIfTargetInLOS`. To check for distance from a current target, see `A_JumpIfCloser`.

A_RaiseChildren

A_RaiseChildren [(int flags)]

Usage

Resurrects the calling actor's children. Currently the only function that sets the necessary information is A_SpawnItemEx.

Parameters

flags: The following flags can be combined using the bit-wise OR operator (|):

RF_TRANSFERFRIENDLINESS “ the resurrected actors will change their affiliation to match that of the calling actor.

RF_NOCHECKPOSITION “ resurrect the actor without checking for room.

Monsters spawned with A_SpawnProjectile are not affected by this. A_SpawnProjectile was never designed to spawn monsters.

Examples

The following is a variant of the doom imp, a monster that spawns clones of itself. Its missile state uses A_Jump to decide on using one of two different attacks. Either spawning a clone, or triggering A_RaiseChildren, reviving all clones that were already spawned and had died.

ACTOR VoodooLeaderImp : DoomImp

```
{
  Game Doom
  SpawnID 5
  Health 100
  Mass 1000
  PainChance 255
  States
  {
    Missile:
      TROO G 0 A_Jump(256, "Missile1", "Missile2")
    Missile1:
      TROO EF 8 A_FaceTarget
      TROO G 6 A_SpawnItemEx("SoldierImp", 50, 50, 60, 0, 0, 0, 0, SXF_SETMASTER)
      Goto See
    Missile2:
      TROO EF 8 A_FaceTarget
      TROO G 6 A_RaiseChildren
      Goto See
  }
}
```

A_RaiseMaster

A_RaiseMaster [(int flags)]

Usage

Resurrects the calling actor's master. Currently the only function that sets the necessary information is A_SpawnItemEx.

Parameters

flags: The following flags can be combined using the bit-wise OR operator (|):

RF_TRANSFERFRIENDLINESS “ the resurrected actors will change their affiliation to match that of the calling actor.

RF_NOCHECKPOSITION “ resurrect the actor without checking for room.

Monsters spawned with A_SpawnProjectile are not affected by this. A_SpawnProjectile was never designed to spawn monsters.

Examples

The following is a variant of the doom imp, a monster whose sole purpose is to revive its master through its death. Upon dying, its death state triggers A_RaiseMaster, reviving the monster who spawned it with the master/child flag.

ACTOR SacrificialImp : DoomImp

```
{
  States
  {
    Death:
      TROO I 8
      TROO J 8 A_Scream
      TROO K 6 A_RaiseMaster
      TROO L 6 A_NoBlocking
      TROO M -1
      Stop
  }
}
```

A_RaiseSelf

bool A_RaiseSelf [(int flags)]

Usage

Resurrects the calling actor. The actor cannot be resurrected if:

it is not a corpse.

the duration of its current state is not -1 and said state lacks the CanRaise keyword.

it is a player.

lacks the Raise state.

there is not enough room for it to raise (this check is bypassed if the RF_NOCHECKPOSITION flag is passed to the function).

CanResurrect resolves to failure.

Parameters

flags: the following flags can be combined using the bit-wise OR operator (|):

RF_NOCHECKPOSITION "resurrect the actor without checking for room.

Return value

The function returns true if the actor is resurrected successfully, otherwise it returns false.

Examples

This undead warrior keeps coming back to life as long it does not sustain heavy damage.

```
class ResKnight : Knight
{
    States
    {
    Death:
        KNIG I 6;
        KNIG J 6 A_Scream;
        KNIG K 6;
        KNIG L 6 A_NoBlocking;
        KNIG MN 6;
        KNIG O 105;
        KNIG O -1
        {
            if (health > -100)
            {
                A_RaiseSelf();
            }
        }
        Stop;

    Raise:
        KNIG NMLKJI 5;
        Goto See;
    }
}
```

A_RaiseSiblings

A_RaiseSiblings [(int flags)]

Usage

Resurrects the calling actor's siblings. Currently the only function that sets the necessary information is A_SpawnItemEx.

Parameters

flags: The following flags can be combined using the bit-wise OR operator (|):

RF_TRANSFERFRIENDLINESS " the resurrected actors will change their affiliation to match that of the calling actor.

RF_NOCHECKPOSITION " resurrect the actors without checking for room.

Monsters spawned with A_SpawnProjectile are not affected by this. A_SpawnProjectile was never designed to spawn monsters.

Examples

The following is a variant of the doom imp, a monster whose sole purpose is to revive its master and siblings through its own death. When killed, its death state triggers A_RaiseSiblings, reviving all of the other monsters who were spawned by its master.

ACTOR SacrificialImp : DoomImp

```
{
  States
  {
    Death:
      TROO I 8
      TROO J 8 A_Scream
      TROO K 6 A_RaiseMaster
      TROO K 0 A_RaiseSiblings
      TROO L 6 A_NoBlocking
      TROO M -1
      Stop
  }
}
```

A_Remove

A_Remove (int pointer [, int flags [, string filter [, string species]]])

Usage

Removes an actor based upon which actor pointer is chosen and the flags.

Parameters

pointers: the following actor pointers are valid:

AAPTR_DEFAULT - Removes the calling actor itself.

AAPTR_TARGET - Removes the actor's target.

AAPTR_MASTER - Removes the actor's master.

AAPTR_TRACER - Removes the actor's tracer.

flags: the following flags can be combined using the | character between name constants:

RMVF_MISSILES - Allows removal of missiles associated with the pointer.

RMVF_NOMONSTERS - The function will not remove monsters and simply skip them. By default, before the introduction of assigning masters to missiles, similar functions would only work on monsters. To ensure consistency, only monsters are allowed for removal by default.

RMVF_MISC - Allows removal of things that are neither missile nor monster.

RMVF_EVERYTHING - Overrides all other flags. Disables discrimination and removes the actor of any type regardless of what it is.

RMVF_EXFILTER " " inverts the case of the class name filter; the pointed to actor is only removed if its class name does not match the value passed to filter.

RMVF_EXSPECIES " " inverts the case of the species filter; the pointed to actor is only removed if its species does not match the value passed to species.

RMVF_EITHER " " the pointed to actor is removed if either of its class name or species matches the values passed to filter and species, respectively.

filter: the actor class to remove. The pointed to actor is only removed if its class name matches the specified filter class. Default is "None".

species: the actor species to remove. The pointed to actor is only removed if its species matches the specified species filter. Default is "None".

A_RemoveChildren

Jump to navigationJump to search

A_RemoveChildren [(bool all [, int flags [, string filter [, string species]])]]

Usage

Called by monsters to completely remove all monsters spawned by the caller or by a projectile spawned by the caller of this function. If all is false or not specified, only dead children are removed. Currently the only function that sets the necessary information is A_SpawnItemEx.

Note that with SXF_ORIGINATOR, projectiles can create its own children, meaning they will not be attached to their projectiles masters.

Monsters spawned with A_SpawnProjectile are not affected by this. A_SpawnProjectile was never designed to spawn monsters.

This function can target actors that are non-monsters, but only by using flags.

Parameters

all: if this is false, only dead children are removed. Otherwise, if true, all children are removed, dead and alive. Default is false.

flags: the following flags can be combined using the | character between name constants:

RMVF_MISSILES - Allows removal of missiles associated with the pointer.

RMVF_NOMONSTERS - The function will not remove monsters and simply skip them. By default, before the introduction of assigning masters to missiles, similar functions would only work on monsters. To ensure consistency, only monsters are allowed for removal by default.

RMVF_MISC - Allows removal of things that are neither missile nor monster.

RMVF_EVERYTHING - Overrides all other flags. Disables discrimination and removes the actor of any type regardless of what it is.

RMVF_EXFILTER " " inverts the case of the class name filter; the calling actor's children are only removed if their class name does not match the value passed to filter.

RMVF_EXSPECIES " " inverts the case of the species filter; the calling actor's children are only removed if their species does not match the value passed to species.

RMVF_EITHER " " the calling actor's children are removed if either of their class name or species matches the values passed to filter and species, respectively.

filter: the actor class to remove. The calling actor's children are only removed if their class name matches the specified filter class. Default is "None".

species: the actor species to remove. The calling actor's children are only removed if their species matches the specified species filter. Default is "None".

Examples

The following is a variant on the doom imp, a monster that spawns clones of itself to attack you. Upon being killed, its death state triggers A_RemoveChildren, removing all of the dead clones it had spawned, before killing the living ones using A_KillChildren. This is so that the fresh corpses of the clones may still be revived somehow by a different monster.

ACTOR VoodooLeaderImp : DoomImp

```
{
    Game Doom
    SpawnID 5
    Health 100
    Mass 1000
    PainChance 255
    States
    {
```

Missile:

TROO G 0 A_Jump(256, "Missile1", "Missile2")

Missile1:

TROO EF 8 A_FaceTarget

TROO G 6 A_SpawnItemEx("SoldierImp", 50, 50, 60, 0, 0, 0, 0, SXF_SETMASTER)

Goto See

Missile2:

TROO EF 8 A_FaceTarget

TROO G 6 A_RaiseChildren

Goto See

Pain:

TROO H 2

TROO H 1 A_Pain

TROO H 1 A_DamageChildren(1, "Voodoo")

Goto See

Death:

TROO I 8 A_RemoveChildren(FALSE)

TROO J 8 A_Scream

TROO K 6 A_KillChildren

TROO L 6 A_NoBlocking

TROO M -1

Stop

XDeath:

TROO N 5 A_RemoveChildren(FALSE)

TROO O 5 A_XScream

TROO P 5 A_KillChildren

TROO Q 5 A_NoBlocking

TROO RST 5

TROO U -1

Stop

}

}

A_RemoveMaster

Jump to navigationJump to search

A_RemoveMaster [(int flags [, string filter [, string species]])]

Usage

Called by monsters to completely remove the monster that spawned this one. Currently the only function that sets the necessary information is A_SpawnItemEx.

Monsters spawned with A_SpawnProjectile are not affected by this. A_SpawnProjectile was never designed to spawn monsters.

This function can target actors that are non-monsters, but only by using flags.

Parameters

flags: the following flags can be combined using the | character between name constants:

RMVF_MISSILES - Allows removal of missiles associated with the pointer.

RMVF_NOMONSTERS - The function will not remove monsters and simply skip them. By default, before the introduction of assigning masters to missiles, similar functions would only work on monsters. To ensure consistency, only monsters are allowed for removal by default.

RMVF_MISC - Allows removal of things that are neither missile nor monster.

RMVF_EVERYTHING - Overrides all other flags. Disables discrimination and removes the actor of any type regardless of what it is.

RMVF_EXFILTER " " inverts the case of the class name filter; the calling actor's master is only removed if its class name does not match the value passed to filter.

RMVF_EXSPECIES " " inverts the case of the species filter; the calling actor's master is only removed if its species does not match the value passed to species.

RMVF_EITHER " " the calling actor's master is removed if either of its class name or species matches the values passed to filter and species, respectively.

filter: the actor class to remove. The calling actor's master is only removed if its class name matches the specified filter class. Default is "None".

species: the actor species to remove. The calling actor's master is only removed if its species matches the specified species filter. Default is "None".

Examples

The following is a variant on the doom imp, a monster spawned by a different monster using the master/child flag. If it dies from the damage type "Banish", its death state triggers A_RemoveMaster, completely removing the monster who spawned it.

ACTOR SoldierImp : DoomImp

```
{
  States
  {
    Pain.Voodoo:
      TROO H 2 A_Pain
      TROO H 50
      Goto See
    Pain.Curse:
      TROO H 2 A_Pain
      TROO H 2 A_DamageSiblings(15)
      Goto See
    Death:
      TROO I 8 A_DamageMaster(-25)
      TROO J 8 A_Scream
      TROO K 6
      TROO L 6 A_NoBlocking
```

```
TROO M -1
Stop
Death.Curse:
TROO I 8 A_KillMaster
TROO J 8 A_Scream
TROO K 6 A_KillSiblings
TROO L 6 A_NoBlocking
TROO M -1
Stop
Death.Banish:
TROO I 8 A_RemoveMaster
TROO J 8 A_Scream
TROO K 6 A_RemoveSiblings
TROO L 6 A_NoBlocking
TROO M 6
TNT1 A -1
Stop
}
}
```

A_RemoveSiblings

Jump to navigationJump to search

A_RemoveSiblings [(bool all [, int flags [, string filter [, string species]]]])

Usage

Called by monsters to completely remove all monsters spawned by the calling actor's master. If all is false or not specified, only dead siblings are removed. Currently the only function that sets the necessary information is A_SpawnItemEx.

Monsters spawned with A_SpawnProjectile are not affected by this. A_SpawnProjectile was never designed to spawn monsters.

This function can target actors that are non-monsters, but only by using flags.

Parameters

all: if this is false, only dead siblings are removed. Otherwise, if true, all siblings are removed, dead and alive. Default is false.

flags: the following flags can be combined using the | character between name constants:

RMVF_MISSILES - Allows removal of missiles associated with the pointer.

RMVF_NOMONSTERS - The function will not remove monsters and simply skip them. By default, before the introduction of assigning masters to missiles, similar functions would only work on monsters. To ensure consistency, only monsters are allowed for removal by default.

RMVF_MISC - Allows removal of things that are neither missile nor monster.

RMVF_EVERYTHING - Overrides all other flags. Disables discrimination and removes the actor of any type regardless of what it is.

RMVF_EXFILTER "inverts the case of the class name filter; the calling actor's siblings are only removed if their class name does not match the value passed to filter.

RMVF_EXSPECIES "inverts the case of the species filter; the calling actor's siblings are only removed if their species does not match the value passed to species.

RMVF_EITHER "the calling actor's siblings are removed if either of their class name or species matches the values passed to filter and species, respectively.

filter: the actor class to remove. The calling actor's siblings are only removed if their class name matches the specified filter class. Default is "None".

species: the actor species to remove. The calling actor's siblings are only removed if their species matches the specified species filter. Default is "None".

Examples

The following is a variant on the doom imp, a monster spawned by a different monster using the master/child flag. If it dies from damage type "Banish", its death state triggers A_RemoveSiblings, completely removing all the other clones who were spawned by their master.

ACTOR SoldierImp : DoomImp

```
{
  States
  {
    Pain.Voodoo:
      TROO H 2 A_Pain
      TROO H 50
      Goto See
    Pain.Curse:
      TROO H 2 A_Pain
      TROO H 2 A_DamageSiblings(15)
      Goto See
    Death:
```

```
TROO I 8 A_DamageMaster(-25)
TROO J 8 A_Scream
TROO K 6
TROO L 6 A_NoBlocking
TROO M -1
Stop
Death.Curse:
TROO I 8 A_KillMaster
TROO J 8 A_Scream
TROO K 6 A_KillSiblings
TROO L 6 A_NoBlocking
TROO M -1
Stop
Death.Banish:
TROO I 8 A_RemoveMaster
TROO J 8 A_Scream
TROO K 6 A_RemoveSiblings
TROO L 6 A_NoBlocking
TROO M 6
TNT1 A -1
Stop
}
}
```

A_RemoveTarget

Jump to navigationJump to search

A_RemoveTarget [(int flags [, string filter [, string species]])]

Usage

Called by monsters to completely remove the monster that is in their target pointer. Currently the only function that sets the necessary information is A_SpawnItemEx.

Monsters spawned with A_SpawnProjectile are not affected by this. A_SpawnProjectile was never designed to spawn monsters.

This function can target actors that are non-monsters, but only by using flags.

Parameters

flags: the following flags can be combined using the | character between name constants:

RMVF_MISSILES - Allows removal of missiles associated with the pointer.

RMVF_NOMONSTERS - The function will not remove monsters and simply skip them. By default, before the introduction of assigning masters to missiles, similar functions would only work on monsters. To ensure consistency, only monsters are allowed for removal by default.

RMVF_MISC - Allows removal of things that are neither missile nor monster.

RMVF_EVERYTHING - Overrides all other flags. Disables discrimination and removes the actor of any type regardless of what it is.

RMVF_EXFILTER " inverts the case of the class name filter; the calling actor's target is only removed if its class name does not match the value passed to filter.

RMVF_EXSPECIES " inverts the case of the species filter; the calling actor's target is only removed if its species does not match the value passed to species.

RMVF_EITHER " the calling actor's target is removed if either of its class name or species matches the values passed to filter and species, respectively.

filter: the actor class to remove. The calling actor's target is only removed if its class name matches the specified filter class. Default is "None".

species: the actor species to remove. The calling actor's target is only removed if its species matches the specified species filter. Default is "None".

Examples

This is a modified imp, whom if summoned or introduced in an area with enemies, will instantly remove any monsters that it attacks, using A_RemoveTarget.

ACTOR AdminImp : Doomimp

```
{
+FRIENDLY
+MISSILEMORE
+MISSILEEVENMORE
States
{
Melee:
Missile:
TROO EF 8 A_FaceTarget
TROO G 6 A_RemoveTarget (RMVF_EVERYTHING)
Goto See
}
}
```

A_RemoveTracer

A_RemoveTracer [(int flags [, string filter [, string species]])]

Usage

Called by monsters to completely remove the monster that is in their tracer pointer. Currently the only function that sets the necessary information is A_SpawnItemEx.

Monsters spawned with A_SpawnProjectile are not affected by this. A_SpawnProjectile was never designed to spawn monsters.

This function can target actors that are non-monsters, but only by using flags.

Parameters

flags: the following flags can be combined using the | character between name constants:

RMVF_MISSILES - Allows removal of missiles associated with the pointer.

RMVF_NOMONSTERS - The function will not remove monsters and simply skip them. By default, before the introduction of assigning masters to missiles, similar functions would only work on monsters. To ensure consistency, only monsters are allowed for removal by default.

RMVF_MISC - Allows removal of things that are neither missile nor monster.

RMVF_EVERYTHING - Overrides all other flags. Disables discrimination and removes the actor of any type regardless of what it is.

RMVF_EXFILTER " " inverts the case of the class name filter; the calling actor's tracer is only removed if its class name does not match the value passed to filter.

RMVF_EXSPECIES " " inverts the case of the species filter; the calling actor's tracer is only removed if its species does not match the value passed to species.

RMVF_EITHER " " the calling actor's tracer is removed if either of its class name or species matches the values passed to filter and species, respectively.

filter: the actor class to remove. The calling actor's tracer is only removed if its class name matches the specified filter class. Default is "None".

species: the actor species to remove. The calling actor's tracer is only removed if its species matches the specified species filter. Default is "None".

A_SentinelBob

(no parameters)

Makes the object bob. This function has to be called repeatedly in a loop for a permanent effect. The effect of this function is not the same as using the FLOATBOB flag.

Examples

For an example how this works it is best to look at Strife's Sentinel:

ACTOR Sentinel 3006

```
{
  [...]
  States
  {
    [...]
    See:
      SEWR A 6 A_SentinelBob
      SEWR A 6 A_Chase
    Loop
    [...]
  }
}
```

A_Teleport

state, bool A_Teleport [(string teleportstate [, string targettype [, string fogtype [, int flags [, float mindist [, float maxdist [, pointer ptr]]]]]])]

Note: Jump functions perform differently inside of anonymous functions.

Usage

Attempts to teleport the actor to a SpecialSpot-derived actor that is at least mindist away and at most maxdist away. If maxdist is 0, there is no maximum distance limit. If successful, an actor of type fogtype is spawned at the old location and calling actor is placed in the state indicated by teleportstate. If a spot is obscured by a ceiling or a floor, it will adjust the actor's z position to not be blocked if possible.

Note that despite requiring an actor at the target location, this function can be used in a monster without a specially prepared field. It should be possible, for instance, for that monster to drop SpecialSpot-derived actors around (possibly with a timed life) to teleport to later.

Parameters

teleportstate: The state in which the actor should go after teleporting. If the actor does not have this state, or an empty string ("") is given, "Teleport" is assumed. If the actor does not have a Teleport state either, depending on the flags below, the default behavior resorts to no teleportation or state jump occurring.

targettype: The type of the target. This needs to be an actor type derived from SpecialSpot. By default, BossSpot is used.

fogtype: The actor to spawn at the position being left. By default, TeleportFog is used. To disable, use None or the flags below.

flags: The following flags can be combined by using the | character between the constant names:

TF_TELEFRAG "Telefrag: Allow telefrag in order to teleport.

TF_RANDOMDECIDE "Random decision: Randomly choose not to teleport based on current health ratio, like

A_Srcr2Decide:

Health fraction	Teleportation chance
-----------------	----------------------

8/8 or more	0%
-------------	----

7/8	6.25%
-----	-------

6/8	12.5%
-----	-------

4/8	25%
-----	-----

1/8	46.875%
-----	---------

Less than 1/8	75%
---------------	-----

TF_FORCED: Forces the teleportation regardless of anything blocking it.

TF_KEEPVELOCITY: By default, the function stops all movement. This allows the calling actor to keep moving after teleporting.

TF_KEEPPANGLE: By default, the function will make the actor face the same direction as the SpecialSpot it teleports to. This flag lets the actor keep their angle.

TF_KEEPPORIENTATION: Implies TF_KEEPVELOCITY and TF_KEEPPANGLE. This is the same as putting in TF_KEEPVELOCITY|TF_KEEPPANGLE.

TF_USESPOTZ: By default, the teleporting actor's z position is set to the floor. This flag prevents grounding so they can teleport into the air.

TF_NOSRCFOG: The calling actor will not spawn teleport fog at the old location they teleport away from.

TF_NODESTFOG: The calling actor will not spawn teleport fog at their arriving destination.

TF_NOFOG: Implies TF_NOSRCFOG and TF_NODESTFOG flags. This is the same as putting in TF_NOSRCFOG|TF_NODESTFOG.

TF_USEACTORFOG: Uses the fogs defined in TeleFogSourceType and TeleFogDestType property fields.

TF_NOJUMP: The actor will not jump after teleporting, and they no longer require a state to jump. This is especially useful for CustomInventory items to prevent breaking actors during an important state transition. If used, it is recommended to set the teleportstate to 'None'.

TF_OVERRIDE: If set, actors with the NOTELEPORT flag set can teleport, still.

TF_SENSITIVEZ: Actors will fail to teleport instead of having their z positions adjusted to make up for floors or

ceilings blocking the way.

mindist: The actor must be this distance away in order to teleport. The default value is 0.

maxdist: The actor must be within this distance in order to teleport. The default value is 0 (unlimited range).

ptr: Determines who will be teleported. Defaults to AAPTR_DEFAULT (self). Jumping and TF_NOJUMP are limited to the caller only, while everything else will affect the pointer instead.

Examples

The following is a variant of the Doom imp, a monster that will randomly teleport around a room, firing at you. It uses A_Jump to decide a certain chance of it triggering A_Teleport, causing it to teleport to a random "ImpSpot" (a variant of BossSpot), in its attack state.

```
actor TeleportImp : DoomImp 601
{
    States
    {
        Spawn:
            TROO AB 10 A_Look
        Loop
        See:
            TROO AABBCDD 3 A_Chase
            TROO A 0 A_Jump(200, "See")
            TROO A 0 A_Teleport("Missile", "ImpSpot")
        Loop
        Melee:
        Missile:
            TROO EF 8 A_FaceTarget
            TROO G 6 A_TroopAttack
            Goto See
    }
}
```

This is the SpecialSpot used for the monster to teleport to randomly.

```
actor ImpSpot : SpecialSpot 600
{
    +INVISIBLE
}
```

A_TurretLook

(no parameters)

Note: This function has been superseded by A_LookEx, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

Looks for players or other attackable actors in the game. This function is similar to A_Look but it only reacts to sound. Just seeing the player will not wake up a monster using this. This function must be used in the idle states of a monster.

The major difference between A_Look2 and A_TurretLook is that A_TurretLook does not do any random state jumps.

Examples

This is a blind imp, you can walk right in front of it and it will not see you, though shooting will awaken it, since its spawn state uses A_TurretLook, making it react to sound. Since it is blind, it never directly aims at you unless you trigger its melee state.

ACTOR BlindImp : DoomImp

```
{
  States
  {
    Spawn:
      TROO AB 10 A_TurretLook
      Loop
    See:
      TROO AABB 3 A_Chase
      TROO CCDD 3 A_Wander
      Loop
    Melee:
      TROO EF 8 A_FaceTarget
      TROO G 6 A_CustomComboAttack("DoomImpBall", 32, 3 * random(1, 8), "imp/melee")
      Goto See
    Missile:
      TROO EF 8
      TROO G 6 A_CustomMissile("DoomImpBall", 20, 0, 0, 2)
      Goto See
  }
}
```

A_VileChase

(no parameters)

Note: This function has been superseded by A_Chase, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

Usage

This function makes the calling actor advance on its target. The actor may enter an attack state (if it has one) at any time. If the calling actor comes into contact with a corpse, it will enter the "Heal" state (if one exists) and resurrect the actor in question (provided the actor being revived has a "Raise" state defined).

The function is named after Doom II's Archvile, where the "healing" ability originated from.

Examples

The archvile uses A_VileChase since it is able to revive monsters, entering its heal state whenever it comes across the corpse of a monster that can be revived.

ACTOR Archvile 64

```
{
  States
  {
    See:
      VILE AABBCCDDEEFF 2 A_VileChase
      Loop
    Heal:
      VILE "[\]" 10 Bright
      Goto See
  }
}
```

A_Wander

void A_Wander [(int flags)]

Usage

Makes the actor wander around aimlessly, as used by Strife's peasants. An actor will not play active sounds, attack players, or attack any other target when calling this function, unlike A_Chase. A_Wander does nothing when the actor calling it has the STANDSTILL flag set. For friendly monsters, the default behavior of this function is to make them follow the player. To make friendly monsters calling this function actually wander around instead, you should use the DONTFOLLOWPLAYERS flag.

Parameters

flags: The following flags can be combined using the | symbol:

CHF_NORANDOMTURN - Actor will not attempt to turn during their chasing frames at random. They will only turn when they encounter an obstacle.

CHF_NODIRECTIONTURN - Actor will not turn its angle to face the direction of travel.

CHF_STOPIFBLOCKED - Actor cannot turn away from obstacles blocking it. It will simply not move, but can still angle itself.

CHF_DONTTURN - Implies CHF_NORANDOMTURN and CHF_STOPIFBLOCKED.

Examples

Here is an example of an Imp that wanders around randomly looking for players.

```
actor ImpScout : DoomImp
{
States {
  Spawn:
    TROO AA 3 A_Wander
    TROO A 0 A_Look
    TROO BB 3 A_Wander
    TROO B 0 A_Look
    TROO CC 3 A_Wander
    TROO C 0 A_Look
    TROO DD 3 A_Wander
    TROO D 0 A_Look
  loop
}
```

A_Look has to repeatedly be called to check for players while it is wandering around. It is also interesting that A_Wander can be in a different way used. This Cacodemon for example is able to teleport, but it's not a real teleport. It just gets invisible and walks around for 0 tics and gets visible again:

```
actor TeleCaco : Cacodemon
{
States {
  See:
    HEAD A 0 A_Jump(16,"Tele")
    HEAD A 4 A_Chase
  loop
  Tele:
    HEAD A 1 A_SetRenderStyle(0.8, STYLE_Translucent)
    HEAD A 1 A_SetRenderStyle(0.6, STYLE_Translucent)
    HEAD A 1 A_SetRenderStyle(0.4, STYLE_Translucent)
    HEAD A 1 A_SetRenderStyle(0.2, STYLE_Translucent)
```

```
TNT1 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 0 A_Wander
HEAD A 1 A_SetRenderStyle(0.2, STYLE_Translucent)
HEAD A 1 A_SetRenderStyle(0.4, STYLE_Translucent)
HEAD A 1 A_SetRenderStyle(0.6, STYLE_Translucent)
HEAD A 1 A_SetRenderStyle(0.8, STYLE_Translucent)
HEAD A 1 A_SetRenderStyle(1.0, STYLE_Translucent)
Goto See
}
}
```

A_BetaSkullAttack

(no parameters)

The attack of press-release beta Doom's version of the lost soul. It is an invisible, instant attack that directly inflicts damage to the target, unless the target belongs to the same species as the calling actor. Since the damage is inflicted in this manner, rather than through a projectile or a hitscan, the function can never trigger infighting.

When damage is to be inflicted, the calling actor plays its attack sound, which is played on the weapon channel (CHAN_WEAPON), faces its target, and does the damage. The damage inflicted is 1d8 times the calling actor's Damage property, and is of damage type None.

A_BrainExplode

(no parameters)

Spawns one element of the brain explosion (see A_BrainScream).

Examples

This example is taken from Doom's Icon of Sin.

Death:

BBRN A 100 A_BrainScream

BBRN AA 10

BBRN A -1 A_BrainDie

Stop

A_BrainScream

(no parameters)

Starts the brain explosion and plays the sound "œbrain/death" at full volume.

Examples

This example is taken from Doom's Icon of Sin.

Death:

BBRN A 100 A_BrainScream

BBRN AA 10

BBRN A -1 A_BrainDie

Stop

A_BrainSpit

A_BrainSpit [(string Actor)]

Spits one boss cube for the Doom2 end boss sequence.

You can also optionally specify a parameter, which is the actor to spawn instead of the default boss cube.

The sound made when launching the projectile is the attack sound of the actor using A_BrainSpit (or "brain/spit" if no attack sound is defined).

Examples

This example is taken from Doom's Icon of Sin

See:

SSWV A 181 A_BrainAwake

SSWV A 150 A_BrainSpit // See SpawnShot

Wait

A_BruisAttack

(no parameters)

The attack of Doom's Hell Knight and Baron of Hell. This either shoots a BaronBall or if the target is sufficiently close plays the sound "baron/melee" and performs a melee attack with a base damage of 10 multiplied by a random value between 1 and 8.

The behavior of this function can be replicated by the use of A_CustomComboAttack in the following manner:

```
A_CustomComboAttack("BaronBall", 32, 10 * random(1, 8), "baron/melee")
```

Examples

This example is taken straight from Doom's Hellknight.

Melee:

Missile:

```
BOS2 EF 8 A_FaceTarget
```

```
BOS2 G 8 A_BruisAttack // See BaronBall
```

```
goto See
```

A_BspiAttack

(no parameters)

The attack of Doom's Arachnotron. This shoots an ArachnotronPlasma missile.

Examples

This example is taken straight from Doom's Arachnotron.

Missile:

```
BSPI A 20 bright A_FaceTarget
BSPI G 4 bright A_BspiAttack // See ArachnotronPlasma
BSPI H 4 bright
BSPI H 1 bright A_SpidRefire
goto Missile+1
```

A_CPosAttack

(no parameters)

The attack of Doom's chaingunner and Nazi. This shoots one bullet and plays the calling actor's attack sound.

It is almost equivalent to:

CPOS F 4 bright

{

 A_PlaySound("chainguy/attack", CHAN_WEAPON);

 A_CustomBulletAttack(22.5, 0, 1, random(1,5)*3, "BulletPuff", 0, CBAF_NORANDOM);

}

Examples

This example is taken straight from Doom's Chaingun Guy.

Missile:

CPOS E 10 A_FaceTarget

CPOS FE 4 bright A_CPosAttack

CPOS F 1 A_CPosRefire

goto Missile+1

A_CyberAttack

(no parameters)

The attack of Doom's cyberdemon. This shoots a rocket.

Examples

This is the Cyberdemon's Missile state in DECORATE:

Missile:

```
CYBR E 6 A_FaceTarget
CYBR F 12 A_CyberAttack // See Rocket
CYBR E 12 A_FaceTarget
CYBR F 12 A_CyberAttack
CYBR E 12 A_FaceTarget
CYBR F 12 A_CyberAttack
goto See
```

A_FatAttack1

A_FatAttack1 [(string spawntype)]

A_FatAttack2 [(string spawntype)]

A_FatAttack3 [(string spawntype)]

The 3 attack stages of Doom's Mancubus. Each one shoots 2 missiles of the specified type in different directions. If no type is specified it will use FatShot (the regular Mancubus missile.)

A_FatAttack1 will shoot one shot straight ahead, and the second 11.25 degrees to its left.

A_FatAttack2 will shoot one shot straight ahead, and the second 11.25 degrees to its right.

A_FatAttack3 will shoot two shots, one 5.625 degrees to the left, and the other 5.625 degrees to the right.

Examples

This is the Mancubus' Missile state:

Missile:

```
FATT G 20 A_FatRaise
FATT H 10 bright A_FatAttack1 // See FatShot
FATT IG 5 A_FaceTarget
FATT H 10 bright A_FatAttack2
FATT IG 5 A_FaceTarget
FATT H 10 bright A_FatAttack3
FATT IG 5 A_FaceTarget
goto See
```


A_HeadAttack

(no parameters)

The attack of Doom's cacodemon. This either shoots a CacodemonBall or if the target is sufficiently close performs a melee attack with a base damage of 10 multiplied by a random value between 1 and 6.

The behavior of this function can be replicated by the use of A_CustomComboAttack in the following manner:

```
A_CustomComboAttack("CacodemonBall", 32, 10 * random(1, 6))
```

Note that the cacodemon doesn't come with a melee sound. Hence, the sound omission from the line above, though it can be added very easily if provided.

Examples

This example is taken straight from Doom's Cacodemon.

Missile:

```
HEAD BC 5 A_FaceTarget
```

```
HEAD D 5 bright A_HeadAttack // See CacodemonBall
```

```
goto See
```

A_M_Saw

A_M_Saw [(string fullsound [, string hitsound [, int damage [, string pufftype]]]])]

A chainsaw attack available to monsters. For weapons, use A_Saw.

Parameters

fullsound: The sound that plays if the attack doesn't hit anything. Defaults to "weapons/sawfull".

hitsound: The sound that plays if the weapon hits a target. Defaults to "weapons/sawhit".

damage: The amount of damage to deal, with the following calculation:

if (damage == 0) damage = 2;

damage *= (random() % 10 + 1);

For example, if damage is 5, the damage dealt will be between 5 and 50. Note that it is not possible to have this function deal 0 damage, since 0 means "use the default value" which is 2.

pufftype: The puff to spawn if the attack hits a wall or invulnerable actor. Defaults to "BulletPuff".

Examples

This example uses a Marine that uses his chainsaw up close.

Melee:

PLAY E 4

PLAY E 4 A_M_Saw

PLAY E 0 A_JumpIfCloser(40, "Melee")

goto See

A_MonsterRefire

state A_MonsterRefire (int chancecontinue, str "abortstate")

A_CPosRefire

(no parameters)

A_CrusaderRefire

(no parameters)

A_SentinelRefire

(no parameters)

A_SpidRefire

(no parameters)

Calls A_FaceTarget and then checks whether the monster should abort its attack sequence and go back to abortstate. If the target is out of sight or dead, has a chancecontinue/256 chance to not jump to the abort state.

The monster-specific functions use the following parameters:

A_CPosRefire: 40, "See"

A_CrusaderRefire: 0, "See"

A_SpidRefire: 10, "See"

A_SentinelRefire: 30, "See"

All these functions jump to the "See" state if the attack is to be aborted. The loop has to be explicitly coded in the actor definition. A_SentinelRefire also has a 10/256 chance of aborting the attack even if the target is still in sight.

A_CrusaderRefire is restricted to Crusader and derived classes.

Example

ACTOR SuperZombie : ZombieMan replaces ZombieMan

```
{
    States
    {
        Missile:
            POSS E 10 A_FaceTarget
            POSS FE 2 Bright A_PosAttack
            POSS F 1 A_MonsterRefire(130, "See") // About 50% chance to jump to "See" if target is out of sight.
            Goto Missile+1 // Looping back to the attack state to allow the actual refire!
    }
}
```

A_Mushroom

A_Mushroom [(string spawntype [, int amount[, int flags[, float vrange[, float hrangle]]]])]

Shoots the specified amount of missiles high into the air with a mushroom cloud effect, performing a radius attack of the calling actor's DamageType as damage type and both damage and radius of 128.

The parameters are:

spawntype: type of projectile to throw. Default Mancubus fireballs.

amount: the number of thrown projectiles. If 0, then the number is determined by the calling actor's damage property. Default 0.

flags: the following flags can be combined by using the | character between the constant names:

MSF_Standard “ uses the normal ZDoom projectile spawning function, in which the vertical velocity is one component of the projectile's speed. This flag is implied if the flags parameter is undefined or zero.

MSF_Classic “ uses the old Doom function in which the vertical velocity is added to the projectile's speed. Since A_Mushroom by default aims its projectiles at very steep angles, the difference between the standard and classic is very noticeable; the older mode results in a more chaotic-looking, asymmetrical explosion, that spreads much higher and farther.

MSF_DontHurt “ the target of the calling actor is considered to be the shooter of the debris projectiles and won't be hurt by the explosion. Use this flag if A_Mushroom is called from a projectile and should not hurt its shooter.

vrange: controls how high the projectiles are aimed, the larger the value, the higher they go. Default 4.0.

hrangle: factor by which the speed of the projectiles is multiplied. Default 0.5.

Example

ACTOR MushRoomBall

```
{
  Radius 3
  Height 2
  Speed 15
  FastSpeed 22
  Damage 6
  Projectile
  +RANDOMIZE
  RenderStyle Add
  Alpha 0.75
  SeeSound "imp/attack"
  DeathSound "imp/shotx"
  States
  {
    Spawn:
      BAL1 AB 2 Bright
      Loop
    Death:
      BAL1 CDE 2 Bright A_Mushroom("PlasmaBall", 10, MSF_DontHurt, 8, 0.1)
      Stop
  }
}
```

A_PainAttack

```
void A_PainAttack [(class<Actor> spawntype [, double addangle [, int flags [, int limit )]]]]  
void A_DualPainAttack [(class<Actor> spawntype)]
```

Usage

The attack of Doom's Pain Elemental. If no parameter is specified this shoots a Lost Soul. A_DualPainAttack is the variant from Doom64. It shoots two actors at an angle of +45 and -45 degrees.

If the calling actor is massacred, the functions do nothing.

Parameters

spawntype: The type of actor to spawn. Default is "LostSoul".

addangle: The angle at which the spawned actor is projected. Default is 0.

flags: The following flags can be combined by using the | character between the constant names:

PAF_NOSKULLATTACK â€” No skull attack: The spawned actor will not immediately call A_SkullAttack as it normally would.

PAF_AIMFACING â€” Aim with current facing: The calling actor will not call A_FaceTarget before spawning the monster.

PAF_NOTARGET â€” No target: The spawned actor will not adopt the calling actor's target as its own.

limit: Spawning the given actor will fail if there are already that many on the map. 0 is unlimited. If compat_limitpain is on and this is less than 0, the limit is set to 21, otherwise it is unlimited. Default is -1.

Examples

This example is taken straight from Doom's Pain Elemental.

Missile:

```
PAIN DE 5 A_FaceTarget  
PAIN F 5 bright A_FaceTarget  
PAIN F 0 bright A_PainAttack // See LostSoul  
goto See
```

A_PainDie

A_PainDie [(string spawntype)]

The death effect of Doom's pain elemental. Calls A_Fall and then three actors of the specified type are shot in different directions. If no parameter is specified it will shoot lost souls.

Examples

This example is taken from Doom's Pain Elemental.

Death:

PAIN H 8 bright

PAIN I 8 bright A_Scream

PAIN JK 8 bright

PAIN L 8 bright A_PainDie // See LostSoul

PAIN M 8 bright

stop

A_PosAttack

(no parameters)

The attack of Doom's zombieman. This shoots one bullet and plays the sound "grunt/attack" .

It is the equivalent to calling A_CustomBulletAttack with the parameters (22.5, 0, 1, random(1,5)*3, "BulletPuff", 0, CBAF_NORANDOM).

Examples

This example is taken straight from Doom's Zombieman.

Missile:

POSS E 10 A_FaceTarget

POSS F 8 A_PosAttack

POSS E 8

goto See

This example uses a generic function to exactly replicate it instead.

Missile:

POSS E 10 A_FaceTarget

POSS E 0 A_PlaySound ("grunt/attack")

POSS F 8 A_CustomBulletAttack (22.5, 0, 1, random(1,5) * 3, "BulletPuff", 0, CBAF_NORANDOM)

POSS E 8

goto See

A_SargAttack

(no parameters)

The attack of Doom's Demon. This performs a melee attack with a base damage of 4 multiplied by a random value between 1 and 10.

Examples

This example is taken straight from Doom's Demon.

Melee:

SARG EF 8 A_FaceTarget

SARG G 8 A_SargAttack

goto See

A_SkelFist

(no parameters)

The melee attack of Doom's Revenant. This plays the sound "skeleton/melee" and performs a melee attack with a base damage of 6 multiplied by a random value between 1 and 10.

Examples

This example is taken from Doom's Revenant.

Melee:

```
SKEL G 0 A_FaceTarget  
SKEL G 6 A_SkelWhoosh  
SKEL H 6 A_FaceTarget  
SKEL I 6 A_SkelFist  
goto See
```

A_SkelMissile

(no parameters)

The missile attack of Doom's Revenant. This shoots a RevenantTracer missile. The spawn height for the missile is 8 units higher than the default.

Examples

This example is taken from Doom's Revenant.

Missile:

```
SKEL J 0 bright A_FaceTarget
SKEL J 10 bright A_FaceTarget
SKEL K 10 A_SkelMissile      // See RevenantTracer
SKEL K 10 A_FaceTarget
goto See
```

A_SkullAttack

A_SkullAttack [(int speed)]

The attack of Doom's lost soul. The calling actor charges at its current target. The speed parameter defines the speed of the charge (20 by default).

Examples

This example is taken straight from Doom's Lost Soul.

Missile:

SKUL C 10 bright A_FaceTarget

SKUL D 4 bright A_SkullAttack

SKUL CD 4 bright

goto Missile+2

A_SpawnFly

A_SpawnFly [(string FogActor)]

DoomWiki.org

For more information on this article, visit the A_SpawnFly page on the Doom Wiki.

Checks whether a boss cube has reached its target and if so spawns a monster there. It also telefrags anything in the way of spawning.

If FogActor is provided, the specified actor will spawn in place of the default SpawnFire. Also, the sound that plays at spawn will be the see sound of the FogActor indicated.

Examples

This example is taken from Doom's spawn cube.

Spawn:

BOSF A 3 BRIGHT A_SpawnSound

BOSF BCD 3 BRIGHT A_SpawnFly // See SpawnFire

Loop

A_SpawnSound

(no parameters)

Note: This function has been superseded by A_PlaySound, which duplicate and extend its functionality. Use of the newer functions is advised in order to maintain maximum flexibility in your code.

Plays the sound "brain/cube" on the "Body" channel and calls A_SpawnFly without parameters. This function is kept for DEHACKED compatibility and should not be used for new projects.

A_SPosAttack

(no parameters)

The attack of Doom's Shotgun Guy. This shoots three bullets and plays the sound "shotguy/attack".

It is equivalent of:

```
SPOS F 0 bright A_PlaySound("shotguy/attack", CHAN_WEAPON)
```

```
SPOS F 10 bright A_CustomBulletAttack(22.5, 0, 3, random(1,5)*3, "BulletPuff", 0, CBAF_NORANDOM)
```

Examples

This example is taken straight from Doom's Shotgun Guy.

Missile:

```
SPOS E 10 A_FaceTarget
```

```
SPOS F 10 bright A_SPosAttack
```

```
SPOS E 10
```

```
goto See
```

A_TroopAttack

(no parameters)

The attack of Doom's imp. This either shoots a DoomImpBall or, if the target is sufficiently close, it plays the sound "imp/melee" and performs a melee attack with a base damage of 3 multiplied by a random value between 1 and 8.

The behavior of this function can be replicated by the use of A_CustomComboAttack in the following manner:

```
A_CustomComboAttack("DoomImpBall", 32, 3 * random(1, 8), "imp/melee")
```

Examples

This example is taken straight from Doom's Imp.

Missile:

```
TROO EF 8 A_FaceTarget
```

```
TROO G 6 A_TroopAttack // See DoomImpBall
```

```
goto See
```

A_VileAttack

A_VileAttack [(str sound [, int initialdamage [, int blastdamage [, int blastradius [, float thrustfactor [, str damagetype [, int flags]]]]]])]

Usage

Performs the actual attack of the Arch-Vile. To hit, the current actor must have a target and that target must be in line of sight. If the hit is successful, the specified sound is played, the initial damage is dealt to the target and they are thrust vertically into the air. If this actor has a tracer assigned (see A_VileTarget) then that actor also explodes and deals the blast damage to the target and surrounding actors.

Parameters

sound: The sound that is played when the attack hits. Default is "vile/stop".

initialdamage: The amount of direct-damage to do to the target immediately on hit. Default is 20.

blastdamage: The damage done by the explosion spawned at the tracer. Default is 70.

blastradius: The radius of the explosion spawned at the tracer. Other actors within this range will take damage from the explosion. Default is 70.

thrustfactor: The amount of vertical thrust to apply to the target. Default is 1.0. Higher values will push the target further into the air, while lower values will lessen the effect.

damagetype: The type of damage to deal to actors hit by this effect. Applies only to the blast attack, unless the below flag is set. Default is "Fire".

flags:

VAF_DMGTTYPEAPPLYTODIRECT – If set, the specified damage type applies to both attacks. Otherwise, it only applies to the blast attack, and in which case, the initial/direct attack's damage type will be None.

Examples

This example is taken from Doom's Arch-Vile.

Missile:

```
VILE G 0 BRIGHT A_VileStart
VILE G 10 BRIGHT A_FaceTarget
VILE H 8 BRIGHT A_VileTarget
VILE IJKLMN 8 BRIGHT A_FaceTarget
VILE O 8 BRIGHT A_VileAttack
VILE P 20 BRIGHT
Goto See
```


A_VileTarget

A_VileTarget [(string type)]

Spawns the Arch Vile's fire in front of the target, and sets the spawned actor as the current actor's tracer. (See A_VileAttack)

If type is specified, this will spawn an actor of that type instead of ArchvileFire.

Examples

This example is taken from Doom's Archvile.

Missile:

```
VILE G 0 BRIGHT A_VileStart
VILE G 10 BRIGHT A_FaceTarget
VILE H 8 BRIGHT A_VileTarget
VILE IJKLMN 8 BRIGHT A_FaceTarget
VILE O 8 BRIGHT A_VileAttack
VILE P 20 BRIGHT
Goto See
```

A_Log

void A_Log (string whattoprint [, bool local])

Usage

Logs a string to the console and displays it on the screen.

The printed text can be formatted by using the escape characters, in a similar manner to how it is done with Print and Log ACS functions.

Parameters

whattoprint: The string to log.

local: If true, the string is logged if the player is either looking out the calling actor's eyes, or the calling actor is the player and the player is looking out the eyes of a non-monster actor. Default is false.

Examples

```
TNT1 A 0 A_Log("Hello World")
```

This will log the above string, with Hello colored in red and World in blue.

```
TNT1 A 0 A_Log("\cGHello \cHWorld")
```

A_LogFloat

void A_LogFloat (float whattoprint [, bool local)

Usage

Logs a floating-point value to the console and displays it on the screen.

Parameters

whattoprint: The floating-point value to log.

local: If true, the value is logged if the player is either looking out the calling actor's eyes, or the calling actor is the player and the player is looking out the eyes of a non-monster actor. Default is false.

Examples

Basic example: this logs the value 42.5.

```
SOUL A 6 A_LogFloat(42.5)
```

A_LogInt

void A_LogInt (int whattoprint [, bool local])

Usage

Logs an integer value to the console and displays it on the screen.

Parameters

whattoprint: The integer value to log.

local: If true, the value is logged if the player is either looking out the calling actor's eyes, or the calling actor is the player and the player is looking out the eyes of a non-monster actor. Default is false.

Examples

Basic example: this logs the number 5.

SOUL A 6 A_LogInt(5)

Whenever this revenant suffers pain, its current health is logged.

ACTOR InformativeRevenant : Revenant

```
{
  States
  {
    Pain:
      SKEL L 5
      SKEL L 5 A_Pain
      SKEL L 0 A_LogInt(health) // See DECORATE expressions.
      Goto See
  }
}
```

A_Print

A_Print (string text[, float time[, string fontname]])

Usage

Prints a text to the caller's screen. This means the caller has to either be a player or be used as a camera by a player.

It can also be used from inventory items or weapons. In these cases the text will be printed to the screen of the player using this item.

The text can be formatted text.

Optionally, a duration for how long the message must be displayed can be passed. The default value of 0 means to use con_midtime.

Also optionally, a different font can be selected. By default, or if the font name given is invalid, it will use SmallFont.

Examples

This inventory item prints a message to the player when it is picked up.

```
actor goldenticket : CustomInventory
{
    Inventory.Amount 1
    Inventory.MaxAmount 1
    +INVBAR
    States
    {
    Spawn:
        GTIK ABCD 4
        Loop
    Pickup:
        GTIK A 0 A_Print("You got the golden ticket! Escape to the chocolate factory!")
        Stop
    }
}
```

A_PrintBold

A_PrintBold (string text[, float time[, string fontname]])

Prints a text to all players' screens.

The text can be formatted text.

Optionally, a duration for how long the message must be displayed can be passed. The default value of 0 means to use con_midtime.

Also optionally, a different font can be selected. By default, or if the font name given is invalid, it will use SmallFont.

Examples

Revenant that shoots 4 times:

Actor RevenantBoss4ZDWiki: Revenant replaces Revenant

```
{
Health 700
Translation "16:47=64:79"
States
{
Missile:
    SKEL J 0 Bright A_FaceTarget
    SKEL J 10 Bright A_FaceTarget
    SKEL KKKK 2 A_SkelMissile
    SKEL K 10 A_FaceTarget
    Goto See
Death:
    SKEL L 7 A_PrintBold( "Next revenant killed..." )
    SKEL M 7
    SKEL N 7 A_ScreamAndUnblock
    SKEL O 7
    SKEL P 7
    SKEL Q -1
    Stop
}
}
```

ACS_NamedExecute

```
bool ACS_NamedExecute (string script, int map, int s_arg1, int s_arg2, int s_arg3)
ACS_NamedExecute (string script, int map, int s_arg1, int s_arg2, int s_arg3)
```

Usage

Variant of ACS_Execute for named scripts.

There is both an ACS and a DECORATE version of this function. Both behave identically.

However, it is not available as an action special: to call named scripts from a a line or thing special, you have to use the non-named variant (ACS_Execute) in UDMF, with the arg0str custom argument set to the name of the script – this will override the first parameter.

Parameters

script: Name of the script to execute

map: Map which contains the script

s_arg1: First argument passed to the script

s_arg2: Second argument passed to the script

s_arg3: Third argument passed to the script

Return value

Returns true if the script could be executed successfully, false otherwise. Deferred scripts are always considered successful.

Examples

Execute a script named GollyFluff.

```
script "PinFeathers" (void)
{
    ACS_NamedExecute("GollyFluff", 0);
}
```

```
script "GollyFluff" (void)
{
    print(s:"Golly Fluff!");
}
```

ACS_NamedExecuteAlways

```
bool ACS_NamedExecuteAlways (string script, int map, int s_arg1, int s_arg2, int s_arg3)
ACS_NamedExecuteAlways (string script, int map, int s_arg1, int s_arg2, int s_arg3)
```

Usage

Variant of ACS_ExecuteAlways for named scripts.

There is both an ACS and a DECORATE version of this function. Both behave identically.

However, it is not available as an action special: to call named scripts from a a line or thing special, you have to use the non-named variant (ACS_ExecuteAlways) in UDMF, with the arg0str custom argument set to the name of the script – this will override the first parameter.

Parameters

script: Name of the script to execute

map: Map which contains the script

s_arg1: First argument passed to the script

s_arg2: Second argument passed to the script

s_arg3: Third argument passed to the script

Return value

Returns true if the script could be executed successfully, false otherwise. Deferred scripts are always considered successful.

Examples

This example gives a one-time full heal to a player when nearly dead, if the player picked a special item. The item runs the script, while the script monitors players status and activates when players health is reduced to 1.

Decorate Item:

```
Actor AvoidDeath : CustomInventory
{
    Inventory.MaxAmount 0
    +INVENTORY.AUTOACTIVATE
    States
    {
        Use:
            TNT1 A 0 ACS_NamedExecuteAlways("AvoidDeathScript", 0)
            Stop
    }
}
```

ACS Script:

```
script "AvoidDeathScript" (void)
{
    SetPlayerProperty(0, 1, PROP_BUDDHA); // Sets buddha mode for player
    while(1) // Permanent Loop
    {
        if(GetActorProperty(0, APROP_HEALTH) <= 1) // Checks if the player is wounded enough to heal
        {
            SetPlayerProperty(0, 0, PROP_BUDDHA); // Remove the buddha mode
            GiveInventory("Health", 100); // and heal the player
            terminate;
        }
    }
}
```



```
}  
  delay(1);  
}  
}
```

ACS_NamedExecuteWithResult

int ACS_NamedExecuteWithResult (string script, int s_arg1, int s_arg2, int s_arg3, int s_arg4)

Usage

Variant of ACS_ExecuteWithResult for named scripts.

There is both an ACS and a DECORATE version of this function. Both behave identically. The DECORATE version can also be called by the shorter alias CallACS in DECORATE expressions. Another alias for this function is ACS_ScriptCall, which introduces the ability for the function to be called by arbitrary actor objects as well as self object (see the second example below). This alias is exclusive to DECORATE and ZScript.

ACS_(Named)ExecuteWithResult has one small difference aside returning a value. They execute immediately once called, while the other types will usually wait a tic before acting.

However, it is not available as an action special: to call named scripts from a a line or thing special, you have to use the non-named variant (ACS_ExecuteWithResult) in UDMF, with the arg0str custom argument set to the name of the script – this will override the first parameter.

Parameters

script: Name of the script to execute

s_arg1: First argument passed to the script

s_arg2: Second argument passed to the script

s_arg3: Third argument passed to the script

s_arg4: Fourth argument passed to the script

Return value

Returns the return value of the executed script. See SetResultValue.

Examples

This example shows an item similar to Hexen's Mystic Ambit item: Checks which player class uses an item and having a different effect.

Decorate Item:

Actor ClassBoost : CustomInventory

```
{
    Inventory.MaxAmount 25
    Inventory.InterHubAmount 25
    +INVENTORY.INVBAR
    Inventory.Icon "ARTIHRAD"
    States
    {
    Spawn:
        HRAD ABCDEFGHIJKLMNOP 4 Bright
    Loop
    Use:
        TNT1 A 0 A_JumpIf(CallACS("CheckPlayerClass", 0, 0, 0) == 0, "NormalPlayer")
        TNT1 A 0 A_JumpIf(CallACS("CheckPlayerClass", 0, 0, 0) == 1, "AlternatePlayer")
    Fail
    NormalPlayer:
        TNT1 A 0 A_RadiusGive("Health", 256, RGF_PLAYERS | RGF_GIVESELF, random(50, 90))
    Stop
    AlternatePlayer:
```

```

    TNT1 A 0 A_RadiusGive("PowerImproveDamage", 256, RGF_PLAYERS | RGF_GIVESELF, 1)
    Stop
}
}
}
ACS Script:

```

```

script "CheckPlayerClass" (void)
{
    if(CheckActorClass(0, "DoomPlayer"))
    {
        SetResultValue(0);
        terminate;
    }
    else if(CheckActorClass(0, "AlternateDoomPlayer"))
    {
        SetResultValue(1);
        terminate;
    }
}
}

```

This example item demonstrates the use of ACS_ScriptCall. Upon pickup, it spawns a baron of hell and makes two ACS script calls: one which is done by the baron of hell itself, and another which is done by the actor which picked up the item.

```

class BaronSummoner : CustomInventory
{
    States
    {
        Spawn:
            SOUL ABCD 6 Bright;
            Loop;

        Pickup:
            TNT1 A 0
            {
                bool res;
                Actor mobj;
                [res, mobj] = A_SpawnItemEx("BaronOfHell", 100);

                if (res && mobj)
                {
                    // Caller is the baron of hell object.
                    // Using ACS_NamedExecuteWithResult, instead, is unacceptable here.
                    mobj.ACS_ScriptCall("SomeScript");

                    // Caller is the self object (the actor which picked the item up).
                    // Using ACS_NamedExecuteWithResult, instead, is okay here.
                    ACS_ScriptCall("SomeOtherScript");
                }
            }
        Stop;
    }
}

```

ACS_NamedLockedExecute

```
bool ACS_NamedLockedExecute (string script, int map, int s_arg1, int s_arg2, int lock)  
ACS_NamedLockedExecute (string script, int map, int s_arg1, int s_arg2, int lock)
```

Usage

Variant of ACS_LockedExecute for named scripts.

There is both an ACS and a DECORATE version of this function. Both behave identically.

However, it is not available as an action special: to call named scripts from a a line or thing special, you have to use the non-named variant (ACS_LockedExecute) in UDMF, with the arg0str custom argument set to the name of the script – this will override the first parameter.

Parameters

script: Name of the script to execute

map: Map which contains the script

s_arg1: First argument passed to the script

s_arg2: Second argument passed to the script

lock: Required key, if any (see key types)

Return value

Returns true if the script could be executed successfully, false otherwise. Deferred scripts are always considered successful.

ACS_NamedLockedExecuteDoor

```
bool ACS_NamedLockedExecuteDoor (string script, int map, int s_arg1, int s_arg2, int lock)
ACS_NamedLockedExecuteDoor (string script, int map, int s_arg1, int s_arg2, int lock)
```

Usage

Variant of ACS_LockedExecuteDoor for named scripts.

There is both an ACS and a DECORATE version of this function. Both behave identically.

However, it is not available as an action special: to call named scripts from a a line or thing special, you have to use the non-named variant (ACS_LockedExecuteDoor) in UDMF, with the arg0str custom argument set to the name of the script – this will override the first parameter.

Parameters

script: Name of the script to execute

map: Map which contains the script

s_arg1: First argument passed to the script

s_arg2: Second argument passed to the script

lock: Required key, if any (see key types)

Return value

Returns true if the script could be executed successfully, false otherwise. Deferred scripts are always considered successful.

ACS_NamedSuspend

bool ACS_NamedSuspend (string script, int map)

ACS_NamedSuspend (string script, int map)

Usage

Variant of ACS_Suspend for named scripts.

There is both an ACS and a DECORATE version of this function. Both behave identically.

However, it is not available as an action special: to call named scripts from a a line or thing special, you have to use the non-named variant (ACS_Suspend) in UDMF, with the arg0str custom argument set to the name of the script – this will override the first parameter.

Parameters

script: Name of the script to suspend

map: Map which contains the script

Return value

Returns true in all cases.

ACS_NamedTerminate

bool ACS_NamedTerminate (string script, int map)

ACS_NamedTerminate (string script, int map)

Usage

Variant of ACS_Terminate for named scripts.

There is both an ACS and a DECORATE version of this function. Both behave identically.

However, it is not available as an action special: to call named scripts from a a line or thing special, you have to use the non-named variant (ACS_Terminate) in UDMF, with the arg0str custom argument set to the name of the script – this will override the first parameter.

Parameters

script: Name of the script to terminate

map: Map which contains the script

Return value

Returns true in all cases.

A_ActiveSound

(no parameters)

Usage

Plays the actor's active sound on the voice channel.

Examples

Strife's Merchant actors use this special to periodically play their greeting sounds:

Spawn:

MRST A 10 A_Look2

Loop

MRLK A 30 A_ActiveSound

Loop

MRLK B 30

Loop

MRBD ABCDEDCB 4

MRBD A 5

MRBD F 6

Loop

A_BFGSound

(no parameters)

Note: This function has been superseded by A_StartSound, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

Usage

Plays the sound "weapons/bfgf".

Examples

This is an excerpt from the BFG9000 code, unrelated sections snipped out. In the firing sequence. A_BFGSound is called to play the famous BFG charging sound.

```
actor BFG9000 : DoomWeapon 2006
```

```
{
```

```
...
```

```
States
```

```
{
```

```
...
```

```
Fire:
```

```
    BFGG A 20 A_BFGSound
```

```
    BFGG B 10 A_GunFlash
```

```
    BFGG B 10 A_FireBFG
```

```
    BFGG B 20 A_ReFire
```

```
    goto Ready
```

```
...
```

```
}
```

```
}
```

A_BrainAwake

(no parameters)

Note: This function has been superseded by A_StartSound, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

Usage

Plays the sound "brain/sight" at full volume.

Examples

This example is taken from the boss cube shooter.

See:

```
SSWV A 181 A_BrainAwake
```

```
SSWV A 150 A_BrainSpit // See SpawnShot
```

```
Wait
```

A_BrainPain

(no parameters)

Note: This function has been superseded by A_PlaySound, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

Plays the sound "æbrain/pain" at full volume.

Examples

This example is taken from Doom's Icon of Sin.

Pain:

BBRN B 36 A_BrainPain

Goto Spawn

A_FLoopActiveSound

(no parameters)

Usage

Plays the actor's ActiveSound, if one is defined. The sound is only played if the current tic is evenly divisible by eight. Contrary to what the name of this function implies, the sound is not looped. The only standard class that uses this function is WaterDropOnFloor, which uses it to make the sound slightly less repetitive.

For general sound playback, you probably want to use A_PlaySound instead.

Examples

This is from the WaterDropOnFloor actor that uses this sound.

Spawn:

```
DRIP A 6 A_FLoopActiveSound
DRIP BC 4
DRIP D 4 A_FLoopActiveSound
DRIP EF 4
DRIP G 4 A_FLoopActiveSound
DRIP H 4
Loop
```

A_LoopActiveSound

(no parameters)

Note: This function has been superseded by A_StartSound, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

Usage

Plays the actor's ActiveSound, if one is defined, and causes it to loop seamlessly. The sound can be stopped with A_StopSound.

This function doesn't work on weapons.

A_Pain

(no parameters)

Usage

Plays the actor's pain sound.

Examples

A new pain animation for the imp for when he's electrocuted. Screams in pain multiple times.

Pain.Lightning:

TROO VW 3

TROO X 3 A_Pain

TROO VW 3

TROO X 3 A_Pain

TROO VW 3

TROO X 3 A_Pain

TROO VW 3

Goto See

A_PlayerScream

(no parameters)

Usage

Plays the player's death sound. It is important to use this over A_Scream for players because it has certain properties to it, such as checking the player's skin.

Examples

An alternative death animation for marine, screams later than usual.

Death.Impale:

```
PLAI A 3 A_Pain  
PLAI BCDDD 3  
PLAI DEDEDEDE 4  
PLAI E 5 A_PlayerScream  
PLAI DFG 5  
PLAI H -1  
stop
```

A_Scream

(no parameters)

Plays the actor's death sound if it has one. If the actor has the BOSS flag, the sound is heard at full volume regardless of distance, otherwise it diminishes by distance. The sound is played on the voice channel.

Examples

This is the Death state of Doom's Imp.

TROO I 8

TROO J 8 A_Scream

TROO K 6

TROO L 6 A_NoBlocking

TROO M -1

stop

A_SoundPitch

void A_SoundPitch (int slot, double pitch)

Usage

Sets the pitch of the sound currently playing on the specified channel.

Parameters

slot: the channel on which the sound is playing.

pitch: the pitch value to set.

A_SoundVolume

void A_SoundVolume (int slot, double volume)

Usage

Sets the volume of the sound currently being played by the actor on the specified channel.

Parameters

slot: the sound channel on which the sound is being played.

volume: the sound volume value to set. This ranges from 0.0 (mute) to 1.0 (full volume).

A_StartSound

clearscope void A_StartSound (sound whattoplay [, int slot [, int flags [, double volume [, double attenuation [, double pitch [, double startTime]]]]]])

Usage

Plays the specified sound.

Parameters

whattoplay: the desired sound to play, as defined in SNDINFO.

slot: the sound slot used for the sound. Unlike A_PlaySound, the slot is not limited to predefined slots and can be any number other than 0 (which instructs GZDoom to use another slot) and -1. The predefined slots are:

CHAN_AUTO (0) — Use the first channel that is not already playing something.

CHAN_WEAPON (1)

CHAN_VOICE (2)

CHAN_ITEM (3)

CHAN_BODY (4) — Default.

CHAN_5 (5)

CHAN_6 (6)

CHAN_7 (7)

flags: adjusts how the sound is played. Multiple flags can be combined by using the bitwise OR (|) operator:

CHANF_DEFAULT — No flags. Default.

CHANF_LISTENERZ — Played from the listener's Z-height. (Verification needed)

CHANF_MAYBE_LOCAL — Is subject to compat_silentpickup and will not play if the sound is made by an actor other than the local player's camera when the compatibility flag is enabled.

CHANF_UI — Is not preserved in savegames.

CHANF_NOPAUSE — Does not pause in menus.

CHANF_LOOP — Loops the sound.

CHANF_OVERLAP — Does not stop any sounds in the channel and instead plays over them.

CHANF_LOCAL — Only plays locally for the calling actor.

CHANF_NOSTOP — If the channel is occupied, do not play the sound.

CHANF_LOOPING — Combines CHANF_LOOP and CHANF_NOSTOP. It is equivalent to the looping parameter of A_PlaySound.

volume: the volume of the sound, which ranges from 0 to 1.0. Default 1.0.

attenuation: this is a positive value that specifies how quickly the sound fades with distance from its source. The exact formula for attenuation is: $\text{attenuation} = \text{default max hearable distance} / \text{desired max hearable distance}$. So, for example, in Doom the default max hearable distance is 1200; with attenuation of 20 the max hearable distance for the sound will be 60 map units ($1200 / 20 = 60$). This argument also accepts the following predefined constants:

ATTN_NONE — Plays the sound globally at the specified volume, disregarding distance.

ATTN_NORM — Uses the close_dist and clipping_dist fields defined in the sound definition. Default.

ATTN_IDLE — Uses Doom's normal default sound attenuation behavior.

ATTN_STATIC — Fades quickly (inaudible from 512 units).

pitch: the sound pitch to play the sound with. Default is 0, which means the engine uses whatever pitch shift (range) that is defined in SNDINFO, if any. For non-zero values anything lower than 1.0 will slow down the sound, while higher values speed it up.

startTime: sets how much of the sound to skip when starting. The value can be anywhere between 0 and 1.0, translating to 0% to 100%. Default is 0.

Examples

```
...
states
{
Spawn:
    BLAH A 5 NoDelay A_StartSound("play/sound")
    BLAH BCD 6
```

BLAH E -1
stop

...
}

A_StartSoundIfNotSame

clearscope void A_StartSoundIfNotSame (sound whatto play, sound check against [, int slot [, int flags [, double volume [, double attenuation [, double pitch [, double startTime]]]]]])

Usage

Plays the specified sound only if it is different from the one to check against.

The two sounds are considered different if they have different names. For aliases, the function checks the name of the sound which is referenced by the alias in order to make a decision. A random sound's assigned set of sounds is of no consequence, as the function only checks the sound's name.

Parameters

whatto play: the desired sound to play, as defined in SNDINFO.

check against: the sound to check against, as defined in SNDINFO.

slot: the sound slot used for the sound. Unlike A_PlaySound, the slot is not limited to predefined slots and can be any number other than 0 (which instructs GZDoom to use another slot) and -1. The predefined slots are:

CHAN_AUTO (0) — Use the first channel that is not already playing something.

CHAN_WEAPON (1)

CHAN_VOICE (2)

CHAN_ITEM (3)

CHAN_BODY (4) — Default.

CHAN_5 (5)

CHAN_6 (6)

CHAN_7 (7)

flags: adjusts how the sound is played. Multiple flags can be combined by using the bitwise OR (|) operator:

CHANF_DEFAULT — No flags. Default.

CHANF_LISTENERZ — Played from the listener's Z-height. (Verification needed)

CHANF_MAYBE_LOCAL — Is subject to compat_silentpickup and will not play if the sound is made by an actor other than the local player's camera when the compatibility flag is enabled.

CHANF_UI — Is not preserved in savegames.

CHANF_NOPAUSE — Does not pause in menus.

CHANF_LOOP — Loops the sound.

CHANF_OVERLAP — Does not stop any sounds in the channel and instead plays over them.

CHANF_LOCAL — Only plays locally for the calling actor.

CHANF_NOSTOP — If the channel is occupied, do not play the sound.

CHANF_LOOPING — Combines CHANF_LOOP and CHANF_NOSTOP. It is equivalent to the looping parameter of A_PlaySound.

volume: the volume of the sound, which ranges from 0 to 1.0. Default 1.0.

attenuation: this is a positive value that specifies how quickly the sound fades with distance from its source. The higher the value the quicker it fades out. The numbers are fairly low. A dramatic drop off of volume is noticeable with an attenuation value of just 3 or 4. With a value of 20, for instance, the player needs to be within around 64 units of the sound source to hear it clearly. The following predefined constants exist:

ATTN_NONE — Plays the sound globally at the specified volume, disregarding distance.

ATTN_NORM — Uses the close_dist and clipping_dist fields defined in the sound definition. Default.

ATTN_IDLE — Uses Doom's normal default sound attenuation behavior.

ATTN_STATIC — Fades quickly (inaudible from 512 units).

pitch: the sound pitch to play the sound with. Default is 0, which means the engine uses whatever pitch shift (range) that is defined in SNDINFO, if any. For non-zero values anything lower than 1.0 will slow down the sound, while higher values speed it up.

startTime: sets how much of the sound to skip when starting. The value can be anywhere between 0 and 1.0, translating to 0% to 100%. Default is 0.

A_StopAllSounds

void A_StopAllSounds ()

Usage

Stops all the sounds that are playing by the calling actor. This function is the equivalent of using A_StopSounds while passing 0 to both parameters.

A_StopSound

A_StopSound [(int slot)]

Usage

Stops the sound currently playing on the specified channel for the calling actor. The function can only stop sounds that have an actor source. Sounds which are played locally (by calling A_PlaySound with local set to true) are source-less, thus cannot be stopped by this function.

Parameters

slot: the sound channel on which the sound to stop is playing. Default is CHAN_VOICE.

Examples

This new fireball plays a loop sound when it spawns, and when it comes to "Death" state, it stops the sound and the projectile explodes.

ACTOR ImpFireBall : DoomImpBall

```
{
    States
    {
    Spawn:
        BAL1 A 0 NoDelay A_PlaySound("Fireball/Loop", CHAN_6, 1, TRUE)
        BAL1 AB 4 Bright
        Loop
    Death:
        BAL1 C 0 A_StopSound(CHAN_6)
        BAL1 CDE 6 Bright
        Stop
    }
}
```

A_StopSounds

void A_StopSounds (int chanmin, int chanmax)

Usage

Stops the sounds that are playing by the calling actor on the specified range of sound channels. If 0 is passed to both parameters, then the function acts exactly like A_StopAllSounds, stopping all the sounds that are playing by the calling actor.

Parameters

chanmin: the start of the sound channels range.

chanmax: the end of the sound channels range.

A_VileStart

(no parameters)

Note: This function has been superseded by A_StartSound, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

Usage

Plays the sound "vile/start" on the Voice channel.

Examples

The following two DECORATE code blocks are equivalent.

```
...
MONS HI 5
MONS J 5 A_VileStart
...
...
MONS HI 5
MONS J 5 A_PlaySound("vile/start", CHAN_VOICE)
...
```

A_XScream

Note: This function has been superseded by A_StartSound, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

A_XScream

(no parameters)

Plays the *gibbed player sound if called by a player, otherwise it plays misc/gibbed. The sound is played on the voice channel.

Examples

XDeath:

POSS M 5

POSS N 5 A_XScream

POSS O 5 A_NoBlocking

POSS PQRST 5

POSS U -1

stop

A_SpawnDebris

A_SpawnDebris (string type [, bool translation [, float mult_h [, float mult_v]]])

Usage

Spawns debris actors based on the specified type and positions them around the calling actor. The debris type has to be defined in a certain way:

Its health parameter specify the amount n of objects being spawned.

The first n states are reserved as the initial states for the separate objects. If you want to do some animations you can jump to later states from here.

Parameters

type: The class name of the debris actor to spawn.

translation: If this is TRUE, the spawned actor will be assigned the same translation table as the actor that called the function. Default is FALSE.

mult_h: This is a multiplier for the x and y velocities of the spawned actor. Default is 1.0.

mult_v: This is a multiplier for the z velocity of the spawned actor. Default is 1.0.

Examples

This is an example of a valid debris class:

ACTOR SentinelDebris

```
{
  Health 15
  Radius 1
  Height 1
  States
  {
  Spawn:
    SNT1 A -1
    SNT2 A -1
    SNT3 A -1
    SNT3 A -1
    SNT4 A -1
    SNT4 A -1
    SNT5 A -1
    SNT6 A -1
    SNT7 A -1
    SNT7 A -1
    SNT8 A -1
    SNT8 A -1
    SNT9 A -1
    SNT9 A -1
    SNT0 A -1
  }
}
```

A_SpawnItem

bool, Actor A_SpawnItem [(class<Actor> missile [, double distance [, double zheight [, bool useammo [, bool transfer_translation]]]])]

Note: This function has been superseded by A_SpawnItemEx, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

Usage

Spawns the specified actor at the specified distance and height relative to the calling actor.

The function does not spawn monster-based actors if the calling actor was massacred or there is not enough space for them to spawn.

If both the originator and spawned actor are monster based, the spawned actor becomes affiliated with the originator. If, however, the originator is a player, the spawned actor, in this case, becomes a friendly monster which is allied with that player.

Parameters

missile: the name of the actor to spawn. This can be any actor type, not just missiles. Default is "Unknown".

distance: the distance from the calling actor to spawn the actor. Positive values shift the spawn point forwards, while negative values shift it backwards. Default is 0.

zheight: how far from the bottom of the calling actor to spawn the actor. Positive values shift the spawn point upwards, while negative values shift it downwards. Default is 0.

useammo: whether or not ammo is required in order to spawn the actor if the function is called from a player's weapon, as well as whether or not to form a master/minion relationship between the originator and the spawned actor. Master and minions do not attack each other, and this relationship allows the use of functions that affect one another, such as A_DamageMaster and A_GiveToChildren. If this is true, ammo is required and the relationship is formed, otherwise, if this is false, ammo is not required and the relationship is not formed. Default is true.

transfer_translation: if this is true, the spawned actor uses the same translation as the calling actor, provided the spawned actor can be translated. Default is false.

Originator

The originator is the actor that is considered to be the "spawner" actor. Normally, the originator is the same as the calling actor of the function, except in the case of missile-based actors. A missile-based actor cannot be the originator, since the originator is whatever non-missile actor that it considers to be its spawner or shooter. If this actor does not exist, then there is no originator in this case.

Return value

The function returns two values:

A boolean based on whether the actor successfully spawned or not. An actor that does not spawn because of lack of ammo or massacre always counts as success.

A pointer to the spawned actor. If the actor fails to spawn due to lack of ammo or massacre, the returned value is null. When testing for space to spawn the actor in, the actor is spawned, tested and then immediately destroyed if it fails the test. The returned value in this case is still a pointer to the actor, not null. The space test is performed on monster-based actors only.

So based on the above, it is important to check for both of these values in order to determine whether an actor has truly spawned or not.

Examples

```
class TerranMarine : ScriptedMarine2
{
    Default
```

```
{
    Obituary "%o was killed by a Terran Marine.";
    Translation "112:127=198:207";
    DropItem "Clip", 256;
    DropItem "GreenMedikit", 128;
    DropItem "GreenStimpack", 128;
    DropItem "GreenArmor", 16;
    DropItem "BlueArmor", 8;
    DropItem "ArmorRed", 4;
    DropItem "ArmorGray", 2;
    +FRIENDLY
    ActiveSound "starcraft/active";
    DeathSound "starcraft/death";
}
```

States

```
{
Spawn:
    PLY2 A 4 A_Look;
    PLY2 B 4;
    Goto See;
See:
    PLY2 AABBCDD 2 A_FastChase;
    PLY2 A 0 NoiseAlert(0, 0);
    Loop;
```

Pain:

```
    PLY2 G 4;
    PLY2 G 4 A_Pain;
    Goto Spawn+3;
```

Missile:

```
    PLY2 E 8 A_FaceTarget;
    PLY2 E 2 A_PlaySound("starcraft/marinef1", CHAN_WEAPON);
    PLY2 F 2 A_CustomBulletAttack(5.6, 0, 1, 5, "BulletPuff", 0);
    PLY2 E 2 A_FaceTarget;
    PLY2 F 2 A_CustomBulletAttack(5.6, 0, 1, 5, "BulletPuff", 0);
    PLY2 E 2 A_FaceTarget;
    PLY2 F 2 A_CustomBulletAttack(5.6, 0, 1, 5, "BulletPuff", 0);
    PLY2 E 10 A_CPosRefire;
    Goto Missile+1;
```

Death:

```
    PLY2 H 10;
    PLY2 I 10 A_Scream;
    PLY2 J 10 A_NoBlocking;
    PLY2 KLM 10;
    PLY2 N 0 A_SpawnItem("SpawnTMarine");
    PLY2 N 512;
    Goto DDisp;
```

DDisp:

```
    PLY2 N 4 A_FadeOut(0.05);
    Loop;
```

XDeath:

```
    PLY2 O 5;
    PLY2 P 5 A_XScream;
    PLY2 Q 5 A_NoBlocking;
```

```

    PLY2 RSTUV 5;
    PLY2 W 0 A_SpawnItem("SpawnTMarine");
    PLY2 W 512;
    Goto XDisp;
XDisp:
    PLY2 W 4 A_FadeOut(0.05);
    Loop;
Raise:
    PLY2 MLKJIH 5;
    Goto See;
}
}

class SpawnTMarine : Actor
{
    States
    {
    Spawn:
        ALLY A 128;
        TFOG A 8
        {
            A_PlaySound("misc/teleport");
            A_SpawnItemEx("TerranMarine", 0, 0, 0, 0, 0, 0, 0, 64, 0);
        }
        TFOG BCDEFGHIJ 8;
        Stop;
    }
}

```

A_SpawnItemEx

bool, Actor A_SpawnItemEx (class<Actor> missile [, double xofs [, double yofs [, double zofs [, double xvel [, double yvel [, double zvel [, double angle [, int flags [, int failchance [, int tid]]]]]]]]))

Usage

Spawns an actor at the specified offset away from the calling actor, facing the given angle, and with the indicated velocity.

The function does not spawn monster-based actors if the calling actor was massacred.

Parameters

missile: The name of the actor to spawn. This can be any actor type, not just missiles.

x-/y-/zofs: The offset from the calling actor at which to spawn the new actor.

xofs: positive numbers move the spawn point further away in front of the calling actor; negative numbers - behind it.

yofs: positive numbers move the spawn point to the right side of the actor, negative numbers - to the left side.

zofs: positive numbers move the spawn point up from the bottom of the calling actor, negative numbers move it down.

x-/y-/zvel: The initial velocity to give to the spawned actor, useful for projectiles.

xvel: forward/backward velocity of the spawned actor.

yvel: right/left sideways velocity of the spawned actor.

zvel: vertical velocity of the spawned actor.

angle: Offsets the spawned actor's angle from the calling actor by the specified amount. Positive numbers rotate the spawned actor to the left, negative numbers turn it to the right in relation to the actor.

flags: The following flags can be combined by using the | character between the constant names:

SXF_TRANSFERTRANSLATION - the spawned actor uses the same translation as the calling actor, provided the spawned actor can be translated. This flag takes precedence over SXF_USEBLOODCOLOR.

SXF_ABSOLUTEPOSITION - applies the spawn offsets according to the absolute XY axes of the map, rather than relative to the direction the calling actor is facing.

SXF_ABSOLUTEANGLE - use the angle parameter as an absolute angle instead of a relative one to the calling actor's angle.

SXF_ABSOLUTEVELOCITY - the spawned actor will have its velocity set in absolute values instead of relative derived from the calling actor's velocities along the X, Y and Z axis.

SXF_SETMASTER - forms a master/minion relationship between the originator and the spawned actor. Master and minions do not attack each other, and this relationship allows the use of functions that affect one another, such as A_DamageMaster and A_GiveToChildren. This flag takes precedence over SXF_TRANSFERPOINTERS when setting the master field.

SXF_NOCHECKPOSITION - do not check the destination for room before spawning. This flag is only meaningful if the spawned actor is monster based, as only actors of this type are tested for room normally.

SXF_TELEFRAG - kills any actor that would prevent the new actor from spawning. This flag implies

SXF_NOCHECKPOSITION.

SXF_CLIENTSIDE - client-side spawning only. SkulltagIcon22.png (Skulltag only: not supported by ZDoom)

SXF_TRANSFERAMBUSHFLAG - if the calling actor has the AMBUSH flag set, then the spawned actor will have it as well.

SXF_TRANSFERPITCH - transfers the calling actor's pitch to the spawned actor. Note that this has no effect on the spawned actor's velocities. If you want the calling actor's pitch to be taken into account for the spawned actor's trajectory, use a formula such as this one: A_SpawnItemEx (<type>,cos(pitch)*<Dist. from spawner>,<yofs>,<zofs>- (sin(pitch)*<Dist. from spawner>),cos(pitch)*<Projectile speed>,0,-sin(pitch)*<Projectile speed>,<angle>,<flags>,<chance>)

SXF_TRANSFERPOINTERS - transfers the calling actor's master, target, and tracer fields to the spawned actor.

SXF_SETMASTER, SXF_SETTARGET and SXF_SETTRACER take precedence over this flag.

SXF_USEBLOODCOLOR - uses the calling actor's BloodColor as the source of translation to be applied to the spawned actor, provided the spawned actor can be translated. If no BloodColor is specified, no translation is applied.

SXF_TRANSFERTRANSLATION takes precedence over this flag.

SXF_CLEARCALLERTID - if the actor is spawned successfully, the calling actor will have its TID set to 0.

SXF_MULTIPLYSPEED - multiplies the spawned actor's velocity by its Speed property.

SXF_TRANSFERSCALE - transfers the current scale factor of the calling actor to the spawned actor.

SXF_TRANSFERSPECIAL - transfers the calling actor's special and arguments to the spawned actor.

SXF_CLEARCALLERSPECIAL - if the actor is spawned successfully, the calling actor will have its special and arguments removed.

SXF_TRANSFERSTENCILCOL - transfers the calling actor's StencilColor to the spawned actor.

SXF_TRANSFERALPHA - transfers the calling actor's alpha to the spawned actor.

SXF_TRANSFERRENDERSTYLE - transfers the calling actor's render style to the spawned actor.

SXF_SETTARGET - sets the originator as the spawned actor's target. This flag takes precedence over SXF_TRANSFERPOINTERS when setting the target field.

SXF_SETTRACER - sets the originator as the spawned actor's tracer. This flag takes precedence over SXF_TRANSFERPOINTERS when setting the tracer field.

SXF_NOPOINTERS - clears the spawned actor's target, master and tracer fields. If the spawned actor is monster based, it also prevents it from becoming affiliated with the originator if the originator is also monster based, or becoming a friendly monster that is allied with the originator if it is a player. SXF_SETTARGET, SXF_SETMASTER and SXF_SETTRACER take precedence over this flag, while this flag takes precedence over SXF_TRANSFERPOINTERS.

SXF_ORIGINATOR - explicitly sets the calling actor as the originator. This is only meaningful if the calling actor is missile based, as actors of this type cannot be originators normally.

SXF_TRANSFERSPRITEFRAME - transfers the calling actor's current sprite and frame letter to the spawned actor. This only works if the spawned actor's first spawn sprite (and/or frame) has the reserved special name for the sprite and frame letter.

SXF_TRANSFERROLL - transfers the calling actor's roll angle to the spawned actor.

SXF_ISTARGET - the spawned actor becomes the calling actor's target.

SXF_ISMASTER - the spawned actor becomes the calling actor's master.

SXF_ISTRACER - the spawned actor becomes the calling actor's tracer.

failchance: This is the chance (out of 256) that the actor has of not spawning. If this is 0, the actor's chance of spawning is 100%. If it is 256, it will never spawn. Default is 0.

tid: The TID to assign to the spawned actor. Passing tid, allows for the spawned actor to have the same TID as the calling actor. In conjunction with SXF_CLEARCALLERTID, this can be used to transfer a TID from the calling actor to the spawned actor.

Originator

The originator is the actor that is considered to be the "spawner" actor. Normally, the originator is the same as the calling actor of the function, except in the case of missile-based actors. A missile-based actor cannot be the originator unless it is explicitly set to be (see SXF_ORIGINATOR). If not, then the originator is whatever non-missile actor that it considers to be its spawner or shooter. If this actor does not exist, then there is no originator in this case.

Return value

The function returns two values:

A boolean based on whether the actor successfully spawned or not. An actor that does not spawn because of chance or massacre always counts as success.

A pointer to the spawned actor. If the actor fails to spawn due to chance or massacre, the returned value is null. When testing for space to spawn the actor in, the actor is spawned, tested and then immediately destroyed if it fails the test. The returned value in this case is still a pointer to the actor, not null. The space test is performed on monster-based actors only.

So based on the above, it is important to check for both of these values in order to determine whether an actor has truly spawned or not.

Examples

The following actor is a variant of the Heretic explosive pod that leaves a poison cloud behind it as well. It uses A_SpawnItemEx so as to make use of the SXF_TRANSFERPOINTERS flag. This way, the monster or player responsible for blowing up the pod will be known as the source of damage created by the poison cloud. This makes sure that deathmatch frags are credited to the proper player, and allows barrel-based infighting to happen. The yofs is set to

28 so as to mimic closely the A_PoisonBagInit function which is not accessible to pod actors.

```
class PoisonPod : Pod
{
    Default
    {
        DeathSound "PoisonPod/Puff";
    }

    States
    {
        Death:
            PPOD C 5 Bright A_RemovePod;
            PPOD D 5 Bright A_Scream;
            PPOD E 5 Bright A_Explode;
            PPOD F 10 Bright A_SpawnItemEx("PoisonCloud", 0, 0, 28, 0, 0, 0, 0, SXF_TRANSFERPOINTERS);
            Stop;
    }
}
```

A_SpawnParticle

```
void A_SpawnParticle (color color1 [, int flags [, int lifetime [, double size [, double angle [, double xoff [, double yoff [, double zoff [, double velx [, double vely [, double velz [, double accelx [, double accely [, double accelz [, double startalphaf [, double fadestepf [, double sizestep]]]]]]]]]]]]]]]]]]]]]]))
```

Usage

Spawns a single particle.

Parameters

color1: The color of the particle. Can be used with a hexadecimal value or a predefined value such as "Black". Set this to FFFFFFFF to spawn

flags: Customizes the behavior of the function. Multiple flags can be combined by using the bitwise OR operator (|) between the constant names:

SPF_FULLBRIGHT — Makes the particle full bright.

SPF_RELPOS — Position is relative to angle.

SPF_RELVEL — Velocity is relative to angle.

SPF_RELACCEL — Acceleration is relative to angle.

SPF_RELANG — Adds the calling actor's angle to angle for relativity.

SPF_NOTIMEFREEZE — The spawned particle is not affected by the time freeze powerup or cheat.

SPF_REPLACE — If the the particle limit is reached, the oldest particles will be removed to make room for particles with SPF_REPLACE. (New from 4.10.0)

SPF_NO_XY_BILLBOARD - The particle does not have any sort of billboarding, causing it to render similarly to normal actor sprites, instead of facing the players' view at all times. (New from 4.10.0)

SPF_RELATIVE — Combines the SPF_RELPOS, SPF_RELVEL, SPF_RELACCEL and SPF_RELANG flags.

Default is 0.

lifetime: The lifetime of the particle in tics. Default is 35.

size: The size of the particle. Default is 1.

angle: The angle to offset the particle by. Default is 0.

x/yoff: The distance from the actor to spawn the particle along the X axis. Note that this is not relative.

Default is 0.

zoff: How high up to spawn the particle from the actor's Z position. Default is 0.

velx/y/z: The velocity along the X/Y/Z axis to apply to the particle. This is in absolute direction, not relative.

Default is 0.

accelx/y/z: Defines how much to accelerate the particle by over its lifespan. Default is 0.

startalphaf: Specifies the particle's alpha upon spawning. Default is 1.0.

fadestepf: The amount by which the particle fades each tic. The particle is automatically removed early if it fades completely before lifetime expires. -1 indicates automatic (a complete fade-out over the length of lifetime). Default is -1.

sizestep: The particle grows or shrinks in size by this amount per tic.

A_SpawnParticleEx (New from 4.9.0)

Note: This feature is for ZScript only.

```
void A_SpawnParticleEx (color color1 [, TextureID texture[, int style[, int flags [, int lifetime [, double size [, double angle [, double xoff [, double yoff [, double zoff [, double velx [, double vely [, double velz [, double accelx [, double accely [, double accelz [, double startalphaf [, double fadestepf [, double sizestep[, double startroll[, double rollvel[, double rollacc]]]]]]]]]]]]]]]]]]]]]]))
```

Note: There is currently a bug with the handling of textured particles, that makes them appear upside down.

Usage

Spawns a single particle. Unlike A_SpawnParticle, this function also allows for creating textured particles, animated textured particles (New from 4.10.0), shading particles, and giving them roll.

Parameters

color1: The color of the particle. Can be used with a hexadecimal value or a predefined value such as "Black". Colors can also be applied to textured particles to tint them a certain color.

texture: The texture to use for the particle, this requires a TextureID. Can be any graphic type, such as textures, sprites, miscellaneous graphics etc. This also supports graphics animated through ANIMDEFS. (New from 4.10.0)

style: The render style to apply to the particle, these styles can be used both on textured and non-textured particles. Available render styles include:

STYLE_None/STYLE_Normal/STYLE_Translucent: The particle is rendered normally, and can have custom alpha. All these styles have the same effect on particles. The default style is STYLE_None.

STYLE_Add: Particle uses additive translucency.

STYLE_Stencil: Particle is drawn ONLY with a the color defined in color1, this is only really useful for textured particles.

STYLE_AddStencil: Same as STYLE_Stencil, but additive.

STYLE_Fuzzy: Creates undefined behavior with textured and untextured particles.

STYLE_Subtract: Uses inverted additive translucency, darkening the particle instead of making it lighter.

STYLE_Shadow: Creates an effect similar to giving the particle STYLE_Stencil and giving it a startalphaf of 0.3.

flags: Customizes the behavior of the function. Multiple flags can be combined by using the bitwise OR operator (|) between the constant names:

SPF_FULLBRIGHT — Makes the particle full bright.

SPF_RELPOS — Position is relative to angle.

SPF_RELVEL — Velocity is relative to angle.

SPF_RELACCEL — Acceleration is relative to angle.

SPF_RELANG — Adds the calling actor's angle to angle for relativity.

SPF_NOTIMEFREEZE — The spawned particle is not affected by the time freeze powerup or cheat.

SPF_ROLL - The particle is allowed to use its' startroll, rollvel, and rollacc parameters.

SPF_REPLACE — If the the particle limit is reached, the oldest particles will be removed to make room for particles with SPF_REPLACE. (New from 4.10.0)

SPF_NO_XY_BILLBOARD - The particle does not have any sort of billboard, causing it to render similarly to normal actor sprites, instead of facing the players' view at all times. (New from 4.10.0)

SPF_RELATIVE — Combines the SPF_RELPOS, SPF_RELVEL, SPF_RELACCEL and SPF_RELANG flags. Default is 0.

lifetime: The lifetime of the particle in tics. Default is 35.

size: The size of the particle. Default is 1.

angle: The angle to offset the particle by. Default is 0.

x/yoff: The distance from the actor to spawn the particle along the X axis. Note that this is not relative. Default is 0.

zoff: How high up to spawn the particle from the actor's Z position. Default is 0.

velx/y/z: The velocity along the X/Y/Z axis to apply to the particle. This is in absolute direction, not relative. Default is 0.

accelx/y/z: Defines how much to accelerate the particle by over its lifespan. Default is 0.

startalphaf: Specifies the particle's alpha upon spawning. Default is 1.0.

fadestepf: The amount by which the particle fades each tic. The particle is automatically removed early if it fades completely before lifetime expires. -1 indicates automatic (a complete fade-out over the length of lifetime). Default is -1.

sizestep: The particle grows or shrinks in size by this amount per tic.

startroll: The amount of roll the particle starts with. Default is 0. SPF_ROLL must be set for this parameter to be used.

rollvel: The velocity at which the particle rotates. Default is 0. SPF_ROLL must be set for this parameter to be used.

rollacc: How much the particle will accelerate its' roll over its' lifespan. Default is 0. SPF_ROLL must be set for this parameter to be used.

Examples

This actor spawns additive textured particles that use the sprites of the Imps' fireball, these particles then randomly pick if they should roll left or right, and accelerate their roll over time. The particles also increase in scale and fade out slowly.

Class TexturedParticleExample : Actor

```
{
    Override Void Tick()
    {
        Bool Left = Random (True,False);
        A_SpawnParticleEx
        (
            "FFFFFF",
            TexMan.CheckForTexture ("BAL1A0"),
            style: STYLE_Add,
            flags: SPF_FULLBRIGHT|SPF_RELATIVE|SPF_ROLL,
            lifetime: TICRATE * FRandom (1,4),
            size: 0.5,
            angle: 0.,
            xoff: FRandom (64,-64),
            yoff: FRandom (64,-64),
            zoff: 0.,
            velx: FRandom (0.5,-0.5),
            vely: FRandom (0.5,-0.5),
            velz: FRandom (0.4,3.0),
            accelx: 0,
            accely: 0,
            accelz: -0.001,
            startalphaf: 1.25,
            fadeStepf: -0.002,
            sizestep: 0.25,
            startroll: 180/2,
            rollvel: Left ? 0.5 : -0.5,
            rollacc: Left ? 0.02 : -0.02
        );
    }
}
```

Animated particles example
(New from 4.10.0)

In addition to being able to use static graphics as particles, you can also use animated textures and graphics as particles as well. Below is an example on how the fireball particle spawner above can be animated using the Imp fireballs' original sprites.

The below TEXTURES definition creates new fireball textures using the original sprites as patches.

```
// Texture definitions generated by SLADE3
// on Fri Nov 11 05:41:11 2022
```

```
Texture "ANIMBAL1", 15, 15
{
    Patch "BAL1A0", 0, 0
}
```

```
Texture "ANIMBAL2", 15, 15
{
    Patch "BAL1B0", 0, 0
}
```

```
// End of texture definitions
```

Now that the original Imp fireball sprites are used to create new textures. ANIMDEFS can be used to turn the new textures into an animated texture.

Texture ANIMBAL1 Range ANIMBAL2 Tics 8

The sprites have now been turned into an animated texture. And you can change the graphic used by the texture parameter to one that is part of the animated texture. Causing the particles to turn into fully animated Imp fireballs.

A_TossGib

(no parameters)

Spawns one actor of type Meat (for bleeding monsters) or Junk (for everything else) and tosses it in a random direction.

Actors of type Meat randomly show a sprite from MEATAxâ€œMEATTx.

Actors of type Junk randomly show a sprite from JUNKAxâ€œJUNKTx.

A CopySpriteFrame

A_CopySpriteFrame(int from, int to[, int flags])

Usage

Copies a sprite and/or frame from the from pointer to the to pointer.

Parameters

from - The pointer to copy the sprite frame from.

to - The pointer to have their sprite frames replaced.

flags - Can be one of the two following (does nothing if both are used):

CPSF_NOSPRITE - The receiver maintains their current sprite.

CPSF_NOFRAME - The receiver maintains their current frame.

A SetSpriteAngle

A_SetSpriteAngle(float angle[, int ptr])

Usage

Sets the absolute sprite angle to use when coupled with the SPRITEANGLE flag.

Parameters

angle - The sprite found at the angle to use.

ptr - The Actor pointer to modify.

A SetSpriteRotation

A_SetSpriteRotation(float angle[, int ptr])

Usage

Sets the rotation of the sprite without modifying the actor's angle property.

Parameters

angle - The angle amount the sprite will be turned. Positive values turn it to the right, negative to the left.

ptr - The Actor pointer to modify.

A_BossDeath

(no parameters)

Checks whether all monsters of the calling type are dead, and if so, executes all special actions that are assigned to this type. Special actions have to be assigned to a monster with the SpecialAction MAPINFO command.

Examples

The Baron of Hell, for example, uses A_BossDeath in its Death sequence - that is called in E1M8 when both barons have died, and thus performs the map's special action (in this case, lower the walls).

Death:

```
BOSS I 8  
BOSS J 8 A_Scream  
BOSS K 8  
BOSS L 8 A_NoBlocking  
BOSS MN 8  
BOSS O -1 A_BossDeath  
Stop
```

Interaction with actor replacement

If the calling actor's class replaces actors of another class, this function will also trigger special actions for that other class. If more than one actor class is declared as replacing the same base actor, and actors of both classes are present on the map at the same time, then the same special action may be triggered more than once, potentially breaking the map.

For example, if a mod adds several variants of the Arachnotron (all of which are declared with replaces Arachnotron), and spawns the different variants at random (with an actor that also replaces Arachnotron), then the map may break when the player kills them all. (This won't happen when using a RandomSpawner, because it specially handles this situation.)

A_BrainDie

(no parameters)

Exits the level if not in deathmatch mode.

Examples

This example is taken from Doom's Icon of Sin.

Death:

BBRN A 100 A_BrainScream

BBRN AA 10

BBRN A -1 A_BrainDie

Stop

A_CheckPlayerDone

(no parameters)

Removes the actor unless it is currently assigned to a player. This can be used in the player's death state to ensure the player has respawned before removing the corpse.

A_CheckTerrain

(no parameters)

Applies the effects of sector types 115 (instant death) and 118 (Strife's water scroller) to the calling actor.

Used in Strife for the shooting range targets and for the converted peasants, to move or remove them according to the sector they are currently in.

Examples

Strife's 'zombie' peasant actor:

ACTOR Zombie : StrifeHumanoid 169

```
{
    Game Strife
    Health 31
    Radius 20
    Height 56
    PainChance 0    // is put into Pain state only by a Teleport_ZombieChanger line
    +SOLID
    +SHOOTABLE
    +FLOORCLIP
    +CANPASS
    +CANPUSHWALLS
    +ACTIVATEMCROSS
    MinMissileChance 150
    MaxStepHeight 16
    MaxDropOffHeight 32
    Translation 0
    ConversationID 28, -1, -1
    DeathSound "zombie/death"
    States
    {
    Spawn:
        PEAS A 5 A_CheckTerrain    // looks like a Peasant
        Loop
    Pain:
        AGRD A 5 A_CheckTerrain    // looks like an Acolyte
        Loop
    Death:
        GIBS M 5 A_TossGib
        GIBS N 5 A_XScream
        GIBS O 5 A_NoBlocking
        GIBS PQRST 4 A_TossGib
        GIBS U 5
        GIBS V 1400
        Stop
    }
}
```

A_GetHurt

(no parameters)

Usage

Decreases the calling actor's health by 1 with a probability of 20%. If the health goes below 0 the actor dies. Main usage is in Wound state.

Examples

This is a rotting zombie, a zombie that is slowly decomposing and will eventually die. Its See, Missile, and even its Pain states all trigger A_GetHurt, lowering its health, and eventually killing it if the player doesn't first. Its pain sound was removed so that it wouldn't make a noise every time A_GetHurt was triggered successfully.

ACTOR RottingZombie : ZombieMan

```
{
  PainSound ""
  States
  {
    See:
      POSS AABB 4 A_Chase
      POSS B 0 A_GetHurt
      POSS CCDD 4 A_Chase
      POSS D 0 A_GetHurt
      Loop
    Missile:
      POSS E 10 A_FaceTarget
      POSS F 8 A_PosAttack
      POSS E 8 A_GetHurt
      Goto See
    Pain:
      POSS G 3 A_GetHurt
      POSS G 3 A_Pain
      Goto See
  }
}
```

A_KeenDie

A_KeenDie [(int tag)]

Checks whether all actors of the calling actor's type are dead and if so opens all doors with the tag 666.

If tag is specified, it will be used in place of 666.

Examples

This example is taken from Doom's Commander Keen decoration.

Death:

```
KEEN AB 6
KEEN C 6 A_Scream
KEEN DEFGH 6
KEEN I6 A_NoBlocking
KEEN J 6
KEEN K 6 A_KeenDie
KEEN L -1
Stop
```

A_KlaxonBlare

(no parameters)

Checks whether a player that has made some noise or triggered an alert is nearby. If so it will play the sound "misc/alarm".

If no player can be found that meets that criteria it will unalert all monsters in nearby sectors.

A_PlayerSkinCheck

state A_PlayerSkinCheck (str state)

Checks if the player is currently using a custom Skin made for another game and if so, jumps to the specified state. This function allows the default DoomGuy to use a Corvus skin and vice-versa. The number of death frames in Doom is different from the number of death frames in Heretic, which would without this function make skins from one game glitchy in another.

This function should only be used for Doom or Heretic custom classes that are allowed to use custom skins.

Examples

The Heretic Player uses this function because it has a larger amount of frames from what custom Doom skins have. Using A_PlayerSkinCheck in both death states, it checks first if the player is using a Doom skin and then jumps to a different death state if he/she is.

Death:

```
PLAY H 6 A_PlayerSkinCheck("AltSkinDeath")
PLAY I 6 A_PlayerScream
PLAY JK 6
PLAY L 6 A_NoBlocking
PLAY MNO 6
PLAY P -1
Stop
```

XDeath:

```
PLAY Q 0 A_PlayerSkinCheck("AltSkinXDeath")
PLAY Q 5 A_PlayerScream
PLAY R 0 A_NoBlocking
PLAY R 5 A_SkullPop
PLAY STUVWX 5
PLAY Y -1
Stop
```

AltSkinDeath:

```
PLAY H 10
PLAY I 10 A_PlayerScream
PLAY J 10 A_NoBlocking
PLAY KLM 10
PLAY N -1
Stop
```

AltSkinXDeath:

```
PLAY O 5
PLAY P 5 A_XScream
PLAY Q 5 A_NoBlocking
PLAY RSTUV 5
PLAY W -1
Stop
```

A_Quake

A_Quake (float intensity, int duration, int damrad, int tremrad [, sound sfx])

Note: For a more advanced and flexible version of spawning quakes with different behaviors, see A_QuakeEx.

Note: As of (development version aed72f5 only) the intensity parameter is a decimal value instead of an integer.

Creates an earthquake around the calling actor.

intensity: Strength of earthquake, ranging from 0 to 9

duration: Duration in tics

damrad: Radius of damage in map units

tremrad: Radius of tremor in map units

sound: Accompanying sound effect for the tremor. (Default: "world/quake".)

Contrarily to the Radius_Quake action special, the radii for damage and tremor are given directly in map units, not in "tiles" of 64x64 map units. This must be kept in mind when updating an actor definition from Radius_Quake to A_Quake.

Examples

This large rocket causes an earthquake for about one third of a second upon exploding. The quake is felt within 800 units of the impact, but at a higher intensity within 400 units.

actor BigRocket : Rocket

```
{
  Scale 2.0
  Radius 22
  Height 16
  Speed 25
  states
  {
    Death:
      MISL B 0 A_Quake(4,12,0,400)
      MISL B 0 A_Quake(2,12,0,800)
      MISL B 8 bright A_Explode(250,200,1)
      goto Super::Death+1
  }
}
```

A_QuakeEx

A_QuakeEx (float intensityX, float intensityY, float intensityZ, int duration, int damrad, int tremrad [, sound sfx [, int flags [, float mulwavex [, float mulwavey [, float mulwavez[, int falloff[, int highpoint[, float rollIntensity[, float rollWave]]]]]]]]))

Note: As of (development version aed72f5 only) the intensity parameters are decimal values instead of integers.

Usage

A_QuakeEx is an extended version of A_Quake with the ability to create an earthquake effect to manipulate the player's view along a specific axis, either in direct map axis or along the camera based upon the flags. It also allows adjusting the Z axis (quaking up and down). It also follows the same functionality, and creates an earthquake around the calling actor. In addition, it can also manipulate the ins and outs of an earthquake.

Wave quakes can stack, adding their effects on top of other wave quakes. This does not affect non-wave quakes any differently, however.

Parameters

intensityX/Y/Z: Strength of earthquake, ranging from 0 to 9, along a particular axis.

duration: Duration in tics

damrad: Radius of damage in map units, things that are not on the ground are unaffected.

tremrad: Radius of tremor in map units

sound: Accompanying sound effect for the tremor. (Default: "world/quake".) The sound is played on the CHAN_BODY sound channel. Passing "" will make the quake silent; however, it'll still occupy the CHAN_BODY channel "â€"

meaning, playing a sound on the CHAN_BODY channel with A_StartSound, and then calling A_QuakeEx will immediately stop the previous sound. There's currently no way to make this function not occupy a sound channel at all.

flags: Can be combined with the pipe "|" character. All flags are compatible with each other.

QF_RELATIVE: Adjust the X, Y and Z intensities to quake along the camera view's front and side respectively.

QF_SCALEDOWN: Scales the intensity over the duration, going from full at the start of the quake to 0 upon finishing. Can be combined with QF_SCALEUP.

QF_SCALEUP: Scales the intensity over the duration, going from 0 at the start of the quake to full upon finishing. Can be combined with QF_SCALEDOWN.

QF_WAVE: Changes the quake from a randomly generated one to a sine wave, and are further controlled by mulwavex/y/z. Intensity is known as 'amplitude' in this form.

QF_MAX: Requires QF_SCALEDOWN and QF_SCALEUP. Fully scaled quakes will gradually scale from 0 to half intensity, and back to 0. This changes 0 to start from the defined intensity instead.

QF_FULLINTENSITY: Requires QF_SCALEDOWN and QF_SCALEUP. Fully scaled quakes will only scale in to the half intensity from their origins. This changes half to full intensity instead.

QF_3D: Makes the screen shake of the earthquake also fall off based on how far away the player is on the Z axis from the source of the earthquake.

QF_GROUNDONLY: The screen shake of the earthquake will stop when the player or their camera is off the ground, similar to real world earthquakes. (New from 4.10.0)

QF_AFFECTACTORS: The damrad property will also be able to harm and throw around non-player actors, instead of only affecting players. Non-players can be made immune to the thrusting with DONTTHRUST. (New from 4.10.0)

QF_SHAKEONLY: The earthquakes' damrad property will only be used to throw the player and other actors around, without actually harming them. (New from 4.10.0)

mulwavex/y/z: Only used with QF_WAVE. Specifies the number of waves per second the wave quake goes through while active. Default is 1. These are float values, so precision can be achieved.

falloff: Determines how far away the quake will start to reduce its amplitude based on distance. Takes the same arguments at tremrad in map units. Anything inside this will experience the full effect of the quake. Default is 0, which is no falloff.

highpoint: Only used with QF_SCALE<DOWN/UP>. Determines how far into the quake in tics for the quake to reach the peak of its shaking (or lack thereof if QF_MAX is included). Default is 0, or directly half way.

rollintensity: The camera roll is affected in a similar way to intensityX/Y/Z if specified. Unlike normal intensity, this is

not capped. If QF_WAVE is used, can also be negative to allow randomization. This feature does not rely upon the standard intensities and can be used separately (placing 0 in intensity and mulwave properties), but the flags affect it exactly the same.

rollwave: Similar to mulwavex/y/z, but for rolling.

Regular random quakes can stack with wave quakes, but to achieve this effect, one call to A_QuakeEx with QF_WAVE specified must happen with another call to the same function without QF_WAVE. This allows for cameras to shake along the wave, as the wave takes priority over where the jittering regular quakes positions. Wave quakes also will pause when the game is paused in any manner. Normally, regular quakes persist through an opened console for example, but wave quakes will always halt when a menu or console pause occurs. Similarly, rolling quakes follow the same behavior.

Contrarily to the Radius_Quake action special, the radii for damage and tremor are given directly in map units, not in "tiles" of 64x64 map units. This must be kept in mind when updating an actor definition from Radius_Quake to A_QuakeEx.

Examples

Cacodemon in this example die with quake around him:

Actor Caco4ZDoomWiki: Cacodemon replaces Cacodemon

```
{
Health 600
Scale 1.3
Translation "16:47=168:191", "112:127=208:223"
States
{
Death:
  HEAD G 12 A_QuakeEx( 2, 2, 2, 60, 0, 3000 ) //Light quake on 3000 map units around cacodemon death
  HEAD H 12 A_Scream
  HEAD IJ 12
  HEAD K 12 A_NoBlocking
  HEAD L -1 A_SetFloorClip
  Stop
}
}
```

A_SetBlend

A_SetBlend (color color1, double alpha, int tics [, color color2 [, double alpha2]])

Sets a palette blend effect to the calling player's screen. This function should only be used by player classes, weapons or custom inventory items. Colors are specified as an "rr gg bb" string or by using one of the color names specified in X11R6RGB lump.

Alpha defines opaqueness. An alpha of 1.0 is completely opaque, and 0.0 is completely invisible.

Over tics duration, color1 will steadily shift to color2. If color2 is not provided, it will instead decrease the intensity of the blending.

This is more or less the DECORATE equivalent of FadeRange.

Example

If you wanted to add a separate pain state for Doomguy getting zorched, and you used the "zorch" damagetype for those weapons, you could use this code to add a convincing red flash to the pain state associated with the zorch damagetype.

```
actor MyDoomPlayer : DoomPlayer
{
    States
    {
        Pain.Zorch:
            PLAY O 4 A_SetBlend("99 20 20", .5, 5)
            PLAY O 4 A_Pain
            Goto Spawn
    }
}
```

A_SkullPop

A_SkullPop (string className)

Will launch an actor of type className from the top of the calling actor in a vertical direction. Only effective when used on players (use A_SpawnItemEx for a generic actor equivalent). The player's viewpoint will immediately switch to that of the spawned actor (hence the name of the function).

Additionally, it prevents the player from using "resurrect" in the console to revive themselves after death.

This function is used in the "XDeath" states of Heretic's Corvus and the Fighter from Hexen, to separate the severed head from the main corpse. The player sees through the eyes of the head as it bounces to a stop.

If className is not given, it defaults to BloodySkull.

Examples

From the Fighter DECORATE definition:

XDeath:

```
PLAY O 5 A_PlayerScream
PLAY P 5 A_SkullPop
PLAY R 5 A_NoBlocking
PLAY STUV 5
PLAY W -1
Stop
```

A_SprayDecal

```
void A_SprayDecal (String name [, double dist [, vector3 offset [, vector3 direction [, bool useBloodColor [, color  
decalColor]]]])
```

Note: starting with 4.4.0, this function is no longer fully supported by DECORATE. If used from DECORATE, all parameters starting from offset should be omitted. Consider switching to ZScript to fully utilize this function.

Usage

Generates the specified decal on a nearby wall. The calling actor needs to be facing the wall in order to successfully generate the decal.

Parameters

name: the name of the decal to generate as defined in DECALDEF.

dist: the maximum distance in map units the calling actor can stand from the wall and still successfully generate the decal. Default is 172.0.

offset: (Need more info) Default is (0, 0, 0).

direction: (Need more info) Default is (0, 0, 0).

useBloodColor: if this is true, the sprayed decal is shaded to match the calling actor's blood color. Default is false.

decalColor: the desired color of the decal. Due to how decals work in the engine, the decal is only properly colorized if the graphic is grayscale. This parameter takes precedence over useBloodColor. Default is 0.

A_SpriteOffset

void A_SpriteOffset [(double ox [, double oy])]

Usage

Sets the calling actor's sprite offsets. The offsets are set relative to the sprite's built-in offsets.

Parameters

ox: the horizontal offset. Positive values shift the sprite to the right, while negative values shift it to the left. Default is 0.0.

oy: the vertical offset. Positive values shift the sprite downward, while negative values shift it upwards. Default is 0.0.

A_CheckBlock

State A_CheckBlock(StateLabel label, int flags = 0, int ptr = AAPTR_DEFAULT, double xofs = 0, double yofs = 0, double zofs = 0, double angle = 0)

Note: Jump functions perform differently inside of anonymous functions.

Usage

Checks if the given Actor pointer would be blocked at the position defined by the x/y/z and angle offsets. If it would be blocked, the caller jumps to the state specified by label. The CBF_DROPOFF flag will enable full movement checking instead of doing a basic position check. An important note is that the pointers from the CBF_SET* flags will still get set even when CBF_NOACTORS is used, but only if CBF_DROPOFF isn't set.

Parameters

label: The State to jump to if the position would be blocked.

flags: The following can be combined with a | symbol (bitwise or)

CBF_NOLINES - Ignore any lines that would block the Actor pointer.

CBF_SETTARGET - If an Actor would block the Actor pointer, set that Actor as the caller's target.

CBF_SETMASTER - Same as CBF_SETTARGET, but for master.

CBF_SETTRACER - Same as CBF_SETTARGET, but for tracer.

CBF_SETONPTR - Set the pointers on the Actor pointer instead of the caller. Requires any of the CBF_SET* flags to be set.

CBF_DROPOFF - Test to see if the Actor pointer would move over any terrain it can't traverse.

CBF_NOACTORS - Ignore any Actors that would block the Actor pointer.

CBF_ABSOLUTEPOS - Use x/y/zofs as an absolute map position. angle will be ignored.

CBF_ABSOLUTEANGLE - Use angle as an absolute direction instead of offsetting from the caller's current angle.

ptr: Which Actor pointer to use for the test. By default this is the caller.

xofs: How much to offset forward from the Actor pointer's position based on the caller's current angle.

yofs: How much to offset to the left from the Actor pointer's position based on the caller's current angle.

zofs: How much to offset upward from the Actor pointer's current position.

angle: How much to offset the caller's current angle. Positive values rotate counter-clockwise.

Return Value

Returns the State the caller wants to jump to. This is null if the position wasn't blocked. The return value is only relevant within anonymous or custom ZScript functions. Otherwise the jump will happen automatically.

Examples

This former human is very nervous; it runs around if anything disturbs it.

Actor NervousZombieman : Zombieman

```
{
    States
    {
        See:
        POSS AA 4 Fast A_Chase
        POSS A 0 A_CheckBlock("Nervous", CBF_SETTARGET, AAPTR_DEFAULT, radius + 1)
        POSS BB 4 Fast A_Chase
        POSS B 0 A_CheckBlock("Nervous", CBF_SETTARGET, AAPTR_DEFAULT, radius + 1)
        POSS CC 4 Fast A_Chase
        POSS C 0 A_CheckBlock("Nervous", CBF_SETTARGET, AAPTR_DEFAULT, radius + 1)
        POSS DD 4 Fast A_Chase
        POSS D 0 A_CheckBlock("Nervous", CBF_SETTARGET, AAPTR_DEFAULT, radius + 1)
    }
    Loop
    Missile:
```

POSS E 10 A_FaceTarget

POSS F 8 A_PosAttack

POSS E 8

Goto See

Pain:

POSS G 3

POSS G 3 A_Pain

Goto Nervous

Nervous:

POSS AABBBCCDD 2 Fast A_Wander

POSS A 0 A_Jump(64, "See")

Loop

Raise:

POSS K 5

POSS JIH 5

Goto See

}

}

A_CheckCeiling

```
state A_CheckCeiling (int offset)
state A_CheckCeiling (str "state")
```

Note: Jump functions perform differently inside of anonymous functions.

Usage

Jumps if actor is touching/submerged into the ceiling (calling actor's z + height).

Examples

This rocket does not explode when it hits the ceiling.

ACTOR Useless_Rocket: Rocket Replaces Rocket

```
{
  DeathSound "None"
  States
  {
    Spawn:
      MISL A 1 Bright
      Loop
    CancelMovement:
      MISL A 1 Bright
      Loop
    Death:
      MISL B 0 A_CheckCeiling("CancelMovement")
      MISL B 0 A_PlaySound("weapons/rocklx")
      MISL B 8 Bright A_Explode
      MISL C 6 Bright
      MISL D 4 Bright
      Stop
  }
}
```

A_CheckFloor

```
state A_CheckFloor (int offset)
state A_CheckFloor (str "state")
```

Note: Jump functions perform differently inside of anonymous functions.
Jumps if actor is standing on or submerged into the floor.

Examples

This rocket does not explode when it hits the ground.

ACTOR Useless_Rocket: Rocket Replaces Rocket

```
{
  DeathSound "None"
  States
  {
    Spawn:
      MISL A 1 Bright
      Loop
    CancelMovement:
      MISL A 1 Bright
      Loop
    Death:
      MISL B 0 A_CheckFloor("CancelMovement")
      MISL B 0 A_PlaySound("weapons/rocklx")
      MISL B 8 Bright A_Explode
      MISL C 6 Bright
      MISL D 4 Bright
      Stop
  }
}
```

A_CheckLOF

state A_CheckLOF (state jump [, int flags [, float range [, float minrange [, float angle [, float pitch [, float offsetheight [, float offsetwidth [, int ptr_target[, float offsetforward]]]]]]]]))

Note: Jump functions perform differently inside of anonymous functions.

Usage

Performs a would-be-hitscan/line of fire test to see if anything other than the target is encountered along the way. If the target would be hit, jump to the specified state.

Parameters

jump: Actor state jump, used if the target or another acceptable victim blocks the line of fire. Default is 0.

flags: The following flags can be combined by using the | character between the constant names:

CLOFF_NOAIM_VERT — Disables vertical aiming.

CLOFF_NOAIM_HORZ — Disables horizontal aiming.

CLOFF_AIM_VERT_NOOFFSET — Do not adjust aim (pitch) according to height offset. Will aim from whatever origin the other flags indicate (base/feet of actor, or calculated hitscan height).

CLOFF_FROMBASE — Move origin to actor's base

CLOFF_MUL_HEIGHT — Multiply height offset by actor's height, i.e. an offsetheight of 2 and a radius of 32 will offset to 64. This does nothing if offsetheight is 0. (If called by a player, takes into account whether they are crouching or not.)

CLOFF_MUL_WIDTH — Multiply width offset by actor's radius, i.e. an offsetwidth of 2 and a radius of 32 will offset to 64. This does nothing if offsetwidth is 0.

CLOFF_JUMPENEMY — Allows jumping if an enemy breaks the line of fire.

CLOFF_JUMPFRIEND — Same as the above, but the monsters must be friendly.

CLOFF_JUMPOBJECT — Anything that isn't a monster can cause the jump.

CLOFF_JUMPNONHOSTILE — Actors in the way that are not attacking the calling actor will trigger the jump.

CLOFF_SKIPENEMY — Always act as if there are no enemies in the way to block the shot.

CLOFF_SKIPFRIEND — Same as the above, only with friendlies.

CLOFF_SKIPOBJECT — Same as CLOFF_SKIPENEMY, but applies to non-monsters.

CLOFF_SKIPNONHOSTILE — Pretends monsters not attacking the calling actor between itself and the desired target are non-obstructive.

CLOFF_JUMP_ON_MISS — Hitting a wall, floor, or ceiling is a success if they are within range.

The following do not apply to the actor targeted by the line of fire check.

CLOFF_MUSTBESHOOTABLE — Ignore actors that are not shootable (checks SHOOTABLE and NONSHOOTABLE).

CLOFF_MUSTBEGHOST — Ignore actors that are not ghosts.

CLOFF_IGNOREGHOST — Ignore actors that are ghosts.

CLOFF_MUSTBESOLID — Ignore actors that are not solid.

Miscellaneous flags:

CLOFF_SKIPTARGET — Ignores the target actor breaking the ray -- none of the other filters apply to the target actor, only towards intercepting actors.

CLOFF_BEYONDTARGET — Requires CLOFF_SKIPTARGET. Trace past the targeted actor (only useful if checking for some of the additional jump qualifiers).

CLOFF_ALLOWNULL — Cast the ray even if target is null. When there is no target, caller aim (pitch, angle) is used regardless of aim-flags.

CLOFF_CHECKPARTIAL — Perform the check even if the target itself is actually out of range. (Useful if the actor is still interested in closer intercepting actors)

CLOFF_SETTARGET — Set the intercepting actor as the target of the calling actor.

CLOFF_SETMASTER — Set the intercepting actor as the master of the calling actor.

CLOFF_SETTRACER — Set the intercepting actor as the tracer of the calling actor.

Combo Flags:

CLOFF_SKIPOBSTACLES — Implies the SKIPENEMY, SKIPFRIEND, SKIPOBJECT, SKIPNONHOSTILE flags.

Partial overriding is allowed with combinations such as CLOFF_SKIPOBSTACLES|CLOFF_JUMPFRIEND.

CLOFF_NOAIM — Implies NOAIM_VERT and NOAIM_HORZ.

range: Range of the check. Default is 0.

minrange: Fail if the line of fire is blocked too near the shooting actor. Default is 0.

angle: Offsets the aim by this angle. Base aim depends on flags; on target by default. Default is 0.

pitch: Offset your aim by this angle (vertical). Base aim depends on flags; on target by default. Default is 0.

Note: To gain absolute pitch when the caller does not have a target, use -pitch [+ <expression>] since the function relies upon pitch when not aiming at anything.

offsetheight: Offset the line of fire's point of origin by this value. The base height depends on flags; hitscan emulation (around the middle) by default. When pitch is determined by aiming for target, aim is adjusted to correct for this offset. Default is 0.

offsetwidth: Offset the line of fire's point of origin by this value along the actor's side (horizontal). Positive values shift the trace origin to the right, negative to the left. Aim is not adjusted to correct for this value. Default is 0.

ptr_target: Pick an actor to aim for, AAPTR_DEFAULT picks TARGET (to ensure no actor is targeted, use AAPTR_NULL). Default is AAPTR_DEFAULT.

offsetforward: Offset the line of fire's point of origin going forward (positive values) or backwards (negative values). Aim is not adjusted for this value. This flag is affected by CLOFF_MUL_WIDTH the same way as offsetwidth. Default is 0.

Examples

This shotgun guy check distance to the target, and if too far, stops shoot:

Actor ScanningShotguy4ZDWiki: ShotgunGuy replaces ShotgunGuy

```
{
  Health 40
  States
  {
    Missile:
      SPOS E 5 A_FaceTarget
      SPOS E 5 A_CheckLOF( "Attack", CLOFF_SKIPOBSTACLES|CLOFF_IGNOREGHOST, 1500 )
      SPOS E 5
      Goto See
    Attack:
      SPOS F 10 Bright A_SPosAttackUseAtkSound
      SPOS E 5 A_CPosRefire
      Goto Missile
  }
}
```

A_CheckProximity

state A_CheckProximity (str jump, str classname, float distance [, int count [, int flags [, int ptr]]])

Note: Jump functions perform differently inside of anonymous functions.

Usage

Performs a check to see if an actor of type classname is within distance, and checks to see the population within distance is greater or equal to count. These conditions can be modified with flags and will count any actor, as long as it is the same classname. Can be performed based upon the actor's pointer as well. For use in ACS, see CheckProximity.

Parameters

jump: The state to jump to when the number of actor classname is found.

classname: The name of the actor to check for.

distance: Determines how far this function should search for the actors, similar to A_JumpIfCloser. Must be greater than 0.

count: The number of actors the function must find within distance in order to jump. Default is 1.

flags: Can be combined using the '|' separator. Default is 0.

CPXF_ANCESTOR: Search the inheritance of an actor, if it has any, for a partial match akin to CheckClass.

CPXF_NOZ: Disables Z height checking -- the function will only care about the actor being in range for X and Y coordinates, regardless of how high or low an actor is.

CPXF_CHECKSIGHT: Restricts actor counting to only those which can be seen.

Only one of the two following may be used due to mutual exclusion:

CPXF_COUNTDEAD: By default, the function only counts living actors. This causes the function to also accept dead ones. This only makes sense for monsters.

CPXF_DEADONLY: Inverses the function to only count dead monsters instead of living ones.

Only one of the two following may be used due to mutual exclusion:

CPXF_LESSOREQUAL: The function will jump if it finds the number of actors equivalent to number or less, instead of greater.

CPXF_EXACT: The function will only jump if the exact number is found defined by count.

Specifying one of the following flags makes the jump state optional to define, allowing a blank state "" to be used:

CPXF_SETTARGET: Sets the first actor matching classname as the calling actor's target.

CPXF_SETMASTER: Sets the first actor matching classname as the calling actor's master.

CPXF_SETTRACER: Sets the first actor matching classname as the calling actor's tracer.

The following flags rely on using one of the SET* flags above:

CPXF_FARTHEST: The actor gets the furthest classname actor within distance from the pointer's location.

CPXF_CLOSEST: The actor gets the closest classname actor within distance to the pointer's location.

CPXF_SETONPTR: The pointer exchange is set on the calling actor's pointer instead of itself.

ptr: The actor pointer to do the checking around. The jump is only performed by the original calling actor, however. Default is AAPTR_DEFAULT.

Examples

This Zombieman waits until a nearby archvile of his type (Nightmare) appears. Here the function is used for prevent resurrection by an incorrect type of archvile.

Actor NightmareZombie: Zombieman

```
{
  RenderStyle Subtract
  States
  {
```

```
Death:
    POSS H 5
    POSS I 5 A_Scream
    POSS J 0 A_DropItem( "Clip" )
    POSS J 0 A_UnsetShootable    // Don't "A_Fall" because archviles can react to this function too.
    POSS J 5 A_UnsetSolid
    POSS K 5
Death.Wait:
    POSS L 5
    POSS L 0 A_CheckProximity( "Death.HelpMe", "NightmareArch", 64 )
    Loop
Death.HelpMe:
    POSS L 1 CanRaise A_CheckProximity( "Death.Wait", "Archvile", 64 )
    POSS L 0 CanRaise A_CheckProximity( "Death.HelpMe", "NightmareArch", 64 )
    Goto Death.Wait
XDeath:
    POSS M 5
    POSS N 5 A_XScream
    POSS O 5 A_NoBlocking
    POSS PQRST 5
    POSS U 100000    // By default, archviles can only resurrect those who are in a state with a "-1" durability.
    Wait
}
}
```


A_CheckRange

state A_CheckRange (float distance, int offset [, bool 2d_check])

state A_CheckRange (float distance, str "state" [, bool 2d_check])

Note: Jump functions perform differently inside of anonymous functions.

Jumps if the calling actor is beyond distance range from all player pawns. If 2d_check is true, the z-coordinates are not taken into account when calculating the distance, making the distance check 2D-based instead of being 3D-based.

Default is false.

Examples

This lost soul's attack will redirect if it is further than 100 mapunits from its target.

Actor CleverLostSoul4ZDoomWiki: LostSoul replaces LostSoul

```
{
  States
  {
    Missile:
      SKUL C 10 Bright A_FaceTarget
      SKUL D 4 Bright A_SkullAttack
    CheckRangeState:
      SKUL CD 4 Bright A_CheckRange( 100, "AttackCorrection", True )
      loop
    AttackCorrection:
      SKUL D 4 Bright A_SkullAttack
      goto CheckRangeState
  }
}
```

A_CheckSight

```
state A_CheckSight (int offset)
state A_CheckSight (str "state")
```

Note: Jump functions perform differently inside of anonymous functions.

Jumps if there is no possible line of sight between the center of the calling actor and that of any player pawn.

This function does not account for whether or not a player is actually looking at the calling actor or not; as long as there is a possible line of sight (i.e. at least one player is in a position where they could see the calling actor if they faced the right direction), no jump will occur. And on the other hand, the jump might happen even though the actor is partially seen by a player.

Note that A_CheckSight also counts for actors being viewed through cameras and co-op spy. It does not check if the actor is being viewed through a camera texture.

Examples

The following actor fades out and disappears from the map once killed, but will not do so until out of sight of all players. Useful for open maps with a high body count, to reduce possible lag.

```
actor FadingZombie : Zombieman
{
    States
    {
        Death:
            POSS H 5
            POSS I 5 A_Scream
            POSS J 5 A_NoBlocking
            POSS K 5
            // intentional fallthrough
        DeathWait:
            POSS L 1 A_CheckSight("DeathFade")
            loop
        DeathFade:
            POSS L 1 A_FadeOut(0.02)
            loop
        XDeath:
            POSS M 5
            POSS N 5 A_XScream
            POSS O 5 A_NoBlocking
            POSS PQRST 5
            // intentional fallthrough
        XDeathWait:
            POSS U 1 A_CheckSight("XDeathFade")
            loop
        XDeathFade:
            POSS U 1 A_FadeOut(0.02)
            loop
        Raise:
            stop // not fair to have the monster revivable just because it's in LOS
    }
}
```

A_CheckSightOrRange

state A_CheckSightOrRange (float distance, int offset [, bool 2d_check])

state A_CheckSightOrRange (float distance, str "state" [, bool 2d_check])

Jumps if the calling actor is beyond distance range from all player pawns and there is no possible line of sight between the center of the calling actor and that of any player pawn. In other words, this function continues to the next state in the sequence if either the sight check or the range check were true; and jumps otherwise.

If 2d_check is true, the z-coordinates are not taken into account when calculating the distance, making the distance check 2D-based instead of being 3D-based. Default is false.

This function does not account for whether or not a player is actually looking at the calling actor or not; as long as there is a possible line of sight (i.e. at least one player is in a position where they could see the calling actor if they faced the right direction), no jump will occur. And on the other hand, the jump might happen even though the actor is partially seen by a player.

Note that like A_CheckSight, A_CheckSightOrRange also counts for actors being viewed through cameras and co-op spy. It does not, however, check if the actor is being viewed through a camera texture.

A_CheckSpecies

state A_CheckSpecies (state jump [, name species [, int ptr]])

Note: Jump functions perform differently inside of anonymous functions.

Usage

Checks for a specified species match with ptr and jumps to the specified state if it matches.

Parameters

jump: The state to jump to when the number of actor classname is found.

species: The name of the species to check. Default is none.

ptr: The actor pointer to check. Cannot be AAPTR_NULL.

A_Jump

```
state A_Jump (int chance, int offset, ...)
state A_Jump (int chance, str "state", ...)
```

Note: Jump functions perform differently inside of anonymous functions.

Usage

Randomly advances to different frame. The chance value can range between 0 and 256. A chance of 0 will never jump, while a chance of 256 will always jump. If the jump does not happen, then the frame or instruction immediately following the A_Jump will be used as if A_Jump had not been present. If more than one offset or state values are present, A_Jump will choose one of them at random.

Note that offset specifies the number of frames to skip over, and not the number of lines. Instructions like goto, loop and stop cannot be jumped to directly for this reason. (If you need the A_Jump command to skip directly to an instruction, see the last example below for a way to do this.)

Jumps made with A_Jump are virtual, i.e. if the execution pointer is currently in the state of a base class of an actor, it will jump to the redefined state (if exists) of the derived actor. This is in contrast to the goto keyword which is static and execution does not leave the base class.

Note: Because of a limitation in the decorate code parser, you cannot use A_Jump on a single line if it has more than one state while using offset to specify the state to jump to. You will either need to split the states onto multiple lines and add an A_Jump to each line individually, or reference an actual state label like "See", "Melee", etc to make it work properly.

Examples

The following examples are the old method of jumping without states. They are still usable but it is strongly recommended that you use state labels for convenience.

```
POSS A 0 A_Jump (256, 2)
POSS A 5
POSS A 5 // <- Jumps to this one (always)
POSS A 5
POSS A 0 A_Jump (127, 4)
POSS ABC 5
POSS D 5 // <- Jumps to this one (%50 chance)
POSS E 3
POSS A 0 A_Jump (5, 1)
goto Melee // <- This line is skipped because it does not define a frame
POSS A 5 // <- Jumps to this one (Very little chance)
POSS B 3
POSS A 0 A_Jump (127, 2, 3, 6)
POSS A 5 // <-- Has a 50% chance of dropping through to here (no jump), otherwise...
goto Death
POSS B 6 // <-- Jumps here...
goto See
POSS BCD 5 // <-- Or here...
goto Death
POSS E -1 // <-- Or here, with equal probability.
```

These examples use state labels. As you can see from both methods, state labels are a much simpler way of doing jumps.

```
POSS A 0 A_Jump (127, "Pain")
```

POSS ABC 3

Stop

Pain: // <-- 50% chance to jump to this state

POSS G 3

POSS G 3 A_Pain // Play pain sound

loop

POSS A 0 A_Jump (256, "Boogie", "Gasp", "Death") // <-- Always jumps to either...

Stop

Boogie: // <-- This state...

POSS ABCD 4

loop

Gasp: // <-- This state...

POSS E 15

Goto See

Death: // <-- Or this state at an equal chance.

POSS H 5

POSS I 5 A_Scream

POSS J 5 A_NoBlocking

POSS K 5

POSS L -1

stop

Here is a real example of where jumping to different states would be useful. This is a custom made DECORATE version of Korax's attack pattern using state jumps.

Missile:

KORX E 8 Bright A_FaceTarget

KORX E 0 A_PlaySound("KoraxAttack")

TNT1 A 0 A_Jump(256, "Wraith", "DemonFX1", "DemonFX2", "FireDemon", "Centaur", "Serpent")

Goto See

Serpent:

KORX A 0 A_SpawnProjectile ("SerpentFX", 55, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("SerpentFX", 55, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("SerpentFX", 82, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("SerpentFX", 82, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("SerpentFX", 110, -35, 0, 4, 0)

KORX F 8 A_SpawnProjectile ("SerpentFX", 110, 35, 0, 4, 0)

KORX E 5 Bright

Goto See

Wraith:

KORX A 0 A_SpawnProjectile ("WraithFX1", 55, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("WraithFX1", 55, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("WraithFX1", 82, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("WraithFX1", 82, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("WraithFX1", 110, -35, 0, 4, 0)

KORX F 8 A_SpawnProjectile ("WraithFX1", 110, 35, 0, 4, 0)

KORX E 5 Bright

Goto See

DemonFX1:

KORX A 0 A_SpawnProjectile ("Demon1FX1", 55, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("Demon1FX1", 55, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("Demon1FX1", 82, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("Demon1FX1", 82, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("Demon1FX1", 110, -35, 0, 4, 0)

KORX F 8 A_SpawnProjectile ("Demon1FX1", 110, 35, 0, 4, 0)

KORX E 5 Bright

Goto See

DemonFX2:

KORX A 0 A_SpawnProjectile ("Demon2FX1", 55, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("Demon2FX1", 55, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("Demon2FX1", 82, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("Demon2FX1", 82, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("Demon2FX1", 110, -35, 0, 4, 0)

KORX F 8 A_SpawnProjectile ("Demon2FX1", 110, 35, 0, 4, 0)

KORX E 5 Bright

Goto See

FireDemon:

KORX A 0 A_SpawnProjectile ("FireDemonMissile", 55, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("FireDemonMissile", 55, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("FireDemonMissile", 82, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("FireDemonMissile", 82, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("FireDemonMissile", 110, -35, 0, 4, 0)

KORX F 8 A_SpawnProjectile ("FireDemonMissile", 110, 35, 0, 4, 0)

KORX E 5 Bright

Goto See

Centaur:

KORX A 0 A_SpawnProjectile ("CentaurFX", 55, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("CentaurFX", 55, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("CentaurFX", 82, -52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("CentaurFX", 82, 52, 0, 4, 0)

KORX A 0 A_SpawnProjectile ("CentaurFX", 110, -35, 0, 4, 0)

KORX F 8 A_SpawnProjectile ("CentaurFX", 110, 35, 0, 4, 0)

KORX E 5 Bright

Goto See

A_JumpIf

state A_JumpIf (expression, int offset)

state A_JumpIf (expression, str "state")

Note: Jump functions perform differently inside of anonymous functions.

Jumps offset amount of states forward, or to the state label state, if expression evaluates to true. See DECORATE expressions for more information on expressions.

Examples

Example of a projectile that “fizzles out” if it enters water:

```
actor FizzleBall : DoomImpBall
{
    states
    {
    Spawn:
        BAL1 AB 4 bright
        BAL1 A 0 A_JumpIf(waterlevel == 2, "DeepWater")
        loop
    DeepWater:
        BAL1 A 0 A_PlaySound("Fizzle")
        FIZZ ABC 6
        stop
    }
}
```


A_JumpIfArmorType

state A_JumpIfArmorType (string "armortype", str "state"[, int minimum])

Note: Jump functions perform differently inside of anonymous functions.

Checks whether the equipped armor in the actor's inventory is of armortype. If so, the jump is performed. Optionally, a minimum value can be given, the jump will then be performed only if the armor is of that type and with at least that amount of points.

Examples

This custom armor bonus can only be picked up if the player has the blue armor equipped.

ACTOR CustomArmorBonus1 : CustomInventory

```
{
  Inventory.PickupMessage "$GOTARMBONUS"
  States
  {
    Spawn:
      BON2 ABCDCB 6
    Loop
  }
  Pickup:
    TNT1 A 0 A_JumpIfArmorType("BlueArmor", "GiveArmorBonus")
    Fail
  GiveArmorBonus:
    TNT1 A 0 A_GiveInventory("ArmorBonus", 1)
    Stop
}
```

This armor shard item can only be picked up if the player has the green armor equipped and less than 100 points of armor at the same time.

ACTOR ArmorShard : CustomInventory

```
{
  Inventory.PickupMessage "Picked up an armor shard."
  States
  {
    Spawn:
      BON2 A 6
      BON2 D 6 Bright
    Loop
  }
  Pickup:
    TNT1 A 0 A_JumpIfArmorType("GreenArmor", "NoPickup", 100)
    TNT1 A 0 A_JumpIfArmorType("GreenArmor", "GiveArmorBonus")
    Fail
  NoPickup:
    TNT1 A 0
    Fail
  GiveArmorBonus:
    TNT1 A 0 A_GiveInventory("ArmorBonus", 1)
    Stop
}
```

A_JumpIfCloser

state A_JumpIfCloser (float distance, int offset [, bool noz])
state A_JumpIfCloser (float distance, str "state" [, bool noz])

Note: Jump functions perform differently inside of anonymous functions.

Usage

Jumps offset amount of states forward, or to the state label state, if the distance to the target is lower than the given value. For weapons and inventory, jumps if the target in the player's crosshair is closer than the given value.

NOTE - This function does not take into account the radius of either actor, so it's possible that either one wide actor or both being wide can prevent the jump from ever occurring. It's important that a blocking actor takes their own radius into account through means such as A_JumpIfCloser(radius + distance, "label") to help avoid such occurrences.

Parameters

distance - The distance an actor's target must be in order to jump. Distance is in units, similar to to the radius of A_Explode.

offset/state - The offset number of frames to jump forward, or the state label to jump to.

noz - If true, the function disables vertical distance checking. The default is false, which includes comparing the distance between how high the calling actor is to the target.

Examples

This makes the imps in Doom transform into archviles at random, but the transformation is guaranteed if you're close to them when they first see you.

ACTOR AngryImp : DoomImp replaces DoomImp

```
{
  States
  {
    See:
      TROO A 0 A_JumpIfCloser(196, "Transform")
      TROO AABBCDD 3 A_Chase
    Loop
  Transform:
      TROO A 0 A_SpawnItemEx("Archvile", 0, 0, 0, 0, 0, 0, angle)
    Stop
  }
}
```

And this makes barrels self-aware and makes them go boom when you're in range:

ACTOR SmartBarrel : ExplosiveBarrel replaces ExplosiveBarrel

```
{
  +LOOKALLAROUND
  +AMBUSH
  +QUICKTORETALIATE
  States
  {
    Spawn:
      TNT1 A 0
      BAR1 AB 6 A_LookEx(LOF_NOSOUNDHECK)
    See:
      BAR1 AB 6 A_JumpIfCloser(127, "Death")
      Goto Spawn
  }
}
```


A_JumpIfHealthLower

state A_JumpIfHealthLower (int health, int offset [, int pointer])

state A_JumpIfHealthLower (int health, str "state" [, int pointer])

Note: Jump functions perform differently inside of anonymous functions.

Usage

Jumps offset amount of states forward, or to the state label state, if the health of the calling actor (or calling actor's pointer) is lower than the given value.

The default value for pointer is AAPTR_DEFAULT, which corresponds to the caller itself.

Examples

This script would cause a boss monster to switch its mode of attack when it reaches 5000 health

States

{

Missile:

POSS E 2 A_FaceTarget

POSS E 2 A_JumpIfHealthLower(5000, "LowHealthAttack")

POSS F 2 A_PosAttack

Goto See

LowHealthAttack:

POSS E 2 A_CPosAttack

Goto See

}

A_JumpIfHigherOrLower

state A_JumpIfHigherOrLower (str "high", str "low" [, float offsethigh [, float offsetlow [, bool includeHeight [, pointer ptr]]]])

Note: Jump functions perform differently inside of anonymous functions.

Usage

Jumps to either the high state or the low state if the actor pointer ptr is higher or lower than the calling actor's self based upon z-height. If the pointed actor is higher than the calling actor, adds (or subtracts) offsethigh to its check, and vice versa if the pointer is lower. includeHeight is important to consider as this can radically change the behavior of the function.

Either state jump can be disabled by leaving the space between the quotation marks empty ("" without parenthesis and no spaces between), but at least one state jump must be enabled to have any effect.

Parameters

high: State jump the calling actor goes to if the pointer is higher. Set to "" to disable.

low: State jump the calling actor goes to if the pointer is lower. Set to "" to disable.

offsethigh: Default is 0. Offsets the calling actor's height or z (see includeHeight) for the pointer being above the caller by this amount. Positive values require the pointer to be this much higher over the calling actor's height in order to jump to the high state. Negative values allow the pointer to be lower by this much and successfully trigger the high state jump.

offsetlow: Default is 0. Offsets the calling actor's z check for the pointer being below the caller by this amount. Negative values will require the pointer to be lower. Positive values will allow the pointer to rise above the callers z by this much, depending on includeHeight.

includeHeight: Defaults to true. When enabled, this works two-fold.

The function will factor in the caller's height if the pointer is above the caller.

Inversely, it will factor in the pointer's height if the pointer is below the caller.

If set to false, the function will simply compare z positions without adding height. offsethigh and offsetlow are still added in, however.

ptr: Defaults to AAPTR_TARGET. Checks the following actor pointer. Can take any pointer except AAPTR_DEFAULT and AAPTR_NULL; the function was built to check another actor, not itself, and will do nothing in that regard.

A_JumpIfInTargetInventory

```
state A_JumpIfInTargetInventory (str "item", int count, int offset[, pointer forward])
state A_JumpIfInTargetInventory (str "item", int count, str "state"[, pointer forward])
```

Note: Jump functions perform differently inside of anonymous functions.

Usage

Jumps if the target has the specified actor type item and amount count in its inventory.

If an actor pointer is given, the test will be forwarded to the actor in the target's corresponding pointer.

Examples

This Imp runs away if the player has a BFG9000 or a Plasma Rifle.

```
actor WiseImp : DoomImp
{
    states
    {
        Spawn:
            TROO A 0 A_ChangeFlag("FRIGHTENED",0) // Clear the FRIGHTENED flag if the Imp loses its target.
            TROO AB 10 A_Look
            Goto Spawn+1
        See:
            TROO AABBBCCDD 3 A_Chase
            TROO A 0 A_JumpIfInTargetInventory("PlasmaRifle", 1, "Ohcrap")
            TROO A 0 A_JumpIfInTargetInventory("BFG9000", 1, "Ohcrap")
        loop
        Ohcrap:
            TROO A 0 A_Pain // Imp roars in fear
            TROO A 3 A_ChangeFlag("FRIGHTENED",1)
            TROO AABBBCCDD 2 A_Chase
            Goto Ohcrap+2
    }
}
```

A_JumpIfInTargetLOS

```
state A_JumpIfInTargetLOS (int offset[, float fov[, int flags[, float dist_max[, float dist_close]]]])  
state A_JumpIfInTargetLOS (str "state"[, float fov[, int flags[, float dist_max[, float dist_close]]]])
```

Note: Jump functions perform differently inside of anonymous functions.

Usage

Jumps if the calling actor is in the field of view of its target. The flags parameter is used to determine the target of the calling actor's target. In the rest of the description, "target" will refer to either:

- the target's master, if the JLOSF_CHECKMASTER flag is set;
- the target's tracer, if the actor is a missile with the SEEKERMISSILE flag and JLOSF_PROJECTILE is set;
- the target's target, in all other cases. This corresponds to a monster's current enemy or a projectile's shooter.

If the calling actor has a valid target (as explained above) and can currently be seen by it, then a jump is performed. If fov is given, it is an angle representing the size (in degrees) of the sight cone. Actors outside of this cone (the center of which is the actor's current facing direction) are not considered to be "visible". If omitted, no sight cone is used and instead this function jumps if the view between the target and the caller is unobstructed (the target can see the caller, regardless of facing direction, given there are no walls between the two).

Parameters

state is label of the state to which the jump must be made.

offset corresponds to the number of states that must be skipped to make the jump.

fov corresponds to the field of vision used by the calling actor. The default value of 0 gives an all-around field of vision.

flags: The following flags can be combined by using the | character between the constant names:

JLOSF_PROJECTILE — if set, the actor checks sight with the target it is seeking (the tracer field) instead of the normal target field (that, for projectiles, corresponds to the actor that fired it). If the actor is a projectile and does not have the SEEKERMISSILE flag set, the target will be considered NULL, otherwise, if it is not a missile, the flag is ignored.

JLOSF_NOSIGHT — disables the sight check.

JLOSF_CLOSENOFOV — disables the FOV check if the target is within dist_close distance.

JLOSF_CLOSENOSIGHT — disables the sight check if the target is within dist_close distance.

JLOSF_CLOSENOJUMP — does not perform a jump if the target is within dist_close distance.

JLOSF_DEADNOJUMP — does not perform a jump if the target is dead.

JLOSF_CHECKMASTER — use the master field; overrides all other flags that specify the target.

dist_max: the maximum distance between the calling actor and the target, no jumps are performed beyond it. The default value of 0 means there are no distance checks.

dist_close: the function of this parameter depends on the flags used; it has otherwise no effect. The default value is 0.

Examples

This revenant will dodge its target's attacks:

See:

```
TNT1 A 0 A_JumpIfInTargetLOS(1, 30, JLOSF_DEADNOJUMP, 500) // if close and we're under their  
crosshair, jump!
```

```
Goto See+3 // skip the evasion code.
```

```
TNT1 A 0 A_FaceTarget
```

```
SKEL A 4 A_ChangeVelocity(6, 0, 0, CFV_RELATIVE)
```

```
SKEL AABBCDDDEEFF 2 A_Chase
```

```
Loop
```

A_JumpIfInventory

```
state A_JumpIfInventory (string "inventorytype", int amount, int offset[, int owner])  
state A_JumpIfInventory (string "inventorytype", int amount, str "state"[, int owner])
```

Note: Jump functions perform differently inside of anonymous functions.

Usage

Checks the amount of inventorytype items in the actor's inventory. If there are at least amount, the jump is performed. The offset parameter specifies how many frames ahead to jump.

If amount is greater than the maximum amount, the jump will be performed.

Note that if amount is 0, the jump is only performed if the actor is carrying the maximum number of that item. This is useful for checking whether the player has the maximum amount of ammo for a particular weapon, regardless of whether or not he's found a backpack.

If one of the actor pointers is specified, it will check their inventory instead of the calling actor's own. By default, AAPTR_DEFAULT refers to itself and is used when left unspecified or blank.

Examples

Shotgun reload

Here's a shotgun with a Counterstrike (or whatever the hell you want to call it)-style reloading system.

```
actor DukeShotgun: Weapon 22009  
{  
    Weapon.AmmoType "InsertedShell"  
    Weapon.AmmoType2 "Shell"  
    Weapon.AmmoGive 0  
    Weapon.AmmoGive2 8  
    Weapon.AmmoUse 1  
  
    Scale 0.625  
  
    AttackSound "duke/shotgun"  
    Inventory.PickupSound "misc/w_pkup"  
  
    Inventory.PickupMessage "Shotgun!"  
    Obituary "%o was wiped out by %k."  
  
    States  
    {  
        Select:  
            SHTG A 1 A_Raise  
            Loop  
  
        Deselect:  
            DSHTG A 1 A_Lower  
            Loop  
  
        Ready:  
            SHTG A 1 A_WeaponReady  
            loop
```


Fire:

```
SHTG A 2 A_GunFlash
SHTG F 2 A_FireBullets(2.0, 1.0, 8, 4)
SHTG EDCBA 3
SHTG G 5 A_FireProjectile("ShotgunShell",90,0)
SHTG HIJHG 4
SHTG A 0 A_JumpIfNoAmmo("ReloadStart_Fire")
SHTG A 0 A_Refire
Goto Ready
```

Flash:

```
SHTF AB 2 bright
Stop
```

AltFire:

```
goto ReloadStart
```

ReloadStart:

```
SHTR A 0 A_JumpIfInventory("Shell", 1, 1)
goto NoReload

SHTR A 0 A_JumpIfInventory("InsertedShell", 0, "NoReload")
SHTR GH 3
goto ReloadLoop
```

ReloadLoop:

```
TNT1 A 0 A_TakeInventory("Shell", 1)
SHTR I 3 A_GiveInventory("InsertedShell", 1)
SHTR J 3 A_PlaySound("weapons/sshot1")
TNT1 A 0 A_JumpIfInventory("InsertedShell", 0, "ReloadEnd")
TNT1 A 0 A_JumpIfInventory("Shell", 1, "ReloadLoop")
Goto ReloadEnd
```

ReloadEnd:

```
SHT3 HG 3
Goto Ready
```

NoReload:

```
TNT1 A 0
Goto Ready
```

Spawn:

```
SHTD A -1
Stop
```

```
}
}
```

Armor Addon

This is an armor shard that is supposed to increase your armor if you are wearing one and have less than 150 armor. But it does not work: if you pick a GreenArmor, then try to pick this item up, you will not succeed. The GreenArmor set the maximum amount for BasicArmor to 100, so the first jump is executed despite falling 50 points short of the 150 value requested.

ACTOR ArmorShard : CustomInventory

```
{
```

```

States
{
Spawn:
    BON2 A -1
    Stop
Pickup:
    TNT1 A 0 A_JumpIfInventory("BasicArmor", 150, "NoNeed")
    TNT1 A 0 A_JumpIfInventory("BasicArmor", 1, "CanAdd")
NoNeed:
    TNT1 A 0
    Stop
CanAdd:
    TNT1 A 0 A_GiveInventory("ArmorBonus")
    Stop
}
}

```

And here is the fixed version: it involves using a workaround by putting the maximum of 150 in a variant armor bonus.

```

ACTOR ShardBonus : ArmorBonus
{
    -INVENTORY.ALWAYSPICKUP
    Armor.MaxSaveAmount 150
}
ACTOR ArmorShard : CustomInventory
{
    States
    {
    Spawn:
        BON2 A -1
        Stop
    Pickup:
        TNT1 A 0 A_JumpIfInventory("BasicArmor", 1, "CanAdd")
        Stop
    CanAdd:
        TNT1 A 0 A_GiveInventory("ShardBonus")
        Stop
    }
}

```

A test for at least 80 rockets would likewise have to find a similar workaround, probably by checking first for the presence of a Backpack. This is not a fool-proof solution, however, as the maximum ammo capacity can be modified through scripting or by picking up another type of BackpackItem.

A_JumpIfMasterCloser

state A_JumpIfMasterCloser (int distance, int offset [, bool noz])

state A_JumpIfMasterCloser (int distance, str "state" [, bool noz])

Note: Jump functions perform differently inside of anonymous functions.

Usage

Jumps the given offset forward or to the given state if the distance to the master is lower than the given value.

Note: This function does not take into account the radius of either actor, so it's possible that either one wide actor or both being wide can prevent the jump from ever occurring. It's important that a blocking actor takes their own radius into account through means such as A_JumpIfMasterCloser(radius + distance, "label") to help avoid such occurrences.

Parameters

distance - The distance an actor's master must be in order to jump. Distance is in units, similar to to the radius of A_Explode.

offset/state - The offset number of frames to jump forward, or the state label to jump to.

noz - If true, the function disables vertical distance checking. The default is false, which includes comparing the distance between how high the calling actor is to the master.

Examples

If this ZombieMan's master is away from him, the zombieman will forget it.

actor TraitorZombieman : ZombieMan

```
{
    States
    {
        See:
        POSS A 0 A_JumpIfMasterCloser(1024, 2)
        POSS A 0 A_RearrangePointers(AAPTR_DEFAULT, AAPTR_NULL, AAPTR_DEFAULT)
        POSS AABBCDD 4 A_Chase
        Loop
    }
}
```

A_JumpIfNoAmmo

```
state A_JumpIfNoAmmo (int offset)
state A_JumpIfNoAmmo (str "state")
```

Note: Jump functions perform differently inside of anonymous functions.

Usage

This function can only be used with weapons. It jumps if the carrying player doesn't have enough ammunition for one attack of the currently used attack mode (defaulting to primary fire if called outside of a fire state).

This function also checks for the Infinite Ammo DMFlag or the infinite ammo powerup. If either are present, this will never jump.

Offset is the number of frames to jump ahead when there is no ammo. State is what state to jump to when you're out of ammo.

Examples

States

```
{
```

Fire:

```
CISG A 0 A_JumpIfNoAmmo("Reload")
CISG B 1 BRIGHT A_GunFlash
TNT1 A 0 A_FireProjectile("PistolShellcasingSpawn",0,0,-6,2)
CISG B 1 A_FireBullets(3,2,-1,9.2,"BULLETHIT",1,0)
CISG CDE 2
Goto Ready
```

A_JumpIfTargetInLOS

```
state A_JumpIfTargetInLOS (int offset [, float fov [, int flags [, float dist_max [, float dist_close]]]])  
state A_JumpIfTargetInLOS (str "state" [, float fov [, int flags [, float dist_max [, float dist_close]]]])
```

Note: Jump functions perform differently inside of anonymous functions.

This can be used with monsters, weapons, projectiles and inventory items.

For weapons and inventory, jumps if the player has a suitable line target under their crosshair. With weapons and inventory, FOV is meaningless and can be omitted. "Suitable" is defined as:

The targeted actor has the SOLID flag.

The targeted actor has the SHOOTABLE flag.

The targeted actor does not have the NOBLOCKMAP flag.

For monsters and seeking projectiles, the flags parameter is used to determine the actor that the calling actor looks for. In the rest of the description, "target" will refer to either:

The calling actor's master, if the actor is a monster and the JLOSF_CHECKMASTER flag is set;

The calling actor's tracer, if the actor is a missile with the SEEKERMISSILE flag and JLOSF_PROJECTILE is set. It will also refer to tracer if the JLOSF_CHECKTRACER is set, regardless if the actor is a missile or not;

The calling actor's target, in all other cases. This corresponds to a monster's enemy or a projectile's shooter. (It is counter-intuitive, but a projectile's target field does not store the projectile's actual target.)

Usage

Jumps if the calling actor has a valid target (as explained above) and can currently see it. If FOV is given, it is an angle representing the size (in degrees) of the actor's sight cone. Actors outside of this cone (the center of which is the calling actor's current facing direction) are not considered to be "visible". If omitted, no sight cone is used and instead this function jumps if the view between the caller and the target is unobstructed (the caller can see the target, regardless of facing direction, given there are no walls between the two).

Parameters

state is label of the state to which the jump must be made.

offset corresponds to the number of states that must be skipped to make the jump.

fov corresponds to the field of vision used by the calling actor. The default value of 0 gives an all-around field of vision.

flags: The following flags can be combined by using the | character between the constant names:

JLOSF_PROJECTILE — Seeking projectile: If set, the actor checks sight with the target it is seeking (tracer field) instead of the normal target field (which, for projectiles, actually corresponds to the actor that fired it). If the actor is not a projectile or does not have the SEEKERMISSILE flag set, this flag is ignored.

JLOSF_NOSIGHT — No sight check: Disables the sight check.

JLOSF_CLOSENOFOV — No FOV check if close: Disable the FOV check if the target is within dist_close distance.

JLOSF_CLOSENOSIGHT — No sight check if close: Disable the sight check if the target is within dist_close distance.

JLOSF_CLOSENOJUMP — No jump if close: Does not perform a jump if the target is within dist_close distance.

JLOSF_DEADNOJUMP — No jump if dead: Does not perform a jump if the target is dead.

JLOSF_CHECKMASTER — Check master: Use the master field instead of the target or tracer field.

JLOSF_TARGETLOS — Check target's line of sight.

JLOSF_FLIPFOV — Check FOV for the actor that does not make a sight check.

JLOSF_ALLYNOJUMP — Jump only if the actors are not allied to each other. Unfriendly monsters have no allies.

JLOSF_COMBATANTONLY — Jump only if the target is a monster or a player. (Particularly useful when working with a player's target, as it could be any actor.)

JLOSF_NOAUTOAIM — Jump only if the target is under the crosshair, taking both horizontal and vertical aims into account. This is to cater for weapons that require manual aiming by nature such as railguns, or ones that have autoaim disabled in them with the use of the WEAPON.NOAUTOAIM flag.

JLOSF_CHECKTRACER — Check line of sight with the calling actor's tracer instead of target. Unlike JLOSF_PROJECTILE, this flag works on non-missile actors.

The combination of the flags JLOSF_TARGETLOS and JLOSF_FLIPFOV duplicates the behavior of A_JumpIfInTargetLOS (target line is checked, caller fov is checked).

dist_max: The maximum distance between the calling actor and the target, no jumps are performed beyond it. The default value of 0 means there are no distance checks.

dist_close: The function of this parameter depends on the flags used; it has otherwise no effect. The default value is 0.

Examples

This is a modified railgun that will not shoot unless it will hit something.

ACTOR ScannerRailgun : Railgun

```
{
+WEAPON.NOAUTOFIRE // Disables Automatic fire.
States
{
Fire:
    RLGG E 1 A_JumpIfTargetInLOS(1, 0, JLOSF_NOAUTOAIM) // Is there something that is being aimed at?
    Goto Ready // Cannot fire, go back to ready
    RLGG E 12 A_FireRailgun
    RLGG F 6 A_CheckForReload(4, "Reloaded")
    RLGG GHIJK 6
    RLGG L 6 A_ResetReloadCounter
}
}
```

A_JumpIfTargetInsideMeleeRange

state A_JumpIfTargetInsideMeleeRange(int offset)

state A_JumpIfTargetInsideMeleeRange(str state)

Note: Jump functions perform differently inside of anonymous functions.

Usage

Jumps the number of frames (offset) forward, or to the specified state when the target of the calling actor is within melee range of the caller.

A_JumpIfTargetOutsideMeleeRange

state A_JumpIfTargetOutsideMeleeRange(int offset)

state A_JumpIfTargetOutsideMeleeRange(str state)

Note: Jump functions perform differently inside of anonymous functions.

Usage

Jumps the number of frames (offset) forward, or to the specified state when the target of the calling actor is beyond melee range of the caller.

A_JumpIfTracerCloser

state A_JumpIfTracerCloser (float distance, int offset [, bool noz])

state A_JumpIfTracerCloser (float distance, str "state" [, bool noz])

Note: Jump functions perform differently inside of anonymous functions.

Usage

Jumps offset amount of states forward, or to the state label state, if the distance to the tracer is lower than the given value.

Note: This function does not take into account the radius of either actor, so it's possible that either one wide actor or both being wide can prevent the jump from ever occurring. It's important that a blocking actor takes their own radius into account through means such as A_JumpIfTracerCloser(radius + distance, "label") to help avoid such occurrences.

Parameters

distance - The distance an actor's tracer must be in order to jump. Distance is in units, similar to to the radius of A_Explode.

offset/state - The offset number of frames to jump forward, or the state label to jump to.

noz - If true, the function disables vertical distance checking. The default is false, which includes comparing the distance between how high the calling actor is to the tracer.

A SetChaseThreshold

A_SetChaseThreshold (int threshold [, bool def [, int ptr]])

Usage

Changes the calling actor (pointer)'s Threshold or DefThreshold to threshold, depending on if def is true or not.

Parameters

threshold: The actor's new threshold.

def: If true, sets the default threshold instead.

ptr: The pointer to set it upon.

A SetFloatSpeed

A_SetFloatSpeed (float speed [, int pointer])

Usage

Changes the calling actor's pointer's FloatSpeed property, which is the speed of an actor's ascending and descending speed while floating and chasing an enemy.

Parameters

speed: The value to set the actor's float speed to.

pointer: The actor to change its speed. This is an actor pointer. Default is AAPTR_DEFAULT, which corresponds to the calling actor.

A SetPainThreshold

A_SetPainThreshold (int threshold [, int ptr])

Usage

Changes the calling actor (pointer)'s PainThreshold to threshold. The higher the pain threshold is, the more damage a shootable actor must take in order to have a chance to enter its pain states, depending on PainChance.

Parameters

threshold: The actor's new pain threshold.

ptr: The pointer to set it upon.

A SetVisibleRotation

A_SetVisibleRotation(float anglestart[, float angleend[, float pitchstart[, float pitchend[, int flags[, int ptr]]]]]))

Usage

Sets the visibility of the sprite based on what angle and pitch the actor is viewed from. Note that setting the angle parameters without explicitly specifying the pitch range does not work either (the actor will be completely invisible) (bug?).

Parameters

anglestart - The start of the angle range to see. Positive values turn it to the right, negative to the left.

angleend - The end of the angle range.

pitchstart - The start of the pitch range to see. Positive values means viewing it from lower to see it, negative means higher.

pitchend - The end of the pitch range.

flags - Can be combined with | symbol:

VRF_NOANGLESTART - The starting angle range will not be modified regardless of number input.

VRF_NOANGLEEND - The ending angle range will not be modified.

VRF_NOPITCHSTART - The starting pitch range will not be modified.

VRF_NOPITCHEND - The ending pitch range will not be modified.

VRF_NOANGLE - Implies VRF_NOANGLESTART|VRF_NOANGLEEND.

VRF_NOPITCH - Implies VRF_NOPITCHSTART|VRF_NOPITCHEND.

ptr - The Actor pointer to modify.

A_ActiveAndUnblock

(no parameters)

Plays the active sound and then calls A_NoBlocking.

Examples

The following two DECORATE code blocks are equivalent.

Death:

```
MONS HI 5
MONS J 0 A_ActiveSound
MONS J 5 A_NoBlocking
MONS K -1
stop
```

Death:

```
MONS HI 5
MONS J 5 A_ActiveAndUnblock
MONS K -1
stop
```

A_ChangeCountFlags

```
void A_ChangeCountFlags [(int kill [, int item [, int secret]])]
```

Usage

Sets whether or not the calling actor should count towards the map's statistics of kills, items and secrets. If any of these parameters is set to 0, the actor is not counted towards the statistic represented by the parameter. If set to FLAG_NO_CHANGE or -1, no change takes place. If set to any other value, the actor is counted.

Parameters

kill: whether or not the actor should count towards the map's kills. The actor needs to be non-friendly and alive for any changes to take effect. Default is FLAG_NO_CHANGE.

item: whether or not the actor should count towards the map's items. Default is FLAG_NO_CHANGE.

secret: whether or not the actor should count towards the map's secrets. Default is FLAG_NO_CHANGE.

A_ChangeModel (New from 4.10.0)

```
void A_ChangeModel (name modeldef [, int modelindex [, string modelpath [, name model [, int skinindex [, string skinpath [, name skin [, int flags [, int generatorindex [, int animationindex [, string animationpath [, name animation]]]]]]]]]]))
```

Usage

This can change the modeldef, model and skins of an actor.

You can add models to an actor to render by using generatorindex to tell your new model index to copy the behavior of another one. This does not actually generate any modeldef data, but does instruct the index to copy the behavior of the index you put for generatorindex. As a result, you may only copy the behavior of a model index already defined in your actor's current modeldef. To remove a model or skin, just pass "" to modeldef/model/skin parameter to revert to the default.

Parameters

modeldef: This is the MODELDEF this actor will now load. The default modeldef can be restored with "".

modelindex: This is the model index where the new model should be attached or replaced. Default is 0.

modelpath: This is the path for where to locate the new model. Default is "".

model: This is the file name of the new model. The extension is required. "" can be used to restore the default model for this index. Default is "".

skinindex: This index specifies which model index must change it's skin. Default is 0.

skinpath: This is the path for where to locate the new skin. Default is "".

skin: This is the file name of the new skin. The extension is required. "" can be used to restore the default skin for this index. Default is "".

flags: Allows the alteration of the function's behavior. The following flags can be combined by using the | character between the constant names:

CMDL_WEAPONTOPLAYER â€” If used on a weapon, this instead change's the model on the player instead.

CMDL_HIDEMODEL â€” Hides the specified model index from rendering. Useful to temporary hide part of the model. Can be used with CMDL_WEAPONTOPLAYER.

CMDL_USESURFACESKIN â€” skinindex instead corresponds to the index of a surface to replace its skin. Default is 0.

generatorindex: Instructs the model in modelindex to copy frame data from this index. Default is -1, which will not copy frame data from any index. Useful for attaching models with similar frames, like a gun to a player model.

animationindex: This index specifies where the new animation should be attached or replaced. Default is 0. (New from 4.10.0)

animationpath: This is the path for where to locate the new animation clip. Default is "" (New from 4.10.0)

animation: This is the file name of the new animation clip. The extension is required. "" can be used to restore the default animation for this index. Default is "" (New from 4.10.0)

Examples

This actor alternates modeldefs every second.

```
actor SwapModelDef
```

```
{
```

```
    Height 16
```

```
    Radius 8
```

```
    States
```

```
    {
```

```
    Spawn:
```

```
        PIST A 35
```

```
        TNT1 A 0 A_ChangeModel("SwapModelDefHelper")
```

```
        PIST A 35
```



```

    TNT1 A 0 A_ChangeModel("SwapModelDef")
    Loop
}
}

```

Another useful function of A_ChangeModel is the ability to change the player's appearance from the weapon itself. Note that index 0 represents the player model, while index 1 is attaching a gun to the player model.

actor NewPistol : Pistol replaces Pistol

```

{
    States
    {
        Ready:
            PISG A 0 A_ChangeModel("", 1, "Models", "w_blaster.md2", 1, "Models", "g_blaster.png",
            CMDL_WEAPONTOPPLAYER, 0)
            PISG A 1 A_WeaponReady
            Wait
    }
}

```

A_ChangeVelocity

A_ChangeVelocity [(float x [, float y [, float z [, int flags [, int ptr]]]])]

Usage

Changes the calling actor's or the pointed to actor velocity on each axis. If the "relative axes" flag is set, the "x" argument will represent forward and backward movement and the "y" argument will represent side-to-side movement. If the "replace old velocity" flag is set, the actor's velocity will be set to the new velocity; otherwise, the new velocity will be added to the old velocity. Negative values indicate movement backwards, right, and down respectively.

Parameters

x: velocity on x axis (or forward and backward). Default is 0.

y: velocity on y axis (or side to side). Default is 0.

z: velocity on z axis (up and down). Default is 0.

flags: The following flags can be combined by using the | character between the constant names:

CVF_RELATIVE â€” Relative axes: Make x, y relative to actor's current angle.

CVF_REPLACE â€” Replace old velocity: Replace old velocity with new velocity.

ptr: The actor to change its velocity. This is an actor pointer. Default is AAPTR_DEFAULT, which corresponds to the calling actor.

Examples

This makes the rocket go forward twice as fast without changing the speed at which it falls/ascends.

Totally breaks it's vertical path (off the crosshair), so don't use this for things that don't fall down or are otherwise vertically unstable. (Or do, but not as blatantly.):

ACTOR TurboRocket : Rocket

```
{
  States
  {
    Spawn:
      TNT1 A 0
      MISL A -1 A_ChangeVelocity(velx*2, vely*2, velz, CVF_REPLACE)
      //We do this by getting the appropriate velocities (x/y of velx/y/z) and multiplying them by 2,
      //and replacing the old ones.
      //Do note that this causes the rocket to not follow the crosshair. So it's best suited for things
      //that already fall do not follow it. (Like grenades)
      Stop
  }
}
```

This is a boomerang-style projectile that reverses direction after traveling a set distance, then enters its Death state after making the return trip (assuming it doesn't hit anything along the way):

ACTOR BoomerangBall : PlasmaBall

```
{
  States
  {
    Spawn:
      PLSS AB 6 Bright
      PLSS A 0 A_ChangeVelocity (-25, 0, -velz, CVF_RELATIVE|CVF_REPLACE)
      //Entering "-velx" does not function correctly, so use proper numbers for reversal.
      PLSS AB 6 Bright
    Death:
      PLSE ABCDE 4 Bright
  }
}
```

Stop

}

}

A_ClearShadow

(no parameters)

Makes the monster opaque and clears the SHADOW flag.

Examples

This spectre becomes visible if its health goes below 30 (provided it enters its Pain state at least once):

Actor Spectre4ZDoomWiki: Spectre replaces Spectre

```
{
  States
  {
    Pain:
      SARG H 2 Fast
      SARG H 2 Fast A_Pain
      SARG H 0 A_JumpIfHealthLower(30, "MakingVisible")
      Goto See
    MakingVisible:
      SARG H 0 A_ClearShadow
      Goto See
  }
}
```

A_CopyFriendliness

A_CopyFriendliness [(pointer copyfrom)]

The calling actor changes its affiliation (friendly or not, and allied to which player if friendly) to match that of the pointed actor.

The copyfrom pointer can be any of the following:

AAPTR_DEFAULT: The calling actor itself (does nothing)

AAPTR_NULL: No actor at all (does nothing)

AAPTR_TARGET: The calling actor's target, if any

AAPTR_MASTER: The calling actor's master, if any (default behavior)

AAPTR_TRACER: The calling actor's tracer, if any

Remember that the nature of these pointers depend on the actor type and are not always intuitive.

For more information on pointers, see the [actor pointers](#) page.

A_DeQueueCorpse

(no parameters)

Removes the actor's corpse from the corpse queue. Queued corpses are limited to a specific amount set by the sv_corpsequeuesize console variable. Hexen uses this to limit the amount of dead items in the game. It might be of interest for mods that produce a lot of dead bodies. Actors that you wish to use this feature should have the A_DeQueueCorpse special occur somewhere in their raise state.

A_FaceMovementDirection

bool A_FaceMovementDirection [(float offset [, float anglelimit [, float pitchlimit [, int flags [, int ptr]]]])]

Usage

Faces the calling actor's ptr to the direction of movement speed, offsetting the angle of the actor by offset after limiting the turning by anglelimit and pitchlimit, if specified.

Parameters

offset - Offsets the actor's angle by this amount, which is applied after the anglelimit. Default is 0.

anglelimit - Specifies the maximum angle the actor can turn in degrees. Default is 0.

pitchlimit - Specifies the maximum pitch the actor can change in degrees. Default is 0.

flags - Flags can be combined with the | separator.

FMDF_NOPITCH - Disables pitch adjusting.

FMDF_NOANGLE - Disables angle adjusting.

FMDF_INTERPOLATE - Interpolates the adjustments of pitch and/or angle.

ptr - The actor pointer to modify. Defaults to AAPTR_DEFAULT (self). Cannot be NULL.

A_FadeIn

A_FadeIn [(float increase_amount [, int flags])]

Usage

Increases the actor's opacity (alpha) by the specified amount. This can be used to slowly fade in some things in a loop.

Parameters

increase_amount: The amount by which to increase the actor's alpha. Default is 0.1.

flags: The following flags can be combined using the pipe character | between the constant names:

FTF_REMOVE " the actor is removed from the game once its alpha reaches 1.0.

FTF_CLAMP " the alpha cannot go above 1.0.

Examples

This cacodemon fireball slowly fades into existence when it is fired.

ACTOR Ghostball : CacodemonBall

```
{
  RenderStyle "Translucent"
  Alpha 0
  States
  {
    Spawn:
      BAL2 AB 4 Bright A_FadeIn(0.2)
    Loop
  }
}
```


A_FadeOut

A_FadeOut [(float reduce_amount [, int flags])]

Usage

Increases the actor's translucency (decreases its alpha) by the specified amount. When the actor is fully transparent it will be removed, unless remove is explicitly set to false (it is true by default). This can be used to slowly fade out some things in a loop. If reduce_amount is not provided, it defaults to 0.1.

Parameters

reduce_amount: The amount by which to reduce the actor's alpha.

flags: The following flags can be combined using the pipe character | between the constant names:

FTF_REMOVE "the actor is removed from the game once its alpha reaches 0. This flag is implied by default, and is equivalent to using "TRUE" in earlier versions of ZDoom. To prevent this from occurring, use 0.

FTF_CLAMP "the alpha cannot go below 0.

Examples

This is a cacodemon fireball which slowly fades out of existence when it is fired.

ACTOR Fadeball : CacodemonBall

```
{
  RenderStyle "Translucent"
  Alpha 1.0
  States
  {
    Spawn:
      BAL2 AB 4 Bright A_FadeOut(0.2)
    Loop
  }
}
```

This code replaces plasma balls with a custom PlasmaBall with trails.

Actor CoolPlasmaBall : PlasmaBall replaces PlasmaBall

```
{
  Radius 9
  Height 6
  Scale 0.75
  RenderStyle Add
  Alpha 0.8

  States
  {
    Spawn:
      PLSS AAAAAABBBBBB 1 Bright A_SpawnItemEx("CPBTrail", 0, 0, 0, 0, 0, 0, 0, 0)
    Loop
  }
}
```

Actor CPBTrail

```
{
  Radius 9
  Height 6
  Scale 0.75
  RenderStyle Add
```

Alpha 0.8

+NOGRAVITY
+NOBLOCKMAP
+NOCLIP

States

{

Spawn:

PLSS AAAAAABBBBBB 1 Bright A_FadeOut

Loop

}

}

A_FadeTo

A_FadeTo (float target [, float amount [, bool remove]])

A_FadeTo (float target [, float amount [, int flags]])

Usage

Alters transparency towards target by amount. This can be used to slowly fade in or out some things in a loop.

Parameters

target: The alpha value to fade towards.

amount: The amount by which to change the actor's alpha. Default is 0.1.

remove: If true, the actor is removed from the game once its alpha reaches target. Otherwise, if false, it is not. Default is true.

flags: The following flags can be combined using the pipe character | between the constant names:

FTF_REMOVE "the actor is removed from the game once its alpha reaches target.

FTF_CLAMP "the alpha cannot go below 0 or above 1.0.

Example

Stealth Spider demon:

Actor StealthSpiderDemon4ZDWiki: SpiderMasterMind replaces SpiderMasterMind

```
{
States
{
See:
SPID A 0 A_FadeTo( 0.01, 0.15 )
SPID A 3 A_Metal
SPID A 0 A_FadeTo( 0.01, 0.15 )
SPID ABB 3 A_Chase
SPID B 0 A_FadeTo( 0.01, 0.15 )
SPID C 3 A_Metal
SPID C 0 A_FadeTo( 0.01, 0.15 )
SPID CDD 3 A_Chase
SPID D 0 A_FadeTo( 0.01, 0.15 )
SPID E 3 A_Metal
SPID E 0 A_FadeTo( 0.01, 0.15 )
SPID EFF 3 A_Chase
SPID F 0 A_FadeTo( 0.01, 0.15 )
Loop
Missile:
SPID A 1 A_FaceTarget
SPID AAAAAAAAAAAAAAAAAAAAAA 1 Bright A_FadeTo( 1.0 )
SPID A 1 A_FaceTarget
SPID G 4 Bright A_SPosAttackUseAtkSound
SPID H 4 Bright A_SPosAttackUseAtkSound
SPID H 1 Bright A_SpidRefire
Goto Missile + 20
Pain:
SPID I 0 A_FadeTo( 1.0, 0.2 )
SPID III 1 A_FadeTo( 1.0, 0.2 )
SPID I 3 A_Pain
SPID I 0 A_FadeTo( 1.0, 0.2 )
Goto See
Death:
```

```
SPID J 0 A_FadeTo( 1.0 )
SPID J 1 A_Scream
SPID JJJJJJJJJJJJJJJJJ 1 A_FadeTo( 1.0 )
SPID K 10 A_NoBlocking
SPID LMNOPQR 10
SPID S 30
SPID S -1 A_BossDeath
Stop
}
}
```

A_Gravity

(no parameters)

Makes the calling actor subject to normal gravity so it will fall down when in mid-air.

Examples

Jump:

```
SILE H 0 A_SetFloat
SILE H 0 A_NoGravity
SILE H 0 A_ChangeFlag ("NODAMAGE", 1)
SILE HHH 1 A_SpawnItemEx ("flythrust")
SILE H 30 A_ChangeFlag ("NODAMAGE", 0)
SILE A 10
SILE P 0 A_Jump (80, "Missile2J")
SILE P 0 A_Jump (90, "MissileKJ")
SILE P 0 A_Jump (85, "MissileKiJ")
SILE L 1 BRIGHT A_FaceTarget
SILE LMN 3 BRIGHT A_FaceTarget
SILE O 10 BRIGHT A_FaceTarget
SILE P 0 A_PlaySound ("ki/shot")
SILE O 2 BRIGHT A_SpawnProjectile ("kiblast", 35, 0)
SILE N 8 BRIGHT A_PlaySound ("ka/charge", 0, 150.0)
SILE N 0 BRIGHT A_PlaySound ("kl/shot", 0, 150.0)
SILE LMNO 3
SILE O 10 BRIGHT A_SpawnProjectile ("KiblastBig", 35, 0)
Goto Land
```

Land:

```
SILE H 0 A_UnsetFloat
SILE H 25 A_Gravity
Goto see
```

This is the jump decorate code to the monster SaiyinVileGoku from Zero Invasion. The code starts out with taking out gravity with A_NoGravity, then, the monster gets a thrust upward causing the monster to appear to jump. When the monster is floating with no gravity, he needs to come back to the ground therefore; utilizing A_Gravity.

This whole decorate function will make this monster appear to jump, fly, and come back down using A_Gravity and A_NoGravity, just like a Saiyin in the Dragon Ball Z series.

A_HideThing

(no parameters)

Makes the calling actor invisible.

Examples

This is a modified doomimp that, when hurt, will suddenly turn invisible, through use of A_HideThing. If it is killed, or left alone to walk for a while, it triggers A_UnHideThing, which makes the imp visible again.

ACTOR HidingImp : Doomimp

```
{
HitObituary "%o was killed by a hiding imp."
Obituary "%o was slashed by a hiding imp."
States
{
See:
TROO AABBCDDAABBCDDAABBCDDAABBCDDAABBCDD 3 A_Chase
TROO A 10 A_UnHideThing
Loop
Melee:
Missile:
TROO EF 8 A_FaceTarget
TROO E 0 A_UnHideThing
TROO G 6 A_TroopAttack
Goto See
Pain:
TROO H 2
TROO H 2 A_Pain
TROO H 2 A_HideThing
Goto See
Death:
TROO I 8 A_UnHideThing
TROO J 8 A_Scream
TROO K 6
TROO L 6 A_NoBlocking
TROO M -1
Stop
XDeath:
TROO N 5 A_UnHideThing
TROO O 5 A_XScream
TROO P 5
TROO Q 5 A_NoBlocking
TROO RST 5
TROO U -1
Stop
}
}
```

A_LowGravity

(no parameters)

Makes the calling actor subject to low gravity so it will fall down when in mid-air - but very slowly.

Examples

This pain elemental so sluggish that falls when die too slow, too.

Actor SlowPainElem4ZDWiki: PainElemental replaces PainElemental

```
{
  Translation "64:79=16:47", "128:143=152:159"
  Speed 4
  PainChance 96
  Health 500
  States
  {
  See:
    PAIN AAAABBBBBCCCC 3 A_Chase( "" )
    Loop
  Death:
    PAIN H 8 Bright A_LowGravity
    PAIN I 8 Bright A_Scream
    PAIN JK 8 Bright
    PAIN L 8 Bright A_PainDie
    PAIN M 8 Bright
    Stop
  }
}
```

A_Morph

```
bool A_Morph (class<Actor> type [, int duration [, int flags [, class<Actor> enter_flash [, class<Actor> exit_flash]]]])
```

Usage

Morphs the calling actor into the specified actor class. For more information, see MorphActor and MorphProjectile.

Parameters

type: The type of actor class to morph to.

duration: The time in tics for how long to stay morphed. Default is 0, which is interpreted as 1400 tics (40 seconds).

flags: Defines the behaviour of the morphing effect. See MorphProjectile.MorphStyle property for the available flags. Default is 0.

enter_flash: Defines the effect flash actor to spawn when morphing. If omitted or is null, the game's default teleport fog is used. Default is null.

exit_flash: Defines the effect flash actor to spawn when unmorphing. If omitted or is null, the game's default teleport fog is used. Default is null.

Return value

The function returns true if the morph was successful, otherwise it returns false.

A_NoBlocking

void A_NoBlocking [(bool drop)]

void A_Fall ()

Usage

The function does the following:

It makes the actor visible if it is stealth.

It makes it non-solid.

It makes it spawn its dialogue-set and "regular" drop items.

It clears the dialogue that is assigned to it.

Note that unlike the dialogue-set drop item, which is cleared from the actor after it is spawned (cleared since the actor no longer has a dialogue assigned to it), the regular drop items are not, which means that these items are spawned each time the function is called. Also, if the actor has both dialogue-set and regular drop items, it only spawns the dialogue-set one for that function call. For all subsequent calls, the regular drop items are spawned, instead.

A_Fall is an alternative name which is Doom's original name for this function. It functions identically to the default A_NoBlocking call with default parameters.

Parameters

drop: Determines whether the actor's DropItem will be spawned. Default is true.

A_NoGravity

(no parameters)

Makes the calling actor not subject to gravity so it won't fall down anymore.

A_QueueCorpse

(no parameters)

Adds the actor's corpse to the corpse queue. Queued corpses are limited to a specific amount set by the `sv_corpsequeue` console variable. Hexen uses this to limit the amount of dead items in the game. It might be of interest for mods that produce a lot of dead bodies. Actors that you wish to use this feature should have this special occur somewhere in their death state.

A_RearrangePointers

ZScript note: Setting pointers in ZScript is as simple as referencing them by name, and as such this function is not strictly necessary. See using pointers in ZScript for more information.

A_RearrangePointers (pointer target, pointer master, pointer tracer, int flags)

The calling actor swaps its actor pointers.

target corresponds to the value to store in the target field.
master corresponds to the value to store in the master field.
tracer corresponds to the value to store in the tracer field.
The pointers can be any of the following:

AAPTR_DEFAULT: no change is performed.

AAPTR_NULL: no actor at all.

AAPTR_TARGET: the calling actor's target, if any.

AAPTR_MASTER: the calling actor's master, if any.

AAPTR_TRACER: the calling actor's tracer, if any.

Remember that the nature of these pointers depend on the actor type and is not always intuitive.

By default master and target become NULL if values are assigned that would cause infinite relationships, such as missiles targeting each other, masters mastering each other. The following flags can manipulate and disable the safe guards to allow for more complex relationships:

PTROP_UNSAFETARGET (1) "do not nullify assignments that result in an infinite chain of missiles referencing each other.

PTROP_UNSAFEMASTER (2) "do not nullify assignments that result in an infinite chain of actors referencing each other.

PTROP_NOSAFEGUARDS (3) "same as putting in PTROP_UNSAFETARGET|PTROP_UNSAFEMASTER.

Note that setting a target to AAPTR_NULL with this function does not perform all actions of A_ClearTarget, that clears other related fields.

Examples

This imp has a 4/256 chance to "forget" its master and its target.

ACTOR AmnesiacImp : DoomImp

```
{
    States
    {
        See:
        TROO A 0 A_Jump(252, 2)
        TROO A 0 A_RearrangePointers(AAPTR_NULL, AAPTR_NULL, AAPTR_DEFAULT)
        TROO AABBCDD 3 A_Chase
        Loop
    }
}
```

A_ResetHealth

A_ResetHealth [(int pointer)]

Usage

Sets the pointed to actor's health to the value it spawned in the game with. This function only has an effect on actors with health value greater than 0.

Parameters

pointer: the actor to reset its health. This is an actor pointer. Default is AAPTR_DEFAULT, which corresponds to the calling actor.

A_Respawn

A_Respawn [(int flags)]

Respawns the calling actor at their initial starting point, and resets its stats in the process. This behaves similarly to if the actor was respawned on the Nightmare skill. If there is a solid object at the spawn point, the respawn will fail, unless the RSF_TELEFRAG flag is set (see below).

Parameters

flags: The following flags can be combined by using the pipe character | between the constant names to alter the behavior of the function:

RSF_FOG “ The respawn fog is shown when the actor reappears. This is the default behavior if the flags parameter is omitted.

RSF_KEEPTARGET “ The calling actor's target is preserved after respawning.

RSF_TELEFRAG “ If a solid telefragable object is in the spawn spot, it will be telefragged and the respawn will be successful.

Examples

This imp will stay dead for 3 seconds before respawning. If it is killed normally it will respawn without fog, but if it is gibbed, it will respawn with one.

ACTOR ImmortalImp : DoomImp replaces DoomImp

```
{
  States
  {
    Death:
      TROO I 8
      TROO J 8 A_Scream
      TROO K 6
      TROO L 6 A_NoBlocking
      TROO M 105
      TROO M 0 A_Respawn(FALSE)
      Stop
    XDeath:
      TROO N 5
      TROO O 5 A_XScream
      TROO P 5
      TROO Q 5 A_NoBlocking
      TROO RST 5
      TROO U 105
      TROO U 0 A_Respawn
      Stop
  }
}
```

A_ScaleVelocity

A_ScaleVelocity (float scale [, int pointer])

Usage

Multiplies the calling actor's or the pointed to actor's velocity on each axis by scale. It can be used to "speed up" or "slow down" an actor.

Parameters

scale: The value by which to scale the actor's velocity.

pointer: The actor to scale its velocity. This is an actor pointer. Default is AAPTR_DEFAULT, which corresponds to the calling actor.

Examples

This lazy rocket slows down and eventually stops.

ACTOR LazyRocket : Rocket

```
{
  States
  {
    Spawn:
      MISL A 1 Bright A_ScaleVelocity(0.95)
    Loop
  }
}
```

A_ScreamAndUnblock

(no parameters)

Plays the death sound and then calls A_NoBlocking.

Examples

A_ScreamAndUnblock is a easier way of combining A_Scream and A_NoBlocking. The following code is the complicated version...

```
DEMO H 8
DEMO I 0 A_Scream
DEMO I 8 A_NoBlocking
DEMO J 6
DEMO K -1
stop
```

...and this is the easier version:

```
DEMO H 8
DEMO I 8 A_ScreamAndUnblock
DEMO J 6
DEMO K -1
stop
```


A_SetAngle

A_SetAngle (float angle [, int flags [, int ptr]])

Usage

Sets the calling actor's or the pointed to actor's angle to angle. It can be used with the angle DECORATE variable to modify the actor's current angle.

Parameters

angle: the actor's new angle, in degrees.

flags:

SPF_INTERPOLATE: interpolates the view.

ptr: The actor to change its angle. This is an actor pointer. Default is AAPTR_DEFAULT, which corresponds to the calling actor.

Examples

This zombie spins in circles in his spawn state.

ACTOR SpinningZombieMan:ZombieMan replaces ZombieMan

```
{
  States
  {
    Spawn:
      POSS ABCD 4 A_SetAngle(15+angle)
      loop
  }
}
```

This firing state randomly jumps the view to the left or right using A_SetAngle.

Fire:

```
PISG B 0 A_GunFlash
PISG B 0 A_Jump(128, "SpinLeft")
PISG B 0 A_SetAngle(angle+((abs(velx)+abs(vely)+abs(velz))/10))
goto SpinDone
```

SpinLeft:

```
PISG B 0 A_SetAngle(angle-((abs(velx)+abs(vely)+abs(velz))/10))
```

SpinDone:

```
PISG B 0 A_SetPitch(pitch-((abs(velx)+abs(vely)+abs(velz))/6))
PISG B 0 A_PlaySound("Weapons/Pistol")
PISG B 3 offset(3, 39) A_FireBullets(1, 1, -1, 12, "BulletPuff")
PISG C 2 offset(2, 37)
PISG D 2 offset(1, 35)
PISG D 0 offset(0, 32)
Goto Ready
```

A_SetArg

A_SetArg (int position, int value)

Changes the calling actor's args[position] field to value.

Examples

This health bonus will have a float bobbing movement (like Heretic's and Hexen's powerups and items) if its first argument (Args[0]) is higher than 0. It won't enter the FloatBobMovement state more than once because Args[0] will be changed to 0 if the actor enters this state.

ACTOR FloatBobHealthBonus : HealthBonus

```
{
  States
  {
    Spawn:
      BON1 A 0 A_JumpIf(Args[0]>0, "FloatBobMovement")
      BON1 ABCDCB 6
      Loop
    FloatBobMovement:
      BON1 A 0 A_SetArg(0, 0)
      BON1 A 0 A_ChangeFlag("FLOATBOB", 1)
      Goto Spawn
  }
}
```

This actor will have the same behavior as D'Sparil

ACTOR NewSorcerer2 : Sorcerer2

```
{
  States
  {
    Death:
      SDTH A 8 A_SetArg(4, 7) //Replicates A_Sor2DthInit
      Goto Super::Death+2
    DeathLoop:
      SDTH DE 7
      SDTH F 7 A_SetArg(4, Args[4]-1)
      SDTH F 0 A_JumpIf(Args[4]>0,"DeathLoop") //Replicates A_Sor2DthLoop
      Goto Super::DeathLoop +4
  }
}
```

A_SetDamageType

A_SetDamageType(name damagetype)

Sets the specified damage type for the caller. This accepts the already-defined damage types supported by the engine as well as user-defined ones.

Examples

This rocket will do two types of damage: Direct hit causes default non-elemental damage, but the explosion itself causes fire damage.

ACTOR RocketElement : Rocket

```
{
  States
  {
    Death:
      MISL B 0 Bright A_SetDamageType("Fire")
      MISL B 8 Bright A_Explode
      MISL C 6 Bright
      MISL D 4 Bright
      Stop
  }
}
```

A_SetFloat

(no parameters)

Makes a monster floating. Floating monsters can change their z-position when moving towards a target. Setting this is only useful when combined with A_NoGravity or setting the NOGRAVITY flag.

A_SetFloorClip

(no parameters)

Sets the FLOORCLIP flag so the calling actor is affected by a terrain's footclip property.

Examples

The Cacodemon calls this function when slain, so its corpse will clip on liquid terrains (such as water).

Death:

HEAD G 8

HEAD H 8 A_Scream

HEAD IJ 8

HEAD K 8 A_NoBlocking

HEAD L -1 A_SetFloorClip

Stop

A_SetFriendly

void A_SetFriendly (bool set)

Usage

Sets or clears the FRIENDLY flag of the calling actor while updating the total monster count of the map accordingly. If the calling actor's health is 0 or below, the flag is merely changed, while the total monster count remains untouched.

Parameters

set: whether to set or clear the flag. If this is true, the flag is set, otherwise if it is false, the flag is cleared.

A_SetGravity

A_SetGravity (float gravity)

Sets the amount of gravity for the calling actor.

A_SetHealth

A_SetHealth(int health [, int pointer])

Usage

Sets the pointed to actor's health to health. This function cannot kill an actor; if the defined health is 0 or less, it will set the actor's health to 1 instead.

Functions such as A_Kill* or A_Damage* should be used for such a task, instead.

Parameters

health: The health value to set for the actor. Valid values are 1 and above.

pointer: The actor to set its health. This is an actor pointer. Default is AAPTR_DEFAULT, which corresponds to the calling actor.

A_SetInvulnerable

(no parameters)

Makes the calling actor invulnerable.

A_SetMass

A_SetMass (int mass)

Changes the calling actor's mass to mass. It can be used with the mass DECORATE variable to modify the actor's current mass.

A_SetPitch

A_SetPitch (float pitch [, int flags [, int ptr]])

Usage

A_SetPitch sets the calling actor's or the pointed to actor's pitch to the value of the first parameter. It can be used with the pitch DECORATE variable to modify the actor's current pitch.

For players, the pitch will be clamped to the [-90, 90] range. The limit can be applied to other actors as well with the SPF_FORCECLAMP flag.

Parameters

pitch: The actor's new pitch, in degrees. Negative numbers makes actor aim higher, while positive numbers make the actor aim lower.

flags: The following flags can be combined by using the pipe character | between the constants names:

SPF_FORCECLAMP: Forces the pitch to be within [-90, 90] range. This flag is always forced on for players.

SPF_INTERPOLATE: Interpolates the view.

ptr: The actor to change its pitch angle. This is an actor pointer. Default is AAPTR_DEFAULT, which corresponds to the calling actor.

Examples

This firing script uses A_SetPitch to add vertical recoil to the pistol.

Fire:

```
PISG A 0 A_GunFlash
PISG A 0 Offset(3, 39) A_FireBullets(1, 1, -1, 12, "BulletPuff")
PISG A 0 A_SetPitch(pitch-1) //Kick the aim up
PISG B 3 A_PlaySound("Weapons/Pistol")
PISG CC 1 Offset(2, 37) A_SetPitch(pitch+0.25) //Lower the aim back down
PISG DD 1 Offset(1, 35) A_SetPitch(pitch+0.25)
PISG A 0 Offset(0, 32)
Goto Ready
```

A more advanced version:

Fire:

```
PISG A 0 A_GunFlash
PISG B 0 Offset(3, 39) A_FireBullets(1, 1, -1, 12, "BulletPuff")
PISG A 0 A_Jump(128, 2)
PISG A 0 A_SetAngle(angle+((abs(velx)+abs(vely)+abs(velz))/10))
PISG A 0 A_Jump(256, 1)
PISG A 0 A_SetAngle(angle-((abs(velx)+abs(vely)+abs(velz))/10))
PISG A 0 A_SetPitch(pitch-((abs(velx)+abs(vely)+abs(velz))/6))
PISG B 3 A_PlaySound("Weapons/Pistol")
PISG C 2 Offset(2, 37)
PISG D 2 Offset(1, 35)
PISG A 0 Offset(0, 32)
Goto Ready
```

A_SetReflective

(no parameters)

Makes the calling actor reflect missiles that hit him.

A_SetReflectiveInvulnerable

(no parameters)

Makes the calling actor invulnerable and makes him reflect missiles.

Examples

This function is used by Hexen's Centaur in its Pain state:

Pain:

CENT G 6 A_Pain

CENT G 6 A_SetReflectiveInvulnerable

CENT EEE 15 A_CentaurDefend

CENT E 1 A_UnSetReflectiveInvulnerable

Goto See

A_SetRenderStyle

A_SetRenderStyle(float alpha, int style)

Usage

Sets the alpha and render style of the actor.

This function is a replacement for A_SetTranslucent.

Parameters

alpha: The alpha value to set, in the range of 0 to 1.0.

style: The render style to set, which can be one of the following:

STYLE_None

STYLE_Normal

STYLE_Fuzzy

STYLE_SoulTrans

STYLE_OptFuzzy

STYLE_Stencil

STYLE_Translucent

STYLE_Add

STYLE_Shaded

STYLE_TranslucentStencil

STYLE_Shadow

STYLE_Subtract

STYLE_AddStencil

STYLE_AddShaded

For information on these styles see [here](#).

A_SetRipMax

A_SetRipMax (int max)

Changes the calling actor's ripping minimum level to max.

This is only beneficial to monsters. Any monster whose maximum ripping level is less than that of a projectile will prevent it from ripping into them, as if the monster had the DONTRIP flag for that particular projectile. Having 0 as a value means no maximum, as long as the projectile is a ripper and has something higher than the minimum, then it can rip through this monster.

This may still be bypassed if the monster has a specific ripper level. I.e. a monster has a minimum of 1 and a maximum of 5 with a direct ripping level of 10, then ripper levels of 1 through 5 OR 10 can successfully rip them.

A_SetRipMin

A_SetRipMin (int min)

Changes the calling actor's ripping minimum level to min.

This is only beneficial to monsters. Any monster whose minimum ripping level is greater than that of a projectile will prevent it from ripping into them, as if the monster had the DONTRIP flag for that particular projectile.

A_SetRipperLevel

A_SetRipperLevel (int level)

Changes the calling actor's ripper level to level. This is not a property that can be used with expressions.

Projectiles have the greatest effect with this, as long as they're RIPPERS. The projectile can rip through any monster if its ripper level is within the monster's minimum and maximum ripper levels or equal to the monster's own ripper level. I.e. a monster has a minimum of 1 and a maximum of 5 with a direct ripping level of 10, then ripper levels of 1 through 5 OR 10 can successfully rip them.

A_SetRoll

Warning: This feature is GZDoom specific, and is not compatible with ZDoom!

A_SetRoll (float roll [, int flags [, int ptr]])

Usage

Sets the calling or pointed to actor's roll to the value of the first parameter. It can be used with the roll DECORATE variable to modify the actor's current roll. This affects the player camera, models and the sprites of actors with the ROLLSPRITE flag.

For players, the view is rolled and so is the model, if one is supplied.. Positive numbers rotate the screen counter-clockwise (left side rising, right side lowering), while negative numbers rotate it clockwise (right side rising, left side lowering).

Parameters

roll: The actor's new roll, in degrees. The range is [0,359].

flags: The following flags can be combined by using the pipe character | between the constants names:

SPF_INTERPOLATE: Interpolates the view, smoothing it out over the delay of 1 tic.

ptr: The actor to change its roll. This is an actor pointer. Default is AAPTR_DEFAULT, which corresponds to the calling actor.

Examples

This example is of a floating object that is rotated randomly between 0 and 359 degrees every time it is loaded into the game.

ACTOR RotatedThing

```
{
  Radius 32
  Height 64
  +NOGRAVITY
  +ROLLSPRITE
  +ROLLCENTER
  States
  {
    Spawn:
      FBTE A -1 NoDelay A_SetRoll(random(0, 359))
      Stop
  }
}
```

A_SetScale

A_SetScale (float scaleX [, float scaleY [, int pointer [, bool usezero]]])

Usage

Changes the calling actor's or the pointed to actor's visual scale. This does not affect the actual collision box and is mainly intended for special effects actors, such as a puff of smoke gradually dissipating by expansion (in combination with A_FadeOut) or a mote of light shrinking and disappearing.

Parameters

scaleX: the actor's new horizontal scale. Using negative values will result in mirroring the sprite on the axis.

scaleY: the actor's new vertical scale. If this parameter is not given, or is set to 0, scaleX is used as well. Default is 0.

pointer: The actor to change its scale. This is an actor pointer. Default is AAPTR_DEFAULT, which corresponds to the calling actor.

usezero: If this is false and scaleY is 0, scaleY uses the same value passed to scaleX, otherwise if it is true, the value of scaleY is used, instead. Default is false.

Examples

This simple actor when spawned will gradually become smaller and more translucent, until it is completely invisible and is removed. It could be used as a trail for some energy-like weapons.

ACTOR PulseRifleSmoke

```
{
States
{
Spawn:
    PMIS B 0 Bright A_SetScale(0.6)
    PMIS B 1 bright A_FadeOut(0.1)
    PMIS B 0 Bright A_SetScale(0.5)
    PMIS B 1 bright A_FadeOut(0.1)
    PMIS B 0 Bright A_SetScale(0.4)
    PMIS B 1 bright A_FadeOut(0.1)
    PMIS B 0 Bright A_SetScale(0.3)
    PMIS B 1 bright A_FadeOut(0.1)
    PMIS B 0 Bright A_SetScale(0.2)
    PMIS B 1 bright A_FadeOut(0.1)
    PMIS B 0 Bright A_SetScale(0.1)
    PMIS B 1 bright A_FadeOut(0.1)
    PMIS BBBB 0 Bright A_FadeOut(0.1)
Stop
}
}
```

A_SetShadow

(no parameters)

Makes the monster 40% translucent and sets the SHADOW flag.

A_SetShootable

(no parameters)

Makes the calling actor shootable by clearing the NONSHOOTABLE flag and setting the SHOOTABLE flag instead.

A_SetSize

bool A_SetSize (double newradius = -1, double newheight = -1, bool testpos = false)

Usage

Changes the blocking dimensions (radius and height) of an actor. It can also check to see if the actor fits after the change, and reverts should the check fail.

Parameters

newradius: The actor's new radius. Any negative value means no change. Default is -1.

newheight: The actor's new height. Any negative value means no change. Default is -1.

testpos: If true, the actor performs a check to see if it will fit in the current position with the new size.

Return Value

Returns false only if testpos is used and the actor is determined to be stuck upon changing size. Returns true otherwise.

A_SetSolid

(no parameters)

Makes the calling actor solid.

A_SetSpecial

A_SetSpecial (int special, int arg0, int arg1, int arg2, int arg3, int arg4)

Changes the calling actor's special action and its arguments.

A_SetSpecies

A_SetSpecies (str species [, int ptr])

Parameters

species: Changes the calling actor's (or the pointed to actor's) species to the defined parameter. Use "None" to remove it from all species and subject it to infighting.

pointer: The pointed actor to change instead of itself.

A_SetSpeed

A_SetSpeed (float speed [, int pointer])

Usage

Changes the calling actor's or the pointed to actor's speed to speed. It can be used with the speed DECORATE variable to modify the actor's current speed. Note that changing a projectile's speed will not affect it's velocity unless it is a seeker missile.

Parameters

speed: The value to set the actor's speed to.

pointer: The actor to change its speed. This is an actor pointer. Default is AAPTR_DEFAULT, which corresponds to the calling actor.

A_SetTeleFog

A_SetTeleFog (class telefogsourcetype, class telefogdesttype)

Changes the calling actor's TeleFogSourceType and TeleFogDestType.

If the actor classes for the function are set to "none" or they do not exist, the actor will spawn nothing for those fields when they teleport.

A_SetTics

A_SetTics(int duration)

Changes the current duration of the tic the action function is called.

Examples

A weird, slow imp with a lot of health who becomes faster when hurt.

```
actor WeirdImp : DoomImp {
  Health 300
  States {
  See:
    TROO A 6 A_SetTics (health / 20)
    TROO A 0 A_Chase
    TROO B 6 A_SetTics (health / 20)
    TROO B 0 A_Chase
    TROO C 6 A_SetTics (health / 20)
    TROO C 0 A_Chase
    TROO D 6 A_SetTics (health / 20)
    TROO D 0 A_Chase
    Loop
  }
}
```

A_SetTranslation

void A_SetTranslation (name transname)

Usage

Applies a TRNSLATE-defined translation to the calling actor. Unlike Thing_SetTranslation, this accepts an actual name instead of an ID.

Parameters

transname: the name of the translation to use.

Examples

This is a regular zombie which has a pain state that allows it to be "glitched", randomly changing its stats, as well as changing its translation.

ACTOR GlitchyZombie : ZombieMan

```
{
    States
    {
        Pain.Glitch:
            POSS G 1 A_PlaySound("Glitch/Explode")
            POSS G 1 A_GiveInventory("Glitcho", 1)
            POSS G 1 A_SetHealth(random(1, 200))
            POSS G 1 A_SetSpecies("Glitchy")
            POSS G 1 A_SetMass(random(0, 1000))
            POSS G 1 A_SetSpeed(random(10, 40))
            POSS G 1 A_ChangeFlag("SPRITEFLIP", 1)
            POSS G 1 A_ChangeFlag("NOTIMEFREEZE", 1)
            POSS G 1 A_ChangeFlag("BRIGHT", 1)
            POSS G 1 A_SetTranslation("Glitch")
            POSS G 3 A_Pain
            Goto See
    }
}
```

A_SetTranslucent

void A_SetTranslucent (double alpha [, int style])

Note: This function has been superseded by A_SetRenderStyle, which duplicates and extends its functionality. Use of the newer function is advised in order to maintain maximum flexibility in your code.

Usage

Sets translucency for the calling actor.

Parameters

alpha: specifies the amount of visibility and is a value between 0.0 (not visible) and 1.0 (fully visible).

style: the translucency mode to set:

0: normal translucency blending. This mode combined with an alpha setting of 1.0 will make the actor opaque. This is the default mode.

1: additive blending.

2: fuzz effect. Alpha has no meaning with this setting.

Examples

This rocket variant displays its explosion with additive translucency. Slight visual enhancement.

ACTOR PrettyRocket : Rocket replaces Rocket

```
{
  States
  {
    Death:
      MISL B 0 A_SetTranslucent(0.75, 1)
      Goto Super::Death
  }
}
```

A_SetUserArray

A_SetUserArray (string name, int index, int value)

Sets the calling actor's user array of type int named name, at position index, to value. The name must begin with user_.

User arrays are not used by anything internally, contrarily to the args array and the special1 and special2 fields.

A_SetUserArrayFloat

A_SetUserArrayFloat (string name, int index, float value)

Sets the calling actor's user array of type float named name, at position index, to value. The name must begin with user_.

User arrays are not used by anything internally, contrarily to the args array and the special1 and special2 fields.

A_SetUserVar

A_SetUserVar (string name, int value)

Sets the calling actor's user variable of type int named name to value. The name must begin with user_.

User variables are not used by anything internally, contrarily to the args array and the special1 and special2 fields.

Notes

User variables on weapons will not work unless they are defined on the player actor itself.

CustomInventory items can modify a monster's variables, but it must be defined in both the monster, and the CustomInventory actors. Until the CustomInventory enters its Use state legitimately (NOT through state jumps, but actual internal triggering), it will only modify its own user variables. If the inventory item needs modifying for only itself once grabbed, do so in the Pickup state.

Examples

This code creates an imp which attacks with a 360 degree "shockwave" attack composed of 360 individual fireballs. Old mods achieved this effect by reproducing a A_CustomMissile call many times. This cuts down on the amount of lines needed and is much cleaner to implement.

actor ShockWaveDoomImp : DoomImp replaces DoomImp

```
{
  var int user_theta;
  states
  {
    Missile:
      TROO EF 8 A_FaceTarget
      TROO G 0 A_SetUserVar("user_theta",0)
    Shock:
      TROO G 0 A_SpawnProjectile("DoomImpBall",32,0,user_theta)
      TROO G 0 A_SetUserVar("user_theta",user_theta+1)
      TROO G 0 A_JumpIf(user_theta==360,"EndShock")
      Loop
    EndShock:
      TROO G 6
      Goto See
  }
}
```

A_SetUserVarFloat

A_SetUserVarFloat (string name, float value)

Sets the calling actor's user variable of type float named name to value. The name must begin with user_.

User variables are not used by anything internally, contrarily to the args array and the special1 and special2 fields.

Notes

User variables on weapons will not work unless they are defined on the player actor itself.

CustomInventory items can modify a monster's variables, but it must be defined in both the monster, and the CustomInventory actors. Until the CustomInventory enters its Use state legitimately (NOT through state jumps, but actual internal triggering), it will only modify its own user variables. If the inventory item needs modifying for only itself once grabbed, do so in the Pickup state.

Examples

This code creates an imp which attacks with a 360 degree "shockwave" attack composed of 360 individual fireballs. Old mods achieved this effect by reproducing a A_SpawnProjectile call many times. This cuts down on the amount of lines needed and is much cleaner to implement.

```
actor ShockWaveDoomImp : DoomImp replaces DoomImp
{
    var float user_theta;
    states
    {
        Missile:
            TROO EF 8 A_FaceTarget
            TROO G 0 A_SetUserVarFloat("user_theta",0.0)
        Shock:
            TROO G 0 A_SpawnProjectile("DoomImpBall",32,0,user_theta)
            TROO G 0 A_SetUserVarFloat("user_theta",user_theta + 1.1)
            TROO G 0 A_JumpIf(user_theta >= 360.0,"EndShock")
        Loop
        EndShock:
            TROO G 6
            Goto See
    }
}
```

A_SetViewAngle

void A_SetViewAngle [(double angle [, int flags [, int ptr]])]

Usage

Parameters

angle: Default is 0.

flags: Default is 0.

ptr: Default is AAPTR_DEFAULT.

A_SetViewPitch

void A_SetViewPitch (double pitch [, int flags [, int ptr]])

Usage

Parameters

- pitch:
- flags: Default is 0.
- ptr: Default is AAPTR_DEFAULT.

A_SetViewRoll

void A_SetViewRoll (double roll [, int flags [, int ptr]])

Usage

Parameters

- roll:
- flags: Default is 0.
- ptr: Default is AAPTR_DEFAULT.

A_SwapTeleFog

A_SetTeleFog (no parameters)

Switches the calling actor's TeleFogSourceType and TeleFogDestType.

This function does nothing if the source and destination fogs are the same.

A_TransferPointer

A_TransferPointer (pointer source, pointer recipient, pointer sourcefield, pointer recipientfield[, int flags])

The calling actor performs a transfer of pointer between itself and another actor.

source corresponds to the actor to use as a model

recipient corresponds to the actor that will change its pointer

sourcefield corresponds to the pointer to transfer from the source (cannot be DEFAULT or NULL)

recipientfield corresponds to which pointer of the recipient will host the transferred value (cannot be DEFAULT or NULL)

The pointers can be any of the following:

AAPTR_DEFAULT: The calling actor itself

AAPTR_NULL: No actor at all (does nothing)

AAPTR_TARGET: The calling actor's target, if any

AAPTR_MASTER: The calling actor's master, if any

AAPTR_TRACER: The calling actor's tracer, if any

Remember that the nature of these pointers depend on the actor type and is not always intuitive.

By default MASTER and TARGET become null if values are assigned that would cause infinite relationships. (Missiles targeting each other, masters mastering each other.)

The following flags can manipulate and disable the safe guards to allow for more complex relationships:

PTROP_UNSAFETARGET (1) - Don't null assignments that result in an infinite chain of missiles referencing each other

PTROP_UNSAFEMASTER (2) - Don't null assignments that result in an infinite chain of actors referencing each other

PTROP_NOSAFEGUARDS (4) - Same as putting in PTROP_UNSAFETARGET|PTROP_UNSAFEMASTER, or 3 (3 and 4 do the same thing, which is redundant).

Additionally, since pointers travel between actors here, an extra check is added: If the actor would end up pointing to itself (which seems both dangerous and useless) the assigned pointer field is nulled. This check cannot be overridden.

For more information on pointers, visit the actor pointers page.

Examples

This Imp's master will acquire the same target as this imp.

ACTOR WimpyImp : DoomImp

```
{
  States
  {
    Missile:
      TNT1 A 0 A_TransferPointer(AAPTR_DEFAULT, AAPTR_MASTER, AAPTR_TARGET, AAPTR_TARGET)
      Goto Super::Missile
  }
}
```

A_UnHideThing

(no parameters)

Makes the calling actor visible.

Examples

This is a modified doomimp that, when hurt, will suddenly turn invisible, through use of A_HideThing. If it is killed, or left alone to walk for a while, it triggers A_UnHideThing, which makes the imp visible again.

ACTOR HidingImp : Doomimp

```
{  
HitObituary "%o was killed by a hiding imp."  
Obituary "%o was slashed by a hiding imp."
```

States

```
{
```

See:

```
    TROO AABBBCCDDAABBBCCDDAABBBCCDDAABBBCCDDAABBBCCDD 3 A_Chase
```

```
    TROO A 10 A_UnHideThing
```

```
    Loop
```

Melee:

Missile:

```
    TROO EF 8 A_FaceTarget
```

```
    TROO E 0 A_UnHideThing
```

```
    TROO G 6 A_TroopAttack
```

```
    Goto See
```

Pain:

```
    TROO H 2
```

```
    TROO H 2 A_Pain
```

```
    TROO H 2 A_HideThing
```

```
    Goto See
```

Death:

```
    TROO I 8 A_UnHideThing
```

```
    TROO J 8 A_Scream
```

```
    TROO K 6
```

```
    TROO L 6 A_NoBlocking
```

```
    TROO M -1
```

```
    Stop
```

XDeath:

```
    TROO N 5 A_UnHideThing
```

```
    TROO O 5 A_XScream
```

```
    TROO P 5
```

```
    TROO Q 5 A_NoBlocking
```

```
    TROO RST 5
```

```
    TROO U -1
```

```
    Stop
```

```
}
```

```
}
```


A_UnsetFloat

(no parameters)

Makes a monster non-floating. Floating monsters can change their z-position when moving towards a target. This function has no effect on the gravity settings. If the monster is still in the air it will remain there.

A_UnSetFloorClip

(no parameters)

Clears the FLOORCLIP flag so the calling actor is no longer affected by a terrain's footclip property.

A_UnSetInvulnerable

(no parameters)

Makes the calling actor vulnerable.

A_UnSetReflective

(no parameters)

Makes the calling actor no longer reflect missiles that hit him.

A_UnSetReflectiveInvulnerable

(no parameters)

Makes the calling actor vulnerable and makes him no longer reflect missiles.

Examples

This function is used by Hexen's Centaur in its Pain state:

Pain:

CENT G 6 A_Pain

CENT G 6 A_SetReflectiveInvulnerable

CENT EEE 15 A_CentaurDefend

CENT E 1 A_UnSetReflectiveInvulnerable

Goto See

A_UnSetShootable

(no parameters)

Makes the calling actor unshootable by clearing its SHOOTABLE flag and giving it the NONSHOOTABLE flag instead. An unshootable actor can no longer be hit by hitscan weapons and is unaffected by missiles and explosions.

A_UnsetSolid

(no parameters)

Makes the calling actor non-solid.

SetMugShotState

void SetMugShotState (str state) â€” ACS version

void A_SetMugshotState (String name) â€” Action function version

Usage

Used to set the state of the mug shot in SBARINFO status bars. The state you set will only be interrupted by damage or if the player picks up a weapon, provided the mugshot supports it.

Examples

The following example will make the player believe that they are invulnerable or make spectators think he/she is cheating.

```
script 1 (void)
```

```
{  
    SetMugShotState("God");  
}
```

The player grins when they pick up this berserk pack.

```
ACTOR GrinBerserk : Berserk
```

```
{  
    States  
    {  
        Pickup:  
            TNT1 A 0 A_SetMugshotState("Grin")  
            Goto Super::Pickup  
    }  
}
```


A_CustomPunch

A_CustomPunch (int damage [, bool norandom [, int flags [, string pufftype [, float range [, float lifesteal [, int lifestealmax [, string armorbonustype [, sound meleesound [, sound misssound]]]]]]]]))

Usage

Defines a custom punch attack. Damage can either be random or fixed and it is possible to specify whether ammo is used or the puff that is spawned when hitting the wall or a non-bleeding actor. It is also possible to define the maximum distance of the attack and its capacity to steal life from the target.

Parameters

damage: The damage to inflict. This can be a number or expression. If the second argument, norandom, is false then whatever number is given in damage is then multiplied by a random number between 1 and 8. If you want the number in damage to be the final damage output of the punch, or to use your own custom expression, set norandom to true.

flags: The following flags can be combined by using the | character between the constant names:

CPF_USEAMMO — Use ammo: the attack drains ammo as indicated by the appropriate Weapon.AmmoUse property.

CPF_DAGGER — Act like a PunchDagger: Monsters with the SEESDAGGERS flag are woken up if they see the attack, even if the weapon has the WEAPON.NOALERT flag. This replicates an element of A_JabDagger. Note that using this flag will cause struck enemies to be unconditionally placed into their pain state. (Or Pain.Dagger if they have it defined.)

CPF_PULLIN — Pull in: Successful hit pulls the player forward like Doom's Chainsaw or Heretic's Gauntlets. It's important to note that a successful melee attack, regardless of this flag, will always turn the player's facing angle towards their struck target but the player's velocity will only be affected if the flag is set.

CPF_NORANDOMPUFFZ — The random z offset given to the puff when spawned is disabled.

CPF_NOTURN — No turn: the player's facing angle is not adjusted towards their struck target on successful hits.

CPF_STEALARMOR — When lifesteal is used, the stolen health is converted to armor points, repairing the player's armor, instead of healing the player like normal.

pufftype: The puff actor to use when striking a wall or non-bleeding actor. The default is BulletPuff.

range: The range of the attack. The default value is 64 (actually the value is 0, but since a range of 0 would be a worthless attack A_CustomPunch will interpret a range of 0 as 64.)

lifesteal: If positive, is the value used as a factor for giving back the inflicted damage as hit points to the actor using the calling weapon. The value works as a multiplier, so 1.0 will give the player exactly as much health as they dealt damage; higher values will give more, lower values will give a smaller fraction.

lifestealmax: The limit of how much the player heals when stealing health from the victim. This works for when stealing health for health and health for armor. Positive values set an explicit limit. If a value of zero is passed, the player is healed up to their maximum health when stealing for health, and up to the armorbonustype item's Armor.MaxSaveAmount (see below) when stealing for armor. Default value is 0.

armorbonustype: The armor bonus item to use for repairing armor when life-stealing. This must be an item derived from BasicArmorBonus. When stealing for armor, the item's Armor.SaveAmount property is taken into account; the life stolen value will be multiplied by that property's value. If this is not specified, ArmorBonus will be used as the default item.

meleesound: If the attack hits anything, the weapon will play this sound effect. If unspecified, the weapon's AttackSound property is used instead.

misssound: The sound to play if the weapon does not hit anything tangible.

Please note that the Berserk powerup will not work with A_CustomPunch. However, you can emulate the effect using A_JumpIfInventory and A_SelectWeapon (see the third example, below).

Examples

Fire:

KNIF B 4

KNIF C 4 A_CustomPunch(20, FALSE, 0) // 20 * random(1, 8) since norandom is false, uses no ammo, will
// spawn a BulletPuff, has a range of 64, and doesn't steal life.

KNIF D 5

```

KNIF C 4
KNIF B 5 A_ReFire
Goto Ready
Fire:
SWRD B 4
SWRD C 4 A_CustomPunch(random(4, 8) * 10, TRUE, 0, "SwordPuff", 96) // Uses a custom damage
formula, spawns
// a custom puff, and has a longer range of 96.
SWRD D 4
SWRD C 6
SWRD B 6 A_ReFire
Goto Ready

```

As mentioned above, A_CustomPunch does not work with the berserk pack "out of the box". The following example provides a workaround for this using Doom's fist as an example:

```

ACTOR CustomFist : Fist
{
  States
  {
    Fire:
    PUNG B 4
    TNT1 A 0 A_JumpIfInventory("PowerStrength", 1, "Berserked")
    Normal:
    PUNG C 4 A_CustomPunch(2 * random(1, 10), TRUE)
    Goto FireEnd
    Berserked:
    PUNG C 4 A_CustomPunch(20 * random(1, 10), TRUE)
    FireEnd:
    PUNG D 5
    PUNG C 4
    PUNG B 5 A_ReFire
    Goto Ready
  }
}

```

By checking for the presence of the PowerStrength powerup in the player's inventory, which is given by the berserk pack, the damage of the punch can be "adjusted".

A_FireAssaultGun

A_FireAssaultGun

(no parameters)

This is the attacking function of Strife's assault gun. It plays the sound "weapons/assaultgun" and then fires bullets whose accuracy is dependent on how many accuracy upgrades the player has received.

Examples

Taked from assault gun fire state:

Fire:

RIFF AB 3 A_FireAssaultGun

RIFG D 3 A_FireAssaultGun

RIFG C 0 A_ReFire

RIFG B 2 A_Light0

Goto Ready

A_FireBFG

DoomWiki.org

For more information on this article, visit the [A_FireBFG](#) page on the Doom Wiki.

(no parameters)

Usage

Fires a BFGBall from the current weapon.

Autoaim is disabled by default through the "Allow BFG aiming" DMFlag. This can be circumvented by spawning it manually:

```
A_FireProjectile("BFGBall", 0, 1)
```

Examples

This is taken directly from the BFG9000.

Fire:

```
BFGG A 20 A_BFGSound
```

```
BFGG B 10 A_GunFlash
```

```
BFGG B 10 A_FireBFG
```

```
BFGG B 20 A_ReFire
```

```
Goto Ready
```

A_FireBullets

A_FireBullets (angle spread_horz, angle spread_vert, int numbullets, int damage [, string pufftype [, int flags [, float range [, string missile [, float spawnheight [, float spawnofs_xy]]]]]])

Usage

Defines a custom hitscan attack for weapons. You have to specify the horizontal and vertical spread, the amount of bullets and the damage per bullet.

If numbullets is 1 and this is the first bullet fired in the Fire (not Hold) sequence, the bullet is fired with perfect accuracy, ignoring the specified spread. If numbullets is 0, however, the bullet is always fired with perfect accuracy. Setting numbullets to a negative value removes this effect while firing a single bullet only.

When successfully called, the function plays the weapon's AttackSound sound, if present, on the weapon channel (CHAN_WEAPON) with normal attenuation.

Note: This function utilizes Player.AttackZOffset for positioning the origin of the bullet. Use this to adjust the height where bullets come from on the player for accuracy.

Parameters

spread_horz - The random spread going right and left.

spread_vert - The random spread going up and down.

numbullets - The number of bullets this function spawns.

damage - The amount of damage to deal per bullet. Damage is multiplied by random(1,3). Damagetypes also have an influence on this.

pufftype - The actor to spawn at the puff's position when it hits. See BulletPuff for more details on spawning conditions.

flags - Can be combined with the | symbol.

FBF_EXPLICITANGLE: If set, the horizontal and vertical spread are used as explicitly stated, instead of being used as a range for random spread.

FBF_USEAMMO: If set, the attack uses ammo. (Set by default if the flags parameter is not specified - if not using any other flags, pass 0 to turn this off)

FBF_NOFLASH: If set, the attack does not cause a weapon flash.

FBF_NOPITCH: If set, the vertical angle is not adjusted to aim at the target.

FBF_NORANDOM: If set, the damage is not multiplied by 1d3.

FBF_NORANDOMPUFFZ: If set, the random z offset given to the puff when spawned is disabled.

FBF_PUFFTARGET: Only works when missile is used. Sets the puff as the missile's target.

FBF_PUFFMASTER: Only works when missile is used. Sets the puff as the missile's master.

FBF_PUFFTRACER: Only works when missile is used. Sets the puff as the missile's tracer.

NOTE: The pointer flags will not work if the puff does not exist, i.e. spawning Blood instead of itself.

range - The maximum distance the bullets can hit something. Default is 8192.

missile: The actor projectile to spawn. This actor faces the bullet puff and travels directly towards it. Spawning a missile does not consume extra ammo. Default is none.

spawnheight: Offsets how high up from the base of the actor missile spawns. Default is 32.

spawnofs_xy: Offsets how far to the calling actor's right to spawn missile from (assuming one is viewing the actor from behind). Negative values spawn it to the left. Default is 0.

Examples

Fire:

```
TRIF A 5 Bright A_FireBullets(0, 0, 1, 45, "RiflePuff", FBF_USEAMMO|FBF_NORANDOM)
```

```
TRIF B 5 Bright
```

```
TRIG A 10
```

```
TRIG B 0 A_ReFire
```

```
Goto Ready
```

A_FireCGun

(no parameters)

Usage

Does the standard Chaingun attack. It plays the sound "weapons/chngun", runs the flash state, and fires one bullet similar to A_FirePistol.

If vertical bullet spread for weapons is enabled, the function applies vertical spread in addition to the horizontal one.

Examples

This code is equivalent to A_FireCGun.

Fire:

```
CHGG A 0 A_PlaySound("weapons/chngun", CHAN_WEAPON)
CHGG A 0 A_GunFlash
CHGG A 4 A_FireBullets(5.6, 0, 1, 5, "BulletPuff")
CHGG B 0 A_PlaySound("weapons/chngun", CHAN_WEAPON)
CHGG B 0 A_GunFlash("Flash2")
CHGG B 4 A_FireBullets(5.6, 0, 1, 5, "BulletPuff")
CHGG B 0 A_ReFire
Goto Ready
```

Flash:

```
CHGF A 4 Bright A_Light1
Goto LightDone
```

Flash2:

```
CHGF B 4 Bright A_Light2
Goto LightDone
```

A_FireMissile

(no parameters)

Fires a Rocket. This is equivalent to doing the following:

MISG B 12 A_FireProjectile ("Rocket")

Examples

Doom's rocket launcher firing sequence:

Fire:

MISG B 8 A_GunFlash

MISG B 12 A_FireMissile

MISG B 0 A_Refire

Goto Ready

A_FireOldBFG

(no parameters)

Performs the BFG9000's attack from the Doom press-release beta version. Each call of the function fires two plasma projectiles, one is green and the other is red, with a slight random spread. The BFG calls this function 40 times costing 40 points of ammunition with 1-tic interval between each call.

Examples

This simulates said Doom version's BFG9000's firing sequence.

```

Fire:
  BFGG A 10 A_BFGSound
  BFGG BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB 1 A_FireOldBFG
  BFGG B 0 A_Light0
  BFGG B 20 A_ReFire
  Goto Ready

```


A_FirePistol

(no parameters)

Usage

Performs Doom's pistol attack, firing one pellet dealing $5 * 1d3$ damage. This is a shortcut to calling A_FireBullets, A_PlaySound and A_GunFlash with certain predetermined hardcoded parameters.

If vertical bullet spread for weapons is enabled, the function applies vertical spread in addition to the horizontal one.

Examples

This code is the equivalent of calling A_FirePistol.

```
A_FireBullets (5.6, 0, 1, 5, "BulletPuff")  
A_PlaySound("weapons/pistol", CHAN_WEAPON)  
A_GunFlash
```

A_FirePlasma

(no parameters)

Fires a PlasmaBall, activates the Flash state sequence and sometimes randomly offsets the Flash sequence by 1 frame.

Examples

Fire state from Plasmagun:

Fire:

PLSG A 3 A_FirePlasma

PLSG B 20 A_ReFire

Goto Ready

A_FireProjectile

Actor A_FireProjectile (class<Actor> missiletype [, double angle [, bool useammo [, double spawnofs_xy [, double spawnheight [, int flags [, double pitch]]]]]])

Actor, Actor A_FireProjectile (class<Actor> missiletype [, double angle [, bool useammo [, double spawnofs_xy [, double spawnheight [, int flags [, double pitch]]]]]]) (New from 4.10.0)

Usage

Fires a projectile from a Weapon or a CustomInventory. Optionally you can specify an angle and a spawn offset.

This serves as a replacement for A_FireCustomMissile which has a pitch miscalculation in it.

If used in a weapon, useammo specifies whether this attack uses ammo or not. This parameter is irrelevant if used in a custom inventory.

The following flags can optionally be used. Multiple flags can be combined by separating them with the pipe character "|".

FPF_AIMATANGLE â€” Alternate autoaim behavior, relevant when playing with autoaim. Autoaiming is based off of the projectile's trajectory instead of the player's aim. If this flag is set, the engine looks at the horizontal line of fire projected for the projectile, independent of the player's aim. If the horizontal line of fire cuts close enough to a valid target (again ignoring vertical aim), that projectile will fire directly towards that target. If this flag is not set, the engine looks at the player's horizontal aim. If the horizontal line of fire cuts close enough to a valid target (ignoring vertical aim), the resulting projectile will fire with the vertical angle adjusted to aim directly at the target.

FPF_TRANSFERTRANSLATION â€” Transfer Translation. The projectile fired uses the same translation as the actor that fired it. In most cases, this will be the player.

FPF_NOAUTOAIM â€” Disables autoaim for this attack.

pitch becomes relevant with autoaim off. It adjusts the player's vertical aim by the given angle value, like the angle parameter affects the horizontal aim. A positive value makes it fire the missile lower than the player's aim, a negative value makes it fire higher, in contrast to how A_FireCustomMissile inverts the effect of the pitch parameter.

Return values

A_FireProjectile returns two Actor pointers, both pointing to the projectile that was fired. The only difference between them Both of those are pointers to the projectile that was fired, but they're created under different conditions:

The first pointer is only created if the projectile managed to enter its Spawn state sequence. If the projectile is fired at an enemy, object or a wall at a point-blank range, it'll instead immediately enter its Death (Crash, XDeath, etc.) state sequence, completely skipping Spawn, and this pointer will be null.

The second pointer (New from 4.10.0) is always created, regardless of the state sequence the projectile entered.

As such, if there's a need to modify some properties of the projectile, the second pointer should be used, because the first one is simply not guaranteed to exist. (The second pointer should still be null-checked for safety, but normally it will never be null).

Example:

```
// Create two pointers:
Actor p1, p2;
// Cast both of them to the projectile:
[p1, p2] = A_FireProjectile("Rocket");
// Modify the resulting projectile's damage:
if (p2)
{
    p2.SetDamage(80);
}
```

```
}
```

Examples

Fire:

```
TRIF A 5 Bright A_FireProjectile("RifleBullet",0,1,8,8,0)
```

```
TRIF B 5 Bright
```

```
TRIG A 10
```

```
TRIG B 0 A_Refire
```

```
Goto Ready
```

A_FireShotgun

(no parameters)

Performs Doom's shotgun attack, firing seven pellets each dealing $5 * 1d3$ damage. This is a shortcut for calling A_FireBullets, A_PlaySound and A_GunFlash with predetermined hard-coded parameters.

This code is the equivalent of calling A_FireShotgun for 7 tics on the A frame of the SHTG sprite:

```
SHTG A 0 A_FireBullets (5.6, 0, 7, 5, "BulletPuff")  
SHTG A 0 A_PlaySound ("weapons/shotgf", CHAN_WEAPON)  
SHTG A 7 A_GunFlash
```

If vertical bullet spread for weapons is enabled, the function applies vertical spread in addition to the horizontal one.

A_FireShotgun2

(no parameters)

Performs Doom's super shotgun attack, firing 20 pellets dealing $5 * 1d3$ damage each. This function is roughly equivalent to A_FireBullets, A_PlaySound and A_GunFlash with certain predetermined hardcoded parameters:

```
SHT2 A 0 A_FireBullets (11.2, 7.1, 20, 5, "BulletPuff")
SHT2 A 0 A_PlaySound ("weapons/sshotf", CHAN_WEAPON)
SHT2 A 7 A_GunFlash
```

A_FireShotgun2 internally uses a somewhat different pitch calculation from A_FireBullets, which results in a slightly different spread pattern.

Note: The ZScript definition below is for reference and may be different in the current version of GZDoom. The most up-to-date version of this code can be found on GZDoom GitHub.

action void A_FireShotgun2()

```
{
    if (player == null)
    {
        return;
    }

    A_StartSound ("weapons/sshotf", CHAN_WEAPON);
    Weapon weap = player.ReadyWeapon;
    if (weap != null && invoker == weap && stateinfo != null && stateinfo.mStateType == STATE_Psprite)
    {
        if (!weap.DepleteAmmo (weap.bAltFire, true, 2))
            return;

        player.SetPsprite(PSP_FLASH, weap.FindState('Flash'), true);
    }
    player.mo.PlayAttacking2 ();

    double pitch = BulletSlope ();

    for (int i = 0 ; i < 20 ; i++)
    {
        int damage = 5 * random[FireSG2](1, 3);
        double ang = angle + Random2[FireSG2]() * (11.25 / 256);

        // Doom adjusts the bullet slope by shifting a random number [-255,255]
        // left 5 places. At 2048 units away, this means the vertical position
        // of the shot can deviate as much as 255 units from nominal. So using
        // some simple trigonometry, that means the vertical angle of the shot
        // can deviate by as many as ~7.097 degrees.

        LineAttack (ang, PLAYERMISSILERANGE, pitch + Random2[FireSG2]() * (7.097 / 256), damage,
            'Hitscan', "BulletPuff");
    }
}
```

A_Punch

(no parameters)

Doom's standard fist attack. This checks whether the player has the Berserk pack by looking for the presence of a PowerStrength item in the inventory. Damage is 2x 1d10, multiplied again by 10 when berserk. If the attack hits a damageable actor, the *fist player sound is played, on the weapon (CHAN_WEAPON) channel.

The range of the attack, which is 64, is determined by adding 20 to the calling actor's MeleeRange. Although the actor property could be used to alter the range, it is best to instead use the more flexible A_CustomPunch function to give an attack its own range. This feature exists only for DoomWikiLogoIcon.pngMBF21 support.

You can implement A_Punch using A_CustomPunch. You can find an example of how to do this on that action's page (near the bottom).

Examples

This example is taken from Doom's Fist weapon.

Fire:

```
PUNG B 4
PUNG C 4 A_Punch
PUNG D 5
PUNG C 4
PUNG B 5 A_ReFire
goto Ready
```

A_RailAttack

```
void A_RailAttack (int damage [, int spawnofs_xy [, bool useammo [, color color1 [, color color2 [, int flags [,
double maxdiff [, class<Actor> pufftype [, double spread_xy [, double spread_z [, double range [, int
duration [, double sparsity [, double driftspeed [, class<Actor> spawnclass [, double spawnofs_z [, int
spiraloffset [, int limit]]]]]]]]]]]]))
```

Usage

Fires a rail attack. Only works on weapons.

Parameters

damage: The damage to inflict on each target that is hit; this can be a fixed value or an expression.

spawnofs_xy: The horizontal distance from the center of the screen at which the railgun tracer should originate. Default is 0.

useammo: Whether or not ammo should be used up when firing. Default is true.

color1: The color of the particles that form the spiral "ring" of the beam. This can be in the form of a RRGGBB string or a string holding a color defined in the X11R6RGB lump. If the string is invalid, the particles will be black. None (without quotes) in DECORATE or "" in ZScript will make the ring invisible. A value of 0, which is treated specially, draws the the particles in one of four shades of blue, picked randomly. Default is 0.

color2: The color of the particles that form the central "core" of the beam. This can be in the form of a RRGGBB string or a string holding a color defined in the X11R6RGB lump. If the string is invalid, the particles will be black. None (without quotes) in DECORATE or "" in ZScript will make the core invisible. A value of 0, which is treated specially, draws the particles in one of three shades of gray, picked randomly. Default is 0.

flags: The following flags can be combined by using the | character between the constant names:

RGF_SILENT — Silent: The railgun will not play an attack sound when firing.

RGF_NOPIERCING — Not piercing: The railgun will stop at the first enemy hit, rather than passing through.

RGF_EXPLICITANGLE — Explicit angle: The spread parameters are taken as explicit angles rather than maximum random amplitude.

RGF_FULLBRIGHT — Full bright: Rail particles will be rendered at maximum brightness, ignoring sector lighting.

RGF_CENTERZ — Attack from center: Z offset originates from half of the player's height (adjusted when crouched), instead of using the `Player.AttackZOffset` property.

RGF_NORANDOMPUFFZ — No random puff Z: Disables the random z-offset of spawned puffs.

maxdiff: This is used to make the rail more jagged, or lightning-like, with higher numbers. Default is 0 (straight).

pufftype: The puff actor to use. By default, the puff will only spawn in rare circumstances (e.g. when hitting a dormant monster) unless the puff actor has the ALWAYSPUFF flag set. Even if not shown, the selected puff will still be used for applying custom damagetypes and other properties. Puffs with the ALWAYSPUFF flag spawn on floors and ceilings. Default is "BulletPuff".

`spread_xy`: Maximum angle of random horizontal spread. Default is 0.

spread_z: Maximum angle of random vertical spread. Default is 0.

range: Maximum distance (in map units, as fixed-point) the rail shot will travel before vanishing. Default is 0, which uses the default value of 8192 as the range.

duration: Lifetime of spawned particles, in tics. Default is 0, which uses the default value of 35 as the duration.

sparsity: Distance between individual particles. Implemented as a multiplier. Default is 1.0.

driftspeed: Speed at which particles "drift" away from their initial spawn point. Implemented as a multiplier. Default is 1.0.

spawnclass: Actor to spawn in place of trail particles. If non-null, the specified actor will be spaced sparsity units apart instead of the usual trail. It will also inherit the pitch of the shooter and track the owner, allowing for explosive trails to not hurt the owner. Particle-specific properties such as duration, driftspeed, and rail color are ignored in such a case. Default is "None".

spawnofs_z: The vertical distance from the center of the screen at which the railgun tracer should originate. Negative values shift the tracer down, positive values shift it up. Default is 0.

spiraloffset: the angle from which the outer ring starts spiraling. Default is 270.

limit: Sets the maximum number of actors to pierce through, if they are applicable for damaging. Default is 0 (no limit is set).

Examples

This is a generic railgun for the player. This will work copy and pasted, but you'll need graphics. Notice that this example does not have a gunflash, you can still add one though

ACTOR PlayerRailgun : Weapon

```
{
  Radius 24
  Height 16
  Obituary "%o got railgunned by %k."
  Weapon.SelectionOrder 100
  Weapon.SlotNumber 6
  Weapon.Kickback 500
  Weapon.AmmoType "Clip"
  Weapon.AmmoUse 1
  Weapon.AmmoGive 30
  AttackSound "weapons/rbeam"
  +EXTREMEDEATH
  States
  {
    Spawn:
      RLGN A -1
      Stop
    Ready:
      RLGN B 1 A_WeaponReady
      Loop
    Deselect:
      RLGN B 1 A_Lower
      Loop
    Select:
      RLGN B 1 A_Raise
      Loop
    Fire:
      RLGN C 4
      RLGN D 0 A_RailAttack(2, 0, 1, "ffffa0", "ffffa0", 0, 0, "none")
      RLGN E 8 Bright
      RLGN F 4
      TNT1 A 0 A_ReFire
      Goto Ready
  }
}
```

A_Saw

```
A_Saw [(string fullsound [, string hitsound [, int damage [, string pufftype [, int flags [, float range [, float spread_xy [, float spread_z [, float lifesteal [, int lifestealmax [, string armorbonustype]]]]]]]]]]]]]]]
```

Usage

Doom's chainsaw attack for weapons. For monsters, use A_M_Saw.

Parameters

fullsound: The sound that plays if the attack doesn't hit anything. Defaults to "weapons/sawfull".

hitsound: The sound that plays if the weapon hits a target. Defaults to "weapons/sawhit".

damage: The amount of damage to deal, with the following calculation:

```
if (damage == 0) damage = 2;
```

```
damage *= (random() % 10 + 1);
```

For example, if damage is 5, the damage dealt will be between 5 and 50. Note that it is not possible to have this function deal 0 damage, since 0 means "use the default value" which is 2.

pufftype: The puff to spawn if the attack hits a wall or invulnerable actor. Defaults to "BulletPuff".

flags: The following flags can be combined by using the | character between the constant names:

SF_NORANDOM — No random: Disables damage randomization.

SF_RANDOMLIGHTMISS — Random light on miss: Adds a 25% chance to toggle lighting if no target is hit.

SF_RANDOMLIGHTHIT — Random light on hit: Randomly changes light on hit, like A_GauntletAttack (25% 0, 37.5% 1, 37.5% 2).

SF_RANDOMLIGHTBOTH — Random light on both: The two above flags combined, works like A_GauntletAttack.

SF_NOUSEAMMOMISS — No ammo use on miss: Do not use ammo if nothing hit.

SF_NOUSEAMMO — No ammo use: Do not use ammo.

SF_NOPULLIN — No pull in: the player is not pulled towards their struck target on successful hits.

SF_NOTURN — No turn: the player's facing angle is not adjusted towards their struck target on successful hits.

SF_STEALARMOR — When lifesteal is used, the stolen health is converted to armor points, repairing the player's armor, instead of healing the player like normal.

SF_NORANDOMPUFFZ — The random z offset given to the puff when spawned is disabled.

range: The maximum range of the attack. If this is 0, the range is determined by adding 20 to the calling actor's MeleeRange, which is 44 by default. Although the actor property could be used to alter the range, it is best to instead set the range directly through this parameter. This feature exists only for DoomWikiLogoIcon.pngMBF21 support. Default is 0.

spread_xy: Maximum angle of random horizontal spread. Defaults to 2.8125.

spread_z: Maximum angle of random vertical spread. Defaults to 0.

lifesteal: If positive, this value is used as a factor for giving back the inflicted damage as hit points to the actor using the calling weapon.

lifestealmax: The limit of how much the player heals when stealing health from the victim. This works for when stealing health for health and health for armor. Positive values set an explicit limit. If a value of zero is passed, the player is healed up to their maximum health when stealing for health, and up to the **armorbonustype** item's **Armor.MaxSaveAmount** (see below) when stealing for armor. Default value is 0.

armorbonustype: The armor bonus item to use for repairing armor when life-stealing. This must be an item derived from `BasicArmorBonus`. When stealing for armor, the item's `Armor.SaveAmount` property is taken into account; the life stolen value will be multiplied by that property's value. If this is not specified, `ArmorBonus` will be used as the default item.

Examples

This example is taken from Doom's chainsaw weapon.

Fire:

SAWG AB 4 A Saw

SAWG B 0 A ReFire

Goto Ready

A_CheckForReload

state A_CheckForReload (int counter, str state[, bool dontincrement])

Jumps to the specified state if the function has not been called counter times. If dontincrement is true (default is false), then the reload counter isn't increased by a call to this function. This action is part of a generalized version of Skulltag's A_CheckRailReload and may not be particularly useful for creating complex reloading systems in weapons. Here's an example from the rail guns fire state using the function.

Examples

Fire:

```
RLGG E 12 A_FireRailgun
```

```
RLGG F 6 A_CheckForReload(4, "Reloaded")
```

```
RLGG GHIJK 6
```

```
RLGG L 6 A_ResetReloadCounter
```

A_CheckReload

(no parameters)

This function checks whether the player still has enough ammunition for another attack with the current weapon. If not, it starts a weapon change.

Examples

This code is taken from Doom 2's SuperShotgun. A_CheckReload is called to check to see if there are any more shells left before starting the "reload" animation of the super shotgun.

Fire:

```
SHT2 A 3
SHT2 A 7 A_FireShotgun2
SHT2 B 7
SHT2 C 7 A_CheckReload
SHT2 D 7 A_OpenShotgun2
SHT2 E 7
SHT2 F 7 A_LoadShotgun2
SHT2 G 6
SHT2 H 6 A_CloseShotgun2
SHT2 A 5 A_ReFire
Goto Ready
```

A_ClearOverlays

int A_ClearOverlays [(int start, int stop, bool safety)]

Usage

Removes a range of layers between [start,stop]. If no value or 0 is used for both, it will clear all but the hardcoded layers. Returns the number of layers cleared that were active between the range.

The following is a list of internal layers used by engine (These are NOT flags or constants for use!):

PSP_STRIFEHANDS (-1) - Used by A_ItBurnsItBurns in Strife.

PSP_WEAPON (1) - Used by the actual weapon.

PSP_FLASH (1000) - Used by the Flash state (see A_GunFlash).

PSP_TARGETCENTER (2147483645)

PSP_TARGETLEFT (2147483646)

PSP_TARGETRIGHT (2147483647) - The absolute highest layer number.

It is recommended that all new overlays should use layers 2 on up, or -2 on down to avoid any potential problems with compatibility.

Parameters

start - The low end of the range to clear through. If 0 including stop, clears all current overlays except hardcoded ones.

stop - The high end of the range to clear through. If 0 including start, clears all current overlays except hardcoded ones.

safety - Enabled by default, the hardcoded layers above are protected from wiping as this may cause problems with switching away from weapons, as there is no active layer to allow switching with. Use extreme caution when disabling.

A_ClearReFire

(no parameters)

Usage

Normally, when A_ReFire is encountered, ZDoom jumps to the Fire/AltFire state unless a specific state is defined, or a Hold/AltHold state is found, provided that the appropriate fire button is being pressed. For weapons where it may be desirable to return to the Fire/AltFire state, and the player has been holding the appropriate fire button down continually, the A_ClearReFire function can be used to send the weapon back to the Fire/AltFire state.

A_ReFire can take a state parameter, allowing it to be much more flexible.

Examples

This is a weapon that uses A_ClearReFire to continue the chaingun animation.

Actor NewChaingun : Chaingun

```
{
    States
    {
        Fire:
            CHGG A 4 A_FireCGun
            CHGG A 0 A_ReFire
            Goto Ready
        Hold:
            CHGG B 4 A_FireCGun
            CHGG B 0 A_ClearReFire //A_ReFire("Fire")
            Goto Ready
    }
}
```

A_GunFlash

void A_GunFlash [(statelabel flashlabel [, int flags])]

Usage

This function can be called in a weapon's Fire or AltFire state sequence. It sets the player sprite to a version where the player is illuminated by his gunfire and it starts the appropriate flash state sequence. These flash state sequences are run at the same time as the normal weapon animation.

It is ideal for the flash to last as long as the weapon's firing state, or at least when the gun is lit. You can make your own muzzleflash sprites which, as long as the offsets are correct, will be overlayed when the gun is fired, making only that part of the gun visible when fires in pitch darkness. If this isn't an option, you can simply use TNT1A0. The example below was taken from a weapon with muzzle flash sprites.

Parameters

flashlabel: The state sequence to enter. If this is null(ZScript only)/0 or "" (empty string)(DECORATE only), the AltFlash state sequence, if it exists, is entered if the function is called from the AltFire state sequence, otherwise it is the Flash state sequence that is entered. Default is null.

flags: The following flags can be combined by using the | character between the constant names:

GFF_NOEXTCHANGE " No external change: The player sprite will not be affected.

Default is 0.

Examples

Fire:

```
DEAG C 0 A_StartSound("weapons/eagle", CHAN_WEAPON)
DEAG C 0 A_GunFlash // This should be run the same time as the gun fires.
DEAG C 3 A_FireBullets(5, 7, 1, 50,"BulletPuff")
Goto Ready
```

Flash:

```
DEFL A 1 Bright A_Light1
DEFL A 2 Bright A_Light2
DEFL A 0 A_Light0
Stop
```

A_Light

A_Light (int intensity)

A customizable version of the A_Light# functions. intensity is a number between -20 and 20, 0 being standard brightness, while 1 is equivalent to A_Light1 and 2 to A_Light2. 3â€“20 are each a step brighter. Negative values for intensity, down to -20, can be used, and it will instead darken the scene.

Examples

This BFG9000 fires dimming surrounding area:

Actor BFG4ZDWiki: BFG9000 replaces BFG9000

```
{
Weapon.SlotNumber 7
States
{
Flash:
    BFGF A 4 Bright A_Light( -1 )
    BFGF A 4 Bright A_Light( -2 )
    BFGF A 3 Bright A_Light( -3 )
    BFGF B 1 Bright A_Light( -3 )
    BFGF B 2 Bright A_Light( -4 )
    BFGF B 3 Bright A_Light( -5 )
    Goto LightDone // A_Light0
}
}
```


A_Light0

A_Light0

A_Light1

A_Light2

A_LightInverse

(no parameters)

These functions set the 'weapon light'. This is a slight illumination of the entire level.

A_Light0, A_Light1 and A_Light2 set this to values 0, 1 and 2 respectively. 0 means off and 2 means bright.

A_LightInverse makes the screen use an inverted greyscale mode. Note that this is not specifically Doom's invulnerability effect, but rather the flash from firing Strife's Sigil. This should not be confused with the invulnerability palette, as other Doom engine games' invulnerability effects (specifically Heretic's gold palette) are not shown when using this function.

Example

This is the Flash state from Strife's Sigil .

Flash:

SIGF A 4 Bright A_Light2

SIGF B 6 Bright A_LightInverse

SIGF C 4 Bright A_Light1

SIGF C 0 Bright A_Light0

Stop

A_Lower

void A_Lower [(int lowerspeed)]

Usage

This function must be called in a weapon's Deselect state sequence. It should not be used anywhere else. It is responsible for lowering the weapon until it is off the screen and for selecting another weapon when it is done.

Each time the function is called, the weapon moves further down the screen until it has reached the fully lowered position. You can therefore make a weapon lower faster than the default by calling the function more than once within the same tic, or by passing the desired speed directly.

Error.gif

Warning: Do not call this function repeatedly in the same state inside another function, anonymous or otherwise. All calls to A_Lower are executed fully inside the other function, even if the weapon has reached its ready position, resulting in unwanted side effects.

Parameters

lowerspeed: how much the weapon is lowered by. Default is 6.

Examples

Here is an example of a basic Deselect state that makes use of this function to move the weapon down towards the lowered position and then select a new weapon:

Deselect:

```
SHTG A 1 A_Lower
```

```
Loop
```

This will lower the weapon down off the screen when a new weapon is selected. Once it has reached the fully lowered position, A_Lower will change to the new weapon that was selected.

This weapon lowers from view twice as fast as normal:

Deselect:

```
FAST A 1 A_Lower(12)
```

```
Loop
```

A_Overlay

bool A_Overlay (int layer[, state start [, bool nooverride]])

Usage

This function is a considerably expanded version of A_GunFlash, allowing users to set up multiple frames of animation for complex weaponry (dual wielding) and/or HUD elements. This can be called by more than just the weapon; a player or CustomInventory item can also call this function.

When a weapon calls this, all overlays will disappear when switching away. When called by the player or inventory items, overlays will persist until the items disappear.

To specify different behaviors for overlays, use A_OverlayFlags.

The following is a list of internal layers used by engine (These are NOT flags or constants for use!):

PSP_STRIFEHANDS (-1) - Used by A_ItBurnsItBurns in Strife.

PSP_WEAPON (1) - Used by the actual weapon.

PSP_FLASH (1000) - Used by the Flash state (see A_GunFlash).

It is recommended that all new overlays should use layers 2 on up, or -2 on down to avoid any potential problems with compatibility.

Parameters

layer - Indicates the draw hierarchy for the layer. Higher numbered layers will be drawn over lower numbers.

start - The name of the state to draw.

nooverride - If specified, the function will not create an overlay on an already active layer and return false. Otherwise, returns true. This can be useful for if/else statements.

A_OverlayAlpha

A_OverlayAlpha (int layer, double alph)

Usage

Sets the alpha of the overlay specified. Requires A_OverlayFlags' PSPF_ALPHA set to true to have any effect.

Parameters

layer - The layer to modify.

alph - The alpha for the layer to have.

A_OverlayFlags

A_OverlayFlags (int layer, int flags, bool set)

Usage

Modifies an overlay's behavior, adding or subtracting flags.

Parameters

layer - The layer to modify. This is not limited to overlays; PSP_WEAPON aka the main layer can also have its flags changed.

flags - Flags can be combined using '|' (without ' '):

PSPF_ADDWEAPON - The overlay will follow the weapon's offsets. Enabled by default. When enabled, A_OverlayOffset's X and Y values will be relative to the main weapon layer's offsets. If disabled, allows setting the overlay's offsets independently. (Note, if you disable it, your overlay's sprite will jump 32 units down because it will no longer be synced with the main layer's position, which is (0, 32) by default, while overlays use (0, 0) by default.)

PSPF_ADDBOB - The overlay will follow the weapon's bobbing. Enabled by default.

PSPF_POWDOUBLE - The overlay will have its speed doubled if a PowerDoubleFiringSpeed is active in the player's inventory.

PSPF_CVARFAST - The overlay will respect sv_fastweapons if enabled.

PSPF_FLIP - Flips the overlay on the X axis. Affects the visuals but not the position.

PSPF_MIRROR - Flips the offset of the overlay. Affects the position but not the visuals. Use in conjunction with PSPF_FLIP to alter the sprite's handedness. Does not imply PSPF_FLIP by itself.

PSPF_ALPHA - Enables alpha changing. Note, many renderstyles enforce their own alpha values, and this flag is not able to override it.

PSPF_FORCEALPHA - Forcefully enables alpha changing regardless of the overlay's renderstyle. PSPF_ALPHA doesn't have to be enabled for this to work.

PSPF_RENDERSTYLE - Enables renderstyle changing via A_OverlayRenderstyle.

PSPF_FORCESTYLE - Some powerup effects like the Blursphere will set all overlays to the same as the players while in effect, applying its renderstyle on the player as well. Enabling this overrides the effect for the specified layer.

PSPF_PLAYERTRANSLATED - Translates the overlay according to the same rules as the player's sprite. It uses the Player.ColorRange property for translation (which by default covers the green hues) and translates it according to player's settings in GZDoom.

PSPF_PIVOTPERCENT - When enabled, the A_OverlayPivot's XY position is based on a scalar between 0.0 (X: Left, Y: Top) to 1.0 (X: Right, Y: Bottom) of the graphic. Otherwise, the coordinates are based on coordinate offsets in pixels.

PSPF_INTERPOLATE - Enables interpolation constantly for the overlay. For backwards compatibility however, A_OverlayOffset called in any tic without the add and interpolate flags will temporarily disable interpolation for that tic.

set - If true, enables the flags on the layer. If false, disables them.

A_OverlayOffset

A_OverlayOffset [(int layer[, float x[, float y[, int flags]]]])]

Usage

Works just like A_WeaponOffset with the ability to adjust other layers, and can be called by weapons, players and inventory items. Offsets a weapon by the defined amount of x and y similar to the Offset frame keyword. This function has the following differences from the aforementioned Offset ability:

Offsets are floats, not integers.

Offsets are not lost when firing a weapon.

Offsets stack on top of weapon bobbing and behave independently, allowing A_WeaponReady calls to it with the WRF_NOBOB flag.

Specifying the default coordinates does not serve as a preservative. They are absolute.

Offsets can go beyond screen limitations.

Because this is an action function itself, Decorate expressions can be used.

NOTE: If an overlay has PSPF_ADDWEAPON, it will inherit the weapon's offsets, which default to (0,32) if none is specified. This should be kept in mind when offsetting the overlay. Simply set the X and Y offsets to 0 to have the overlay follow the weapon's position. This does not apply to A_WeaponOffset as that strictly targets the weapon layer itself, never overlays.

Parameters

layer - The layer to adjust.

x - Adjusts horizontal position based on positive (right) or negative (left) numbers. Default is 0.

y - Adjusts vertical position. Default is 32. Larger values lower the weapon, and vice versa.

flags - Flags can be combined with the | separator.

WOF_KEEPPX - x parameter will not be used.

WOF_KEEPPY - y parameter will not be used.

WOF_ADD - Adds the current x and y parameters to the current offsets instead of overriding them. Also implies interpolation (see below).

WOF_INTERPOLATE - Interpolates the offset between tics, smoothing out the animation.

A_OverlayPivot

A_OverlayPivot (int layer, double wx = 0.5, double wy = 0.5, int flags = 0)

Usage

Sets the position of the pivot on the overlay. This is used to set a position that will act as the center of rotation for A_OverlayRotate and A_OverlayScale. These offsets are applied on top of A_OverlayPivotAlign's settings.

This function's offsets are affected by A_OverlayFlags's PSPF_PIVOTPERCENT flag, which is set to true by default.

Parameters

layer - The layer to modify.

wx - Positive numbers offset to the right. Default is 0.5 (which is center if PSPF_PIVOTPERCENT is applied).

wy - Same as `wx` but goes down if positive, up if negative instead.

flags - Flags can be combined using '|' (without ' '):

WOF_RELATIVE - Takes into account the current rotation of the overlay.

WOF_ADD - The input adds instead of replaces the current offset, and implies WOF_INTERPOLATE.

WOF_KEEPIX - The 'wx' parameter is ignored.

WOF_KEEPLY - The 'wy' parameter is ignored.

WOF_INTERPOLATE - Enables interpolation for the frame called on. For backwards compatibility however,

A_OverlayOffset will override the current interpolation setting if the function is called in the same tic.

Note, despite the default values for wx and wy arguments of the function are 0.5, the default values of the corresponding vector2 pivot field in the PSprite class are 0.0, meaning the default pivot point for overlay rotation and scale is the graphic's top left corner, not its center.

A_OverlayPivotAlign

A_OverlayPivotAlign (int layer, int halign, int valign)

Usage

Sets the starting alignment to one of the corners of the sprite for the pivot. A_OverlayPivot's offset is applied afterwards.

Parameters

layer - The layer to modify.

halign - Sets the horizontal alignment corner, which can be one of the following (not combinable). Set to -1 to keep the current alignment.

PSPA_LEFT - Default.

PSPA_CENTER

PSPA_RIGHT

valign - Same as halign but takes the following:

PSPA_TOP - Default.

PSPA_CENTER

PSPA_BOTTOM

A_OverlayRenderstyle

A_OverlayRenderstyle (int layer, int style)

Usage

Sets the renderstyle of the overlay specified. Requires A_OverlayFlags' PSPF_RENDERSTYLE set to true to have any effect.

Parameters

layer - The layer to modify.

style - Can be one of the following:

STYLE_None

STYLE_Normal

STYLE_Fuzzy

STYLE_SoulTrans

STYLE_OptFuzzy

STYLE_Stencil

STYLE_Translucent

STYLE_Add

STYLE_Shaded

STYLE_TranslucentStencil

STYLE_Shadow

STYLE_Subtract

STYLE_AddStencil

STYLE_AddShaded

A_OverlayRotate

void A_OverlayRotate (int layer [, double degrees [, int flags]])

Usage

Rotates the overlay around the pivot, set by A_OverlayPivot and A_OverlayPivotAlign.

Parameters

layer: The layer to modify.

degrees: Sets the rotation in degrees. Negative numbers define clockwise rotation, positive numbers are counter-clockwise. Default is 0.

flags: Can be combined with the '|' (without the):

WOF_ADD “ The input adds instead of replaces the current offset, and implies WOF_INTERPOLATE.

WOF_INTERPOLATE “ Enables interpolation for the frame called on. For backwards compatibility however, A_OverlayOffset will override the current interpolation setting if the function is called in the same tic.

Default is 0.

A_OverlayScale

A_OverlayScale (int layer, double wx = 1, double wy = 0, int flags = 0)

Usage

Changes the scale of the overlay, affected by the pivot's position.

Parameters

layer - The layer to modify.

wx - Sets the X scale.

wy - Sets the Y scale.

flags - Flags can be combined using '|' (without ' '):

WOF_RELATIVE - Takes into account the current rotation of the overlay.

WOF_ADD - The input adds instead of replaces the current offset, and implies WOF_INTERPOLATE.

WOF_KEEPPX - The 'wx' parameter is ignored.

WOF_KEEPPY - The 'wy' parameter is ignored.

WOF_ZEROY - If 'wy' is 0, it copies the 'wx' parameter. This flag disables that behavior.

WOF_INTERPOLATE - Enables interpolation for the frame called on. For backwards compatibility however,

A_OverlayOffset will override the current interpolation setting if the function is called in the same tic.

A_OverlayTranslation

void A_OverlayTranslation (int layer, name tname)

Usage

Sets the translation of the specified overlay by name.

Parameters

layer: the layer whose translation to set.

tname: the name of the translation to set, as defined in TRNSLATE.

A_OverlayVertexOffset

A_OverlayVertexOffset (int layer, int index, double x, double y, int flags = 0)

Usage

Allows for skewing of an overlay by manipulating the vertex directly. Note that modifications of the vertices are always relative to the rotation set by A_OverlayRotate.

Parameters

layer - The layer to modify.

index - Vertices are accessed by an array, and this is the index of the array. Accepted ranges are between [0,3].

x - Offsets the X position of the vertex to the right of the base position if positive.

y - Offsets the Y position of the vertex to the bottom of the base position if positive.

flags - Can be combined with the '|' (without the):

WOF_ADD - The input adds instead of replaces the current offset, and implies WOF_INTERPOLATE.

WOF_KEEPPX - The 'x' parameter is ignored.

WOF_KEEPPY - The 'y' parameter is ignored.

WOF_INTERPOLATE - Enables interpolation for the frame called on. For backwards compatibility however, A_OverlayOffset will override the current interpolation setting.

A_Raise

Note: this function should not be confused with the resurrection functions which have similar naming.

```
void A_Raise [(int raisespeed)]
```

Usage

This function must be called in a weapon's Select state sequence. It should not be used anywhere else. It is responsible for raising the weapon and for entering the Ready state sequence when it is done.

Each time the function is called, the weapon moves further up the screen as it approaches the ready position. You can therefore make a weapon raise faster than the default by calling the function more than once within the same tic, or by passing the desired speed directly.

Error.gif

Warning: Do not call this function repeatedly in the same state inside another function, anonymous or otherwise. All calls to A_Raise are executed fully inside the other function, even if the weapon has reached its ready position, resulting in unwanted side effects.

Parameters

raisespeed: how much the weapon is raised by. Default is 6.

Examples

Here is an example of a basic Select state that makes use of this function to bring the weapon up to the ready position:

Select:

```
SHTG A 1 A_Raise
```

```
Loop
```

This will raise the weapon up from the bottom of the screen when it is selected. Once it has reached the ready position, A_Raise will automatically jump to the Ready state.

This weapon raises into view twice as fast as normal:

Select:

```
FAST A 1 A_Raise(12)
```

```
Loop
```

A_Recoil

A_Recoil (double xyvel)

Pushes the calling actor back by the specified force. Although this is designed to simulate weapon recoil it can also be used to do special monster movement and other things. If xyvel is negative it can also be used to push the actor forward instead of backward.

Note that this function does not account for the player's pitch when used on a weapon. To account for pitch, multiply xyvel by $\cos(\text{pitch})$.

Examples

This extremely powerful shotgun throws you back when you fire it.

```
actor UltimaShotgun : Shotgun
{
    States
    {
        Fire:
            SHTG A 3
            SHTG A 0 A_Recoil(80)
            SHTG A 7 A_FireBullets(20, 20, 50, 5, "BulletPuff", 5)
            SHTG A 20
            SHTG BC 5
            SHTG D 4
            SHTG CB 5
            SHTG A 3
            SHTG A 7 A_ReFire
            Goto Ready
    }
}
```

A_ReFire

A_ReFire [(str "state")]

state: The state to go to. By default, it is the Hold state, or the Fire state if Hold is not defined, or the AltHold or AltFire state if the function is called from the AltFire sequence.

This function is normally called by weapons after the attack. It checks whether the fire key is still held and if so enters the relevant state.

Examples

Fire:

WEAP A 3 A_FireBullets(3, 3, 1, 6)

WEAP B 3

WEAP C 3 A_ReFire //If fire button is held, then it goes back to the beginning of the sequence, Otherwise it would continue

WEAP D 3

Goto Ready

A_ResetReloadCounter

(no parameters)

Resets the counter used by A_CheckForReload. This action is part of a generalized version of Skulltag's A_CheckRailReload and may not be particularly useful for creating complex reloading systems in weapons.

A_SetCrosshair

A_SetCrosshair (int number)

Sets or disables a custom crosshair for the currently selected weapon.

number: The crosshair number to use. A value of 0 resets the crosshair to the player's default selection.

You can use this function to set a specific crosshair for the currently-selected weapon, or change the current crosshair based on zoom factor or other conditions. The last crosshair set will be saved to the current weapon and reset if the player switches back to it at a later time.

Both the default crosshairs present in ZDoom.pk3 and custom crosshairs can be set with this function. If you want to disable the crosshair for a given weapon, you can include an empty graphic for your custom crosshair and use this function to set the weapon to use that crosshair.

Note that the player may override the custom crosshair by setting the crosshairforce CVAR to true.

A_WeaponOffset

```
void A_WeaponOffset [(double wx [, double wy [, int flags]])]
```

Usage

This function can only be called on weapons. Offsets a weapon by the defined amount of wx and wy similar to the Offset frame keyword. This function has the following differences from the aforementioned function:

Offsets are floats, not integers.

Offsets are not lost when firing a weapon.

Offsets stack on top of weapon bobbing and behave independently, allowing you to use A_WeaponReady without losing offsets.

Specifying the default coordinates does not serve as a preservative. They are absolute.

Offsets can go beyond screen limitations.

Because this is an action function itself, Decorate expressions can be used.

This function is the same as calling A_OverlayOffset with the layer parameter set to PSPF_WEAPON.

Parameters

wx: adjusts horizontal position based on positive (right) or negative (left) numbers. Default is 0.

wy: adjusts vertical position. Larger values lower the weapon, and vice versa. Default is 32.

flags: flags can be combined with the | separator. Default is 0:

WOF_KEEPM — wx parameter will not be used.

WOF_KEEPM — wy parameter will not be used.

WOF_ADD — Adds the current x and y parameters to the current offsets instead of overriding them. Also implies interpolation (see below).

WOF_INTERPOLATE — Interpolates the offset between tics, smoothing out the animation.

A_WeaponReady

A_WeaponReady [(int flags)]

Usage

A_WeaponReady is responsible for weapon bobbing and checking the fire keys and weapon changing keys. This function should be called in a weapon's "Ready" state so the weapon can be fired. The function can also be called in other states to allow a player to re-fire at any point.

Keep in mind that when A_WeaponReady is called, the sprite will accumulate a bobbing animation for around half a second if the player is in motion.

In case parts of this behavior are undesired, the following flags can be used:

WRF_NOBOB: The weapon's HUD sprites will not bob.

WRF_NOFIRE: The weapon will not be made ready for firing (same as

WRF_NOPRIMARY|WRF_NOSECONDARY).

WRF_NOSWITCH: The weapon will not be made ready for deselection (and so cannot be switched off until the next call to A_WeaponReady without that flag).

WRF_DISABLESWITCH: Prevents from deselecting the weapon entirely until the next call to A_WeaponReady.

NOSWITCH puts deselection on hold, whereas DISABLESWITCH cancels the deselection entirely.

WRF_NOPRIMARY: The weapon will not be ready for its normal fire.

WRF_NOSECONDARY: The weapon will not be ready for its alternate fire.

WRF_ALLOWRELOAD: The weapon will jump to the "Reload" state if the reload key is currently being pressed.

WRF_ALLOWZOOM: The weapon will jump to the "Zoom" state if the zoom key is currently being pressed.

WRF_ALLOWUSER#: The weapon will jump to the "User#" state defined, where # can be a number from 1-4.

Examples

Ready:

WEAP A 1 A_WeaponReady //Makes the weapon ready to fire

Loop

Fire:

WEPF A 4

WEPF B 4

WEPF C 4 A_WeaponReady(WRF_NOBOB|WRF_NOSWITCH) //Allows the weapon to re-fire, but without any weapon bobbing or switching

WEPF D 4

WEPF E 4

Goto Ready

A_ZoomFactor

A_ZoomFactor [(float zoom [, int flags])]

zoom: The amount to zoom in or out. The player's FOV is divided by this value. Default is 1.0.

flags: Flags to modify the behavior of the zoom.

A_ZoomFactor lets weapons scale their player's FOV. Each weapon maintains its own FOV scale independent from any other weapons the player may have.

Flags may be a combination of either of the following (or omitted):

ZOOM_INSTANT: The zoom is normally spread out across a few ticks to make a zooming effect. Use this flag to make the zoom instant.

ZOOM_NOSCALETURNING: Player turning is normally scaled as well by this function. Use this flag to retain the player's unzoomed sensitivity while zoomed.

This codepointer is restricted to Weapon and derived classes.

Examples

This sniper pistol has two levels of zoom, 2x and 4x.

```
actor SniperPistol_Zoomed : Inventory
{
    inventory.maxamount 2
}

actor SniperPistol : Pistol
{
    States
    {
        AltFire:
            PISG ABC 6
            TNT1 A 0 A_JumpIfInventory("SniperPistol_Zoomed", 2, "ZoomOut")
            TNT1 A 0 A_JumpIfInventory("SniperPistol_Zoomed", 1, "Zoom2")
            //fall through
        Zoom1:
            TNT1 A 0 A_ZoomFactor(2.0)
            TNT1 A 0 A_GiveInventory ("SniperPistol_Zoomed", 1)
            Goto "AltFireDone"
        Zoom2:
            TNT1 A 0 A_ZoomFactor(4.0)
            TNT1 A 0 A_GiveInventory ("SniperPistol_Zoomed", 1)
            Goto "AltFireDone"
        ZoomOut:
            TNT1 A 0 A_ZoomFactor(1.0)
            TNT1 A 0 A_TakeInventory ("SniperPistol_Zoomed", 2)
            Goto "AltFireDone"
        AltFireDone:
            PISG C 5 A_ReFire
            Goto "Ready"
        Deselect:
            TNT1 A 0 A_TakeInventory ("SniperPistol_Zoomed", 2)
            TNT1 A 0 A_ZoomFactor(1.0)
```

```

    Goto "Super::Deselect"
}
}
}

```

This is a fully functional sniper rifle. The altfire brings up the scope, and then gives a dummy inventory item. There's also an A_JumpIfInventory flag on the ready and fire states to check if the player have this dummy item, so the weapon can go to an alternate ready and fire states if you have pressed the altfire button.

ACTOR HKG3SG1 : Weapon replaces Berserk

```

{
    Inventory.PickupMessage "H&K G3/SG1"
    Inventory.PickupSound "misc/w_pkup"
    Weapon.AmmoGive 10
    Weapon.AmmoType "Clip"
    Weapon.AmmoType2 "Clip"
    Weapon.Ammouse 1
    Weapon.Ammouse2 1
    Weapon.Kickback 40
    DamageType "FriendlyFire"
    +NOAUTOFIRE
    +NOALERT
    Scale 0.14
    States
    {
        Spawn:
            G3SG A -1
            LOOP
        Ready:
            HKG3 A 0 A_JumpIfInventory("G3Zoom", 1, "ReadyScope")
            HKG3 A 1 A_WeaponReady
            LOOP
        ReadyScope:
            ASNS A 1 A_WeaponReady(WRF_NoBob)
            LOOP
        Deselect:
            TNT1 A 0 A_TakeInventory ("G3Zoom", 2)
            TNT1 A 0 A_ZoomFactor(1.0)
            HKG3 A 1 A_Lower
            Goto Deselect+2
        Select:
            HKG3 A 1 A_Raise
            TNT1 A 0 A_Raise
            LOOP
        Fire:
            ASNS A 0 A_JumpIfInventory("G3Zoom", 1, "FireScope")
            HKGF A 0 A_AlertMonsters
            HKGF A 0 A_PlayWeaponSound("weapons/sg550_fire")
            HKGF A 0 A_FireProjectile("GunSmokeSpawner",0,0,5,9)
            HKGF A 1 Bright A_FireBullets (1.5, 1.5, -1, 24, "Puff2")
            HKG3 B 0 A_FireProjectile("556MMCasingSpawner",0,0,14,4)
            HKG3 B 0 ACS_Execute(987,0,125,random(2,4),0)
            HKG3 BCDEFGHI 1 A_WeaponReady(WRF_NoFire)
            HKG3 A 9 A_WeaponReady(WRF_NoFire)
            Goto Ready
        FireScope:
    }
}

```

```

    ASNS A 0 A_AlertMonsters
    ASNS A 0 A_PlayWeaponSound("weapons/sg550_fire")
    ASNS A 1 Bright A_FireBullets (0, 0, -1, 24, "Puff2")
    ASNS A 0 A_FireProjectile("556MMCasingSpawner",0,0,14,4)
    ASNS A 0 ACS_Execute(987,0,95,random(2,4),0)
    ASNS A 17
    Goto ReadyScope
AltFire:
    ASNS A 0 A_SetCrosshair(50)
    ASNS A 2 A_WeaponReady(WRF_NoBob)
    TNT1 A 0 A_JumpIfInventory("G3Zoom", 2, "ZoomOut")
    TNT1 A 0 A_JumpIfInventory("G3Zoom", 1, "Zoom2")
    //fall through
Zoom1:
    TNT1 A 0 A_ZoomFactor(3.0)
    TNT1 A 0 A_GiveInventory ("G3Zoom", 1)
    Goto "AltFireDone"
Zoom2:
    TNT1 A 0 A_ZoomFactor(8.0)
    TNT1 A 0 A_GiveInventory ("G3Zoom", 1)
    Goto "AltFireDone"
ZoomOut:
    TNT1 A 0 A_SetCrosshair(0)
    TNT1 A 0 A_ZoomFactor(1.0)
    TNT1 A 0 A_TakeInventory ("G3Zoom", 2)
    Goto "AltFireDone"
AltFireDone:
    ASNS A 1 A_ReFire
    Goto "Ready"
}
}

actor G3Zoom : Inventory
{
    inventory.maxamount 2
}

```

Classes

Every type of actor (enemies, projectiles, item pickups, obstacles, special effects...) in ZDoom is called a class. For the purpose of this article, a class can be thought of as a way of grouping all the properties of an object under a single name. ZDoom defines all the classes needed for each of the supported game, and as long as the needed graphics and sounds are provided they can all be used in any game.

Normally, objects are placed on a map using either the thing numbers in a level editor, or using the SpawnID in an ACS script. However, each game supported by ZDoom has conflicting thing numbers, so to facilitate spawning objects from other games, the Spawn and SpawnSpot ACS commands were created. To place actors from another game directly in the map editor, you can define spawners that will have a non-conflicting editor number and drop the wanted actor.

You can also spawn classes manually while playing, via the Summon console command.

Categories by type

Artifact: Items that can be used after having been picked up.

Ammo: Ammunition for the various weapons.

Armor: Protective gear.

Breakable: Decoration that can be destroyed.

Bridge: Visible or invisible objects that can be walked upon.

Decoration: Sundry clutter and obstacles.

Dynamic light: Things that emit light. OpenGL.png (OpenGL only: not supported by ZDoom)

Explosive: Projectiles and puffs from traps, monsters and weapons.

Health: Restorative items.

Interactive object: Objects that can be interacted with, not just destroyed.

Gibs: Meat chunks, blood and guts, created during game.

Gore: Corpses and fleshy bits used as decoration.

Hazard: Traps and other dangerous objects.

Internal: Classes used by the engine to do various stuff.

Key: For opening locked doors.

Light sources: A type of decoration that is supposed to emit light.

Map spot: Invisible things that achieve certain special effects.

Monster: Enemies and allies that aren't controlled by a player.

Player: Enemies and allies that are controlled by a player.

Powerup: Items (other than health or armor) that are used on pickup.

Power: Effects and abilities gained by using an artifact or powerup.

Puzzle item: In practice, a type of key that is lost when used.

Quest item: An item that must be in the inventory to progress in Strife quests.

Script thing: For scripting actors and points of view.

Token: An hidden item placed in the inventory to help the engine keep track of things.

SFX: Special effects such as projectile trails, sparks, dripping water and so on.

Vegetation: Trees, mushrooms, mosses and lichens.

Weapon: What can go in a player's arsenal.

Actor mover

Actor type:Script thing

Game: ZDoom

DoomEd Number:9074

Class Name:ActorMover

Classes: PathFollower†'ActorMover

An ActorMover will move any thing along a path of interpolation points. If you move a monster, the monster will automatically be made dormant while it is on the path.

The actor mover takes four parameters:

1. low byte: low byte of tid of first interpolation point in path.
2. high byte: high byte of tid of first interpolation point in path.
3. options: (Add any of the following values; i.e. for options 2 and 4, this parameter would be 6):
 - 1: path is linear instead of curved.
 - 2: Thing will adjust its angle to match those of the points it passes.
 - 4: Thing will adjust its pitch to match those of the points it passes.
 - 8: When used with 2 and/or 4, the thing faces in the direction of movement instead of the direction the interpolation points are facing.
 - 128: If the thing being moved is normally solid, make it nonsolid so that it can't be blocked.
4. tid: the tid of the thing to move.

Use the Thing_Activate(ActorMover's tid) special to start the actor mover.

To stop an ActorMover use the reverse special, Thing_Deactivate(ActorMover's tid). If the camera is restarted using Thing_Activate, it will automatically start at the beginning of the path.

DECORATE definition

```
ACTOR ActorMover : PathFollower native {}
```

Hate target
Actor type:Script thing
Game:ZDoom
DoomEd Number:9076
Class Name:HateTarget
Classes: HateTarget

A hate target is an invisible actor that can be attacked. It is meant to be used in scripts along with Thing_Hate; for example it can be used to make monsters attack a wall (actually the hate targets on the wall), and the wall can then be scripted to be destroyed when the hate targets die.

A hate target's health depends on its angle: if the angle is 0, the hate target is immortal and takes no damage. If the angle is a positive value, however, its health is equal to ten time the angle value and it takes damage normally.

DECORATE definition

ACTOR HateTarget native

```
{
  Radius 20
  Height 56
  +SHOOTABLE
  +NOGRAVITY
  +NOBLOOD
  +DONTSPASH
  Mass 0x7ffffff
  States
  {
  Spawn:
    TNT1 A -1
  }
}
```

Path follower
Actor type: Script thing
Game: ZDoom
DoomEd Number: 9071
Class Name: PathFollower
Classes: PathFollower
→ActorMover
→MovingCamera

A path follower is an invisible thing that follows a path of interpolation points and can provide something for a camera to aim it if you want the camera to follow a path with a complicated aiming sequence.

The path follower takes three parameters:

1. low byte: low byte of tid of first interpolation point in path.
 2. high byte: high byte of tid of first interpolation point in path.
 3. options: (Add any of the following values; i.e. for options 2 and 4, this parameter would be 6):
 - 1: path is linear instead of curved.
 - 2: Camera will adjust its angle to match those of the points it passes.
 - 4: Camera will adjust its pitch to match those of the points it passes.
 - 8: When used with 2 and/or 4, the camera faces in the direction of movement instead of the direction the interpolation points are facing.
- Use the Thing_Activate special to start the path follower. The Thing_Activate takes a single parameter, the tid of the thing to activate. To stop a path follower use Thing_Deactivate. Thing_Deactivate takes one parameter, the tid of the thing to deactivate. If the camera is restarted using Thing_Activate, it will automatically start at the beginning of the path.

DECORATE definition

```
ACTOR PathFollower native
{
    +NOBLOCKMAP
    +NOSECTOR
    +NOGRAVITY
    +DONTSPASH
}
```

Patrol point
Actor type:Script thing
Game:ZDoom
DoomEd Number:9024
Class Name:PatrolPoint
Classes: PatrolPoint

Monster routes are controlled through the use of patrol points, or path nodes. A patrol point ignores its special and assigns special meaning to its arguments.

Arguments

Patrol points take two arguments:

args[0] → TID of the next PatrolPoint in the path.

args[1] → time (in seconds) a monster should wait at this node before proceeding to the next one.

To set a monster on a path, you have two options: use the Thing_SetGoal special in a script or on a line to send a monster or group of monsters to a node, or set a monster's special to Thing_SetGoal. If a monster's special is Thing_SetGoal, and the tid specified for the special is 0, the monster will be sent to the node automatically when the level is loaded, and its special will be ignored if it dies.

Also of note is the patrol special thing which can be used to execute a thing special when an actor reaches a certain path node.

DECORATE definition

ACTOR PatrolPoint

```
{  
    Radius 8  
    Height 8  
    Mass 10  
    +NOGRAVITY  
    +NOBLOCKMAP  
    +DONTSPASH  
    RenderStyle None  
}
```

Patrol special
Actor type: Script thing
Game: ZDoom
DoomEd Number: 9047
Class Name: PatrolSpecial
Classes: PatrolSpecial

A patrol special triggers its special whenever a patrol point with the same tid is used by a monster on patrol.

DECORATE definition

ACTOR PatrolSpecial

```
{  
  Radius 8  
  Height 8  
  Mass 10  
  +NOGRAVITY  
  +NOBLOCKMAP  
  +DONTSPASH  
  RenderStyle None  
}
```

Bridge thing
Actor type: Bridge
Game: Doom
DoomEd Number: 118
Class Name: ZBridge
Spawn ID: 21
Identifier: T_BRIDGE
Classes: CustomBridge†ZBridge

The bridge is a "3D" element that can be placed at any height and allows players and monsters alike to walk upon it. It is usually combined with mid-textures in order to create an approximation of a 3D floor; though this trick is no longer needed now that "3D mid-textures" are supported.

For a reason known only to Randy, the default sprite provided by ZDoom.pk3 is a yellow rectangle with "GROSS HACK" written on it, so in order to use this bridge in other games than Hexen it is necessary to provide replacement sprites.

DECORATE definition

```
ACTOR ZBridge : CustomBridge
{
    Args 36, 4, 0, 0
}
```

Custom bridge
Actor type: Bridge
Game: ZDoom
DoomEd Number: 9991
Class Name: CustomBridge
Classes: CustomBridge
→Bridge
→ZBridge

CustomBridge can be used to create either a Doom-style or Hexen-style invisible bridge, and can also further customize the appearance and behavior of the rotating "bridge balls" when used as a Hexen-style bridge.

When placed in a map, the bridge's arguments control its appearance and behavior, as follows:

Arg 1: Bridge radius, in mapunits
Arg 2: Bridge height, in mapunits
Arg 3: Number of bridge balls to display.
Arg 4: Rotation speed and direction of bridge balls. 0 uses Hexen's default. Values from 1-128 rotate counterclockwise, while values from 129-255 go clockwise. For example:
0: Hexen default
11: 15° / second
21: 30° / second
32: 45° / second
64: 90° / second
128: 180° / second
192: -90° / second
223: -45° / second
233: -30° / second
244: -15° / second
Arg 5: Rotation radius of bridge balls, in bridge radius %. 0 is Hexen-default (15 units regardless of bridge radius)

Note: When inheriting from this actor in Decorate, you can redefine the Spawn state to change the way the bridge behaves as a Doom-format bridge (Arg 3 set to 0), or the See state to alter the way it behaves as a Hexen bridge. This can be used, for example, to change which actors are spawned instead of the bridge balls.

DECORATE definition

ACTOR CustomBridge native

```
{
  +SOLID
  +NOGRAVITY
  +NOLIFTDROP
  +ACTLIKEBRIDGE
  Radius 32
  Height 2
  RenderStyle None

  action native A_BridgeInit(class<Actor> balltype = "BridgeBall");

  States
  {
    Spawn:
      TLGL ABCDE 3 Bright
      Loop
```

See:
TLGL A 2
TLGL A 2 A_BridgeInit
TLGL A -1
Stop

Death:
TLGL A 2
TLGL A 300
Stop

}
}

Glitter bridge ball
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: BridgeBall
Classes: BridgeBall

One of the tiny sparkles that orbit around the center of a bridge. When creating derived classes, the native action must be called every tic to work as intended.

DECORATE definition

ACTOR BridgeBall

```
{  
+NOBLOCKMAP  
+NOTELEPORT  
+NOGRAVITY
```

```
action native A_BridgeOrbit();
```

States

```
{  
Spawn:  
    TLGL A 2 Bright  
    TLGL A 1 Bright A_BridgeOrbit  
    Wait  
}  
}
```

Glitter Bridge

Actor type: Bridge

Game: Raven

DoomEd Number: 118

Class Name: Bridge

Spawn ID: 21

Identifier: T_BRIDGE

Classes: CustomBridge'Bridge

The bridge is a "3D" element that can be placed at any height and allows players and monsters alike to walk upon it. It is made visible by three tiny glittering orbs orbiting the center of the bridge.

DECORATE definition

ACTOR Bridge : CustomBridge

{

RenderStyle None

Args 32, 2, 3, 0

}

Invisible bridge thing
Actor type: Bridge
Game: ZDoom
DoomEd Number: 9990
Class Name: InvisibleBridge
Classes: InvisibleBridge
→InvisibleBridge8
→InvisibleBridge16
→InvisibleBridge32

The invisible bridge is a "3D" element that can be placed at any height and allows players and monsters alike to walk upon it. It is usually combined with midtextures in order to create an approximation of a 3D floor, and is in itself invisible.

A bridge (and derived classes) can be customized by giving it non-zero arguments: the first argument is used as its radius, the second as its height. See also CustomBridge for visible bridges.

DECORATE definition

```
ACTOR InvisibleBridge native
{
  RenderStyle None
  Radius 32 // Or args[0]
  Height 4 // Or args[1]
  +SOLID
  +NOGRAVITY
  +NOLIFTDROP
  +ACTLIKEBRIDGE
  States
  {
    Spawn:
      TNT1 A -1
      Stop
  }
}
```

Invisible bridge thing
Actor type Bridge
Game: ZDoom
DoomEd Number: 5064
Class Name: InvisibleBridge16
Classes: InvisibleBridge†'InvisibleBridge16

The invisible bridge is a "3D" element that can be placed at any height and allows players and monsters alike to walk upon it. It is usually combined with midtextures in order to create an approximation of a 3D floor, and is in itself invisible.

A bridge (and derived classes) can be customized by giving it non-zero arguments: the first argument is used as its radius, the second as its height. See also CustomBridge for visible bridges.

DECORATE definition

```
ACTOR InvisibleBridge16 : InvisibleBridge
{
    Radius 16 // Or args[0]
    Height 8 // Or args[1]
}
```

Invisible bridge thing
Actor type: Bridge
Game: ZDoom
DoomEd Number: 5061
Class Name: InvisibleBridge32
Classes: InvisibleBridge†'InvisibleBridge32

The invisible bridge is a "3D" element that can be placed at any height and allows players and monsters alike to walk upon it. It is usually combined with midtextures in order to create an approximation of a 3D floor, and is in itself invisible.

A bridge (and derived classes) can be customized by giving it non-zero arguments: the first argument is used as its radius, the second as its height. See also CustomBridge for visible bridges.

DECORATE definition

```
ACTOR InvisibleBridge32 : InvisibleBridge
{
    Radius 32 // Or args[0]
    Height 8 // Or args[1]
}
```

Aiming camera

Actor type: Script thing

Game: ZDoom

DoomEd Number: 9073

Class Name: AimingCamera

Classes: SecurityCamera→AimingCamera

The AimingCamera is another ZDoom camera similar to the SecurityCamera. However, this camera will aim at and follow a target.

The camera is activated by using the ChangeCamera (237) special or by a SetCameraToTexture in ACS.

Arguments

The aiming camera takes four arguments:

args[0] → pitch of camera in degrees, equal to the angle subtracted from 256.

args[1] → maximum number of degrees to yaw (rotate side to side) per second.

args[2] → maximum number of degrees to pitch (rotate up and down) per second.

args[3] → tid of thing to follow.

DECORATE definition

ACTOR AimingCamera : SecurityCamera native {}

Interpolation point
Actor type: Script thing
Game: ZDoom
DoomEd Number: 9070
Class Name: InterpolationPoint
Classes: InterpolationPoint

Interpolation points define a path to follow. They operate similarly to the patrol points of a monster patrol route.

Arguments

Interpolation points take five arguments:

args[0] → pitch of camera in degrees, equal to the angle subtracted from 256.
args[1] → time (in octics) to travel to next node.
args[2] → time (in octics) to stop at this node before continuing
args[3] → low byte of tid of next InterpolationPoint in the path.
args[4] → high byte of tid of next InterpolationPoint in the path.
args[3] has the low byte, and args[4] has the high byte, so the tid is $\text{args}[3] + (\text{args}[4] * 256)$. This lets you use more than 255 points for all paths. For example, if the next point had a tid of 4 then the low byte is 4 and the high byte is 0. But if the next point has a tid of 3027, the low byte is 211 and the high byte is 11 ($11 \times 256 + 211 = 3027$).

Since an interpolation point needs parameters to work, it cannot have a thing special. To have a thing special executed when an interpolation point is used by one of the moving objects (ActorMover, MovingCamera or PathFollower), you need to create a an interpolation special with the same TID as the interpolation point.

DECORATE definition

```
ACTOR InterpolationPoint native
{
    +NOBLOCKMAP
    +NOGRAVITY
    +DONTSPASH
    RenderStyle None
}
```

Interpolation special
Actor type: Script thing
Game: Zdoom
DoomEd Number: 9075
Class Name: InterpolationSpecial
Classes: InterpolationSpecial

An interpolation special triggers its special whenever an interpolation point with the same TID is used by any of the moving objects (ActorMover, MovingCamera or PathFollower).

DECORATE definition

ACTOR InterpolationSpecial native

```
{  
+NOBLOCKMAP  
+NOSECTOR  
+NOGRAVITY  
+DONTSPASH  
}
```


Moving camera
Actor type: Script thing
Game: ZDoom
DoomEd Number: 9072
Class Name: MovingCamera
Classes: PathFollower→MovingCamera

Usage

The MovingCamera is, as the name implies, a camera that moves along a predetermined path. The movement is quite smooth and is perfect for creating those cut scenes found in most games today.

To start a moving camera use Thing_Activate. To stop a moving camera use Thing_Deactivate. The camera will automatically start at the beginning of the path when using Thing_Activate. The camera will not respond to activation again until it has finished moving along the path. If you wish to restart the camera while it is still following the path, you must call Thing_Deactivate first. You can use the ChangeCamera (237) special or SetCameraToTexture in ACS to view the camera.

Arguments

The moving camera takes four arguments:

args[0] → low byte of tid of first InterpolationPoint in path.
args[1] → high byte of tid of first InterpolationPoint in path.
args[2] → options: (Add any of the following values; i.e. for options 2 and 4, this parameter would be 6):
1: path is linear instead of curved.
2: camera will adjust its angle to match those of the points it passes.
4: camera will adjust its pitch to match those of the points it passes.
8: when used with 2 and/or 4, the camera faces in the direction of movement instead of the direction the interpolation points are facing.
128: all players will see the camera's view, and not just the player who activates it.
args[3] → tid of thing to look at (or 0). If specified, settings 2 and 8 in args[2] are ignored. Setting 4 will adjust pitch to center the thing being aimed at.

DECORATE definition

```
ACTOR MovingCamera : PathFollower native
{
    CameraHeight 0
}
```

Security camera
Actor type: Script thing
Game: ZDoom
DoomEd Number: 9025
Class Name: SecurityCamera
Classes: SecurityCamera
→AimingCamera

Usage

The SecurityCamera is a camera that pans from side to side, much like the security camera in Duke Nukem 3D. It is a great addition to either a single player or multiplayer map.

Place the camera in the map and set the thing angle to point in the direction you want the camera to face. Set the three arguments of the camera and give the camera a unique TID.

The camera is activated by using the ChangeCamera (237) special or by a SetCameraToTexture in ACS.

Arguments

The security camera takes three arguments:

args[0] → pitch of camera in degrees, equal to the angle subtracted from 256.

args[1] → number of degrees camera will rotate in either direction from its original orientation.

args[2] → number of octics to complete one cycle of turning.

DECORATE definition

ACTOR SecurityCamera native

```
{
  +NOBLOCKMAP
  +NOGRAVITY
  +DONTSPASH
  RenderStyle None
  CameraHeight 0
}
```

Error marker
Actor type: Internal
Game: ZDoom
DoomEd Number: None
Class Name: Unknown
Classes: Unknown
→SpeakerIcon

This actor class is as an exclamation mark that replaces any problematic objects in a map. An actor is replaced if it is:

a map thing that specifies a DoomEd Number that does not match any existing actor
an actor whose sprite for the first "Spawn" frame is missing
a RandomSpawner that went into too many recursions and might be stuck in an infinite loop
In such an event of replacement an error message is logged into the console to help diagnose the cause of the problem. This actor is always guaranteed to be defined and must be defined for ZDoom to function properly.

DECORATE definition

```
ACTOR Unknown
{
    Radius 32
    Height 56
    +NOGRAVITY
    +NOBLOCKMAP
    +DONTSPASH
    States
    {
        Spawn:
            UNKN A -1
            Stop
    }
}
```

Random spawner

Actor type Internal Game MiniZDoomLogoIcon.png
DoomEd Number None Class Name RandomSpawner

Classes: RandomSpawner

The RandomSpawner is a special actor which randomly spawns one of a series of actors specified with the DropItem property in its DECORATE code.

The RandomSpawner should not be used directly. Instead, authors should derive a new class from the actor and specify the actors they wish to be randomly spawned in the code of the new actor.

In addition to specifying the actors that may spawn, two optional parameters may be specified for each entry in the list. The first is an integer used to specify the spawn chance of any given monster in the list, assuming it is selected to spawn, with 0 being never and 255 being always (default is 255). The second number specifies this actor's "weight" or chance of spawning versus other actors in the list (default is 1).

It is possible for RandomSpawners to spawn other RandomSpawners. However, to prevent infinite recursion loops, an error marker will instead be created if more than 32 random spawners get nested in such a way.

Contents

- 1 Examples
 - 1.1 Basic example
 - 1.2 Use of parameters
 - 1.3 Keyword-based replacer
 - 1.4 Editor number-based replacer
- 2 Base RandomSpawner definition
 - 2.1 ZScript definition
 - 2.2 DECORATE definition

Examples

Basic example

This will create a random spawner which when placed in the map will always spawn one of four enemies with equal chance:

DECORATE:

```
actor MySpawner : RandomSpawner 1111
{
    DropItem "ZombieMan"
    DropItem "DoomImp"
    DropItem "HellKnight"
    DropItem "BaronOfHell"
}
```

Note: when using ZScript, editor numbers (aka DoomEdNums) have to be defined via MAPINFO.

Use of parameters

This code will create a spawner which will randomly choose from among four enemies with varying chance, and will sometimes spawn nothing when one of the larger enemies is chosen:

DECORATE:

```
actor MySpawner : RandomSpawner 1112
```

```
{
  DropItem "ZombieMan", 255, 10
  DropItem "DoomImp", 255, 8
  DropItem "HellKnight", 128, 4
  DropItem "BaronOfHell", 64, 1
}
```

Both parameters affect the likelihood that one of the actors will appear, but in different ways.

The first affects the probability that it will appear if selected. Using 255 is the equivalent of omitting the parameter entirely, and guarantees that the actor will spawn when selected. In this case, the Zombieman and Imp will always spawn if selected. The Hell Knight, if selected, will only spawn half the time, and the Baron will only spawn 1/4th of the time. Otherwise nothing will spawn at all.

The second weighs the probability the actor will be selected. Here, the Zombieman will be selected 10 times on 23 (about 43% of the times); ten is its weight and 23 is the total weight of all actors (10+8+4+1).

Keyword-based replacer

A random spawner can be used to replace a given actor with a randomly-chosen one. There is one caveat, though: the replaced actor cannot be spawned directly. For example, the following code will create a spawner which replaces all the zombies in the map and will spawn either a pistol zombie (75% chance) or a shotgun zombie (25% chance). Note the creation of the ZombieMan2 actor to avoid creating a recursion with the spawner spawning a copy of itself if it selects a ZombieMan.

DECORATE:

```
actor ZombieMan2 : ZombieMan {}

actor ZombieReplacer : RandomSpawner replaces ZombieMan
{
  DropItem "ZombieMan2", 255, 3
  DropItem "ShotgunGuy", 255, 1
}
```

Editor number-based replacer

To avoid the inconvenience of having to define clones of replaced actors, it is possible to instead give the random spawner the doom editor number of the actor to replace. Our previous example would then be:

DECORATE:

In DECORATE editor numbers can be given directly:

```
actor ZombieReplacer : RandomSpawner 3004
{
  DropItem "ZombieMan", 255, 3
  DropItem "ShotgunGuy", 255, 1
}
```

3004 is the doom editor number of the ZombieMan, so he is replaced when the map is initialized. The advantage of this method is that this makes the spawner more compatible with other mods (for example, one that attaches dynamic lights to its firing frame, or one that replaces them with a modified actor to achieve certain special effects). The drawback is that only actors directly placed on the map in the editor will be replaced; not those spawned by ACS or other custom actors.

Important note: for replacing boss monsters (Arachnotron, BaronOfHell, Cyberdemon, Fatso, Ironlich, Minotaur, Sorcerer2 and SpiderMastermind), you need to use the replaces keyword or the game will not be able to know that the boss monsters are being replaced. In ZScript you can also use the CheckReplacement() event.

A spawner normally removes itself entirely from the game after spawning a monster; but if it finds the BOSS or BOSSDEATH flag on a monster it replaces or spawns, it will stay around and monitor its health so as to call A_BossDeath when the monster dies. It is the spawner, not the monster itself, which will trigger the map's special action in such a scenario.

For a scripted map, as long as the monsters which must trigger special actions have the special directly set on them or are identified by a TID rather than by their type, the random spawners should work correctly. However, scripts which identify monsters by their type (using functions such as ThingCount or ThingCountName) will be broken unless the spawned replacements are made as bosses.

DECORATE definition

ACTOR RandomSpawner native

```
{
+NOBLOCKMAP
+NOSECTOR
+NOGRAVITY
+THRUACTORS
}
```

Secret trigger

Actor type: Script thing

Game: ZDoom

DoomEd Number: 9046

Class Name: SecretTrigger

Classes: SecretTrigger

Usage

A SecretTrigger counts as a found secret when activated.

The secret trigger destroys itself after activation and cannot be reactivated.

Arguments

The secret trigger takes one argument, which indicates how the player is told about it:

args[0]â€™ notification type

0: Message and sound effect

1: Message only

2: Sound effect only

3: Neither message nor sound effect

DECORATE definition

ACTOR SecretTrigger native

```
{
+NOBLOCKMAP
+NOSECTOR
+NOGRAVITY
+DONTSPASH
}
```

Sector silencer
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9082
Class Name: SectorSilencer
Classes: SectorSilencer

A sector silencer makes the sector it is in silent. No game world sound can be emitted from nor heard within the sector.

DECORATE definition

ACTOR SectorSilencer native

```
{  
+NOBLOCKMAP  
+NOGRAVITY  
+DONTSPASH  
RenderStyle None  
}
```


Sound environment
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9048
Class Name: SoundEnvironment
Classes: SoundEnvironment

A sound environment changes the reverberation qualities of an area in the map. This is not limited to a sector, boundaries must be drawn with Line_SetIdentification (in Hexen format) or the zoneboundary flag set to true (in UDMF). Only one reverb can be active in an area at a time.

The thing takes two arguments, which define which type of environment to use. See the REVERBS lump for a list of environments. For example, with a first argument of 30 and a second argument of 0, the "Castle Alcove" environment (30-0) is used.

A sound environment thing given the DORMANT flag in the map editor does not take effect until activated, but an active environment cannot be simply deactivated to disable the effect; instead another environment must be activated in the same area. The environment 0-0 can be used in this way to disable reverberation effects.

DECORATE definition

```
ACTOR SoundEnvironment native
{
+NOSECTOR
+NOBLOCKMAP
+NOGRAVITY
+DONTSPASH
}
```

Water zone

Actor type: Map spot

Game: ZDoom

DoomEd Number: 9045

Class Name: WaterZone

Classes: Actor + WaterZone

A water zone thing makes the sector it is in underwater and possible to swim within. Mainly used for Transfer_Heights dummy sectors to make swimmable deep water but can be used in any sector. Players can drown in water zones after a certain time. This can be defined in MAPINFO.

DECORATE definition

ACTOR WaterZone native

```
{
+NOSECTOR
+NOBLOCKMAP
+NOGRAVITY
+DONTSPASH
}
```

Player gibs
Actor type: Internal
Game: ZDoom
DoomEd Number: None
Class Name: PlayerChunk
Classes: Actor → PlayerPawn → PlayerChunk
→ BloodyFighterSkull
→ BloodySkull
→ IceChunkHead

Base class for actors that are created with the death of a player.

DECORATE definition

ACTOR PlayerChunk : PlayerPawn native

```
{  
  +NOSKIN  
  -SOLID  
  -SHOOTABLE  
  -PICKUP  
  -NOTDMATCH  
  -FRIENDLY  
  -SLIDESONWALLS  
  -CANPUSHWALLS  
  -FLOORCLIP  
  -WINDTHRUST  
  -TELESTOMP  
}
```

Player pawn
Actor type: Internal
Game: ZDoom
DoomEd Number: None
Class Name: PlayerPawn
Classes: PlayerPawn
→DoomPlayer
→ChexPlayer
→ClericPlayer
→ChickenPlayer
→FighterPlayer
→HereticPlayer
→MagePlayer
→PigPlayer
→PlayerChunk
→StrifePlayer

PlayerPawn is the base class for player actors. By inheriting from it you can create custom player classes.

Using in DECORATE

The PlayerPawn base class defines the following properties which are available to all player subclasses:

Player.AirCapacity value

Sets the air supply for the player class. This acts as a multiplier for the level's air supply value. A value of 2 will double the air supply, for instance. A value of 0 or less means the air supply is infinite, and thus the player class cannot drown.

Default is 1.0.

Player.AttackZOffset value

The height offset (in Doom map pixels) from the center of the player at which his attacks are fired. Scales according to crouched height.

Player.ClearColorSet number

Removes a color set from the player class. This allows to remove color sets inherited from a parent class.

Player.ColorRange start, end

Defines the color range in which to translate the player's sprite.

Default is no translation (0, 0).

Player.ColorSet number, name, start, end, color [...]

Defines a preset color set for the player class. The number must be unique. The name is used to identify the preset in the player setup menu. The start and end parameters determine the translation for the ColorRange, and the color parameter is used as the visual identifier for the set in the scoreboard.

Up to six additional color ranges can be defined, each with four parameter for range_start, range_end, first_color and last_color, in this order. See StrifePlayer for an example.

Player.ColorSetFile number, name, table, color

Defines a preset color set for the player class. The number must be unique. The name is used to identify the preset in the player setup menu. The translation is indicated by a translation table lump using the format defined by Hexen (TRANTBL0 to TRANTBLD). The color parameter is used as the visual identifier for the set in the scoreboard.

Player.CrouchSprite sprite

Defines the sprites for when your skin is crouching. If you don't define them your sprite will shrink half its vertical size to show it is crouching. For the crouch sprites you have to define all frames apart from the dying and gib frames. Please note that there is no "Crouch" state that is definable for players, as the engine would not support such a feature. Players need to be able to move seamlessly between crouching and standing during any given state (See, Missile, Melee, etc.) without losing their place in that state, hence there can be no generic "Crouch" state to jump to.

Player.DamageScreenColor color[, intensity[, damagetype]]

The color the player's display turns when the player is in pain. This can be specified as three hex values for

R, G, B respectively, or as a literal color name (e.g. "blue"). If this isn't defined, it defaults to red. The intensity of the flash can be scaled from 0.0 to 1.0. If a damagetype is specified, the color only applies when the last damage type received is that type. If there is no color specified for a given damage type, it defaults to what is set without one.

Player.DisplayName string

This is the identification string of the player class. It is used in addplayerclass KEYCONF command, in playerclass console variable, in skin definitions, and in menus. Each player class used in game must have a unique display name!

Player.Face string

This is the three-character prefix that will be used by the status bar face display built into DOOM and by custom status bars that define faces; note that with custom status bars in SBARINFO, faces can be used in all games (not just DOOM), provided the necessary graphics are supplied. This is like the face property of S_SKIN but for player classes. To include a status bar face you must put the 3 first letters of the sprite for the face here. For example, if one of the animations was BUGST00 you would put Face "BUG".

When deciding which face to use, the engine checks for STF**** (DOOM games only), the SBARINFO face, the current player class face, the face of the currently loaded skin and finally the morphed animal player class face; whichever it finds last "wins".

The engine also maintains faces across morphing; for example, if you were playing deathmatch in HeXen and have loaded an SBARINFO which defines faces for the fighter and also a DECORATE script containing a custom morph attack weapon that changes your enemy into a dog and includes dog face graphics; if you are turned into a dog, the status bar would change to show your face as a dog and turn back to a human again when you unmorph.

Player.FallingScreamSpeed value_min, value_max

When a player is falling within this range of speeds, they will play *falling sound.

Default is (35.0, 40.0).

Player.FlechetteType string

Type of fléchette used by this player class.

Player.FlyBob value (development version 0d23816 only)

A multiplier for how much the players' view and height bobs up and down when they are flying and the fviewbob console variable is on.

Default is 1.0.

Player.ForwardMove value[, value-run]

Speed modifier for forward/backward movement. The value is for walking and value-run is for running.

Default is 1 for both values. The default run speed is double the walking speed, so Player.ForwardMove 1, 0.5 would disable running.

Player.GruntSpeed value

The minimum speed a player must be falling at the time of landing to play *grunt sound.

Default is 12.0.

Player.HealRadiusType string

The nature of the healing effect the player offers to his allies in cooperative multiplayer when using a Mystic Ambit Incant.

Value can be Health, Armor or Mana. Default is Health.

Player.HexenArmor base value, value armor, value shield, value helm, value amulet

The player's armor class values in Hexen. All values must be a multiple of five (they are divided by five when displayed on the HUD). Base value is the player class's minimum, all other values are the increases gained by picking up the corresponding Hexen armor item.

Player.InvulnerabilityMode string

The secondary effect an invulnerability powerup will have on the player, in addition to the invulnerability and display gimmick.

Ghost will make the player phase in an out of ghost mode rapidly; Reflective will make the player reflect projectile attacks.

Default is unnamed. To obtain the default effect, just make sure the line is not present.

Player.JumpZ value (fixed point)

The player's jump speed. The player's jump height in normal gravity is $(\text{value} \times (\text{value} + 1)) / 2$. A player can still climb their MaxStepHeight while jumping, allowing to reach heights higher than indicated purely by the jump height. For example with the default values, a player can climb a height of up to 60 map units: 36 from jump, 24 from step.

Default is 8.0.

Player.MaxHealth value

Default is 100.

The maximum health reachable by using normal health items.

Player.MorphWeapon weapon

Sets the weapon the player will use when morphed to this class.

Player.MugShotMaxHealth value

Defines the value that is considered to be max health for the status bar mugshot. Negative values use the player's max health as the mug shot max health, zero (default value) uses 100 as the mug shot max health, and positive values are used directly as the mug shot max health.

Player.Portrait string

Replaces the player portrait in the class selection menu to show a customized image, much like Hexen's player classes.

Player.RunHealth value

The minimum health the player can have and still be able to run. (This is normally 16% for Strife)

Default is 0.

Player.ScoreIcon sprite

The icon visible next to the player's name on multiplayer scoreboard.

Player.SideMove value [value-walk]

Speed modifier for left/right movement. The value is for walking and value-run is for running.

Default is 1 for both values. The default run speed is double the walking speed, so Player.SideMove 1, 0.5 would disable side-running.

NOTE: Side movement running speed is actually 4/5ths forward running speed, and the side walk speed is 24/25ths forward walk.

Player.SoundClass string

The sound class used by SNDINFO's \$playersound command.

Default is "player".

Player.SpawnClass spawnclass

This is the filter for spawning things in a map. The spawnclass can be one of Any, Fighter, Cleric, Mage or a number between 1 and 16. 1 is synonymous with Fighter, 2 with Cleric and 3 with Mage.

Player.StartItem classname [amount]

Adds an item to player's start inventory. First weapon added is the weapon selected at start.

Note: The initial startitem list is never inherited and must be specified in full for each player class.

Player.TeleportFreezeTime value

The time in tics the player remains immobile after teleportation. If this is 0 or less, or the player has at least one active powerup which has the INVENTORY.NOTELEPORTFREEZE flag set, this behavior is overridden, and as a result, the player does not become immobile after teleportation.

Default is 18.

Player.UseRange range

Determines how far away a player class can activate lines or objects with the +use command.

Default is 64.0.

Player.ViewBob value

A multiplier for move and still bob.

Default is 1.0.

Player.ViewHeight value

Specifies how high above the floor the player's eyes are.

Default is 41.0.

Player.WaterClimbSpeed value (development version 73159da only)

The speed at which the player can climb walls when swimming.

Default is 3.5.

Player.WeaponSlot slot, weapon1[, weapon2, weapon3, ...]

Assigns the specified weapons to the given slot number for this player class. slot can be any single digit from 0 - 9. This is followed by a comma-separated list of each weapon's class name that should be assigned to that slot. If there is more than one weapon assigned to a given slot, the player can cycle between them by repeatedly pressing the slot button. For other ways to assign weapons to slots, see weapon slots.

PlayerPawn-specific flags also exist:

PLAYERPAWN.NOTHRUSTWHENINVUL

Player is not thrusted by attacks when invulnerable.

PLAYERPAWN.CANSUPERMORPH

If this flag is given to a player morph, the morphed player receives a tome of power if they are attempted to be morphed again. This is used by the chicken morph to reproduce the original behavior of Heretic and its "super chickens".

PLAYERPAWN.CROUCHABLEMORPH

Enables crouching for morphed player classes.

PLAYERPAWN.WEAPONLEVEL2ENDED

Signals that an enhanced weapon power has expired, so the code responsible switches the player's powered-up weapon back to its normal form.

This flag was introduced as part of a solution to make switching from the powered-up weapon to its normal form seamless upon the power's expiration, so it may not be used in user code.

DECORATE definition

ACTOR PlayerPawn native

```
{
    Health 100
    Radius 16
    Height 56
    Mass 100
    Painchance 255
    Speed 1
    +SOLID
    +SHOOTABLE
    +DROPOFF
    +PICKUP
    +NOTDMATCH
    +FRIENDLY
    +SLIDESONWALLS
    +CANPASS
    +CANPUSHWALLS
    +FLOORCLIP
    +WINDTHRUST
    +TELESTOMP
    +NOBLOCKMONST
    Player.AttackZOffset 8
    Player.JumpZ 8
    Player.GruntSpeed 12
    Player.FallingScreamSpeed 35, 40
    Player.ViewHeight 41
    Player.UseRange 64
    Player.ForwardMove 1,1
    Player.SideMove 1,1
    Player.ColorRange 0,0
    Player.SoundClass "player"
    Player.DamageScreenColor "ff 00 00"
    Player.MugShotMaxHealth 0
    Player.FlechetteType "ArtiPoisonBag3"
    Player.AirCapacity 1
    Obituary "$OB_MPDEFAULT"
}
```

Player speed trail
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: PlayerSpeedTrail
Classes: PlayerSpeedTrail

The trail of after-images left by a player using the boots of speed or equivalent power. They gradually fade out in seven tics, becoming more and more translucent every tic, and are destroyed at the eighth tic. The actor can only be seen in chasecam, by other players and cameras but never by creator's point of view.

DECORATE definition

ACTOR PlayerSpeedTrail native

```
{  
+NOBLOCKMAP  
+NOGRAVITY  
Alpha 0.6 // This value is decreased by 0.075 every tic, and the actor destroyed after it reaches 0.075 or below.  
RenderStyle Translucent  
}
```


Map marker
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9040
Class Name: MapMarker
Classes: MapMarker

A map marker is an actor which shows up on the automap, and can be used to point out points of interest, specific monsters, etc. Inherit from this class to create your own custom map marker. Setting the thing's special arguments can be used to control its behavior. If the first argument is non-zero, the map marker will follow the thing with the same TID as specified. If the second argument is 1, then the map marker will not show up until the player has seen the sector it resides in. If the third argument is 1, the map marker scales relative to the automap zoom, rather than keep a constant scale.

You can use Thing_Activate and Thing_Deactivate to turn map markers on and off, due to a typo in the original implementation, dormancy is reversed: Activating hides the map marker, while Deactivating shows the map marker. Also, this feature ONLY works through ACS, meaning map markers intended to be hidden by default need to be manually "Activated" in the OPEN script. (protip: put them all in a single range and use a loop)

DECORATE definition

```
ACTOR MapMarker native
{
    +NOBLOCKMAP
    +NOGRAVITY
    +DONTSPASH
    +INVISIBLE
    Scale 0.5
    States
    {
    Spawn:
        AMRK A -1
        Stop
    }
}
```

Map spot with gravity
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9013
Class Name: MapSpotGravity
Classes: MapSpot†'MapSpotGravity

A map spot is an actor used to identify a particular spot on the map so as to have events happening there. This map spot is affected by gravity and thus always at floor level. It has pretty much nothing unique to it, it is just a way to place a TID somewhere on a map and any other actor can be used instead if appropriate.

DECORATE definition

```
ACTOR MapSpotGravity : MapSpot
{
  -NOBLOCKMAP
  -NOSECTOR
  -NOGRAVITY
}
```

Map spot

Actor type: Map spot

Game ZDoom

DoomEd Number 9001 Class Name MapSpot

Classes: MapSpot

â€“,â†’MapSpotGravity

A map spot is an actor used to identify a particular spot on the map so as to have events happening there. It has pretty much nothing unique to it, it is just a way to place a TID somewhere on a map and any other actor can be used instead if appropriate.

DECORATE definition

ACTOR MapSpot

```
{
+NOBLOCKMAP
+NOSECTOR
+NOGRAVITY
+DONTSPASH
RenderStyle None
CameraHeight 0
}
```

Point puller

Actor type: Map spot

Game: ZDoom

DoomEd Number: 5002

Class Name: PointPuller

Classes: PointPuller

A point puller pulls nearby actors towards it. This force crosses sector boundaries, decreases linearly with distance, and is felt within a circle centered on itself. The force is felt only if the point puller can see the target object. To push actors away, see PointPusher.

The sector in which the thing is placed must have the "Pusher Enabled" bit set, see sector specials.

DECORATE definition

ACTOR PointPuller

{

+NOBLOCKMAP

+INVISIBLE

+NOCLIP

}

Point pusher

Actor type: Map spot

Game: ZDoom

DoomEd Number: 5001

Class Name: PointPusher

Classes: PointPusher

A point pusher pushes nearby actors away from it. This force crosses sector boundaries, decreases linearly with distance, and is felt within a circle centered on itself. The force is felt only if the point pusher can see the target object. To pull actors in, see PointPuller.

The sector in which the thing is placed must have the "Pusher Enabled" bit set, see sector specials.

DECORATE definition

ACTOR PointPusher

```
{  
  +NOBLOCKMAP  
  +INVISIBLE  
  +NOCLIP  
}
```

Special spot
Actor type: Internal
Game: ZDoom
DoomEd Number: None
Class Name: SpecialSpot
Classes: SpecialSpot
→BossSpot
→BossTarget
→MaceSpawner

The base class used for special map points that must be part of a collection where a single element can be chosen randomly. The Icon of Sin in Doom II, D'Sparil and the firemace in Heretic all use this in different ways.

The Icon of Sin regularly spits boss cubes at a randomly-chosen BossTarget.
D'Sparil, once his mount is dead, will often teleport to a randomly-chosen BossSpot. The more wounded he is, the more likely he'll decide to teleport.
Only one firemace can be present in a level, so it will appear at a randomly-chosen MaceSpawner. Except in deathmatch, the mace only has a 75% chance of appearing at all.

DECORATE definition

```
ACTOR SpecialSpot native
{
    action native A_SpawnSingleItem(class<Actor> type, int fail_sp = 0, int fail_co = 0, int fail_dm = 0);
}
```

Teleport destination with height and gravity

Actor type: Map spot

Game: ZDoom

DoomEd Number: 9043

Class Name: TeleportDest3

Classes: TeleportDestâ†’TeleportDest2â†’TeleportDest3

A teleport destination, or derived actor, is where actors are moved by the Teleport line specials. This destination can be given a height (Z) value but is subject to gravity and will fall if the sector it is in is lowered or ascend if it is raised. It is recommended to use this one rather than TeleportDest for TeleportGroup specials

DECORATE definition

ACTOR TeleportDest3 : TeleportDest2

{

-NOGRAVITY

}

Teleport destination with height

Actor type: Map spot

Game: ZDoom

DoomEd Number: 9044

Class Name: TeleportDest2

Classes: TeleportDestâ†’TeleportDest2

â€“,â†’TeleportDest3

A teleport destination, or derived actor, is where actors are moved by the Teleport line specials. This destination can be given a height (Z) value and be used to teleport over a pit for a trap, a 3D floor, a bridge thing or similar solid obstacle.

DECORATE definition

ACTOR TeleportDest2 : TeleportDest

{

+NOGRAVITY

}

Teleport destination
Actor type: Map spot
Game: ZDoom
DoomEd Number: 14
Class Name: TeleportDest
Classes: TeleportDest
→TeleportDest2
→TeleportDest3

A teleport destination, or derived actor, is where actors are moved by the Teleport specials.

DECORATE definition

```
ACTOR TeleportDest
{
    +NOBLOCKMAP
    +NOSECTOR
    +DONTSPASH
}
```

Color setter

Actor type: Map spot

Game: ZDoom

DoomEd Number: 9038

Class Name: ColorSetter

Classes: ColorSetter

Sets the containing sector's light color using the values specified by its arguments:

Arg1: Red

Arg2: Green

Arg3: Blue

Arg4: Desaturation

This thing changes the light color of the sector it is placed in. By default, sectors have white light (red, green, blue are all 255). You can use the testcolor console command to test a color from within the game.

The 4th argument specifies how much the colors in the sector are desaturated. 0 means no desaturation at all resulting in normal colors. 255 means full desaturation, resulting in a colorized black and white display.

DECORATE definition

ACTOR ColorSetter native

```
{
+NOBLOCKMAP
+NOGRAVITY
+DONTSPASH
RenderStyle None
}
```

Fade setter
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9039
Class Name: FadeSetter
Classes: FadeSetter

Sets the containing sector's fade (fog) color using the values specified by its arguments:

Arg1: Red
Arg2: Green
Arg3: Blue

Place this in a sector to change its fade value. This controls the color that light in the sector fades to as it gets further away from the viewer. This will give the sector a 'fog' effect and can greatly enhance the mood of a level. By default, this is whatever the level's fadeto is specified as being in a MAPINFO lump, or black, if it doesn't specify a fadeto. You can use the testfade console command to test a fade from within the game. Keep in mind that the fade color is affected by a sector's light level, so a faded sector with a light value of 255 will have virtually no visible fade color, and a faded sector with a light value of 0 will be nearly a solid color.

DECORATE definition

```
ACTOR FadeSetter native
{
  +NOBLOCKMAP
  +NOGRAVITY
  +DONTSPASH
  RenderStyle None
}
```

Lower stack thing

Actor type: Map spot

Game: ZDoom

DoomEd Number: 9078

Class Name: LowerStackLookOnly

Classes: SkyViewpoint†'StackPoint†'LowerStackLookOnly

The lower stack thing is used to make stacked sectors. It goes into the lower stack sector. It takes one argument, the opacity of sector: 0 is invisible, 255 is fully opaque. It is to be used with a UpperStackLookOnly with the same TID, and both sectors should have the same shape and size.

Thing based sector stacks should no longer be used. A more robust method to define them exists with Sector_SetPortal

DECORATE definition

ACTOR LowerStackLookOnly : StackPoint {}

Sky picker

Actor type: Map spot

Game: ZDoom

DoomEd Number: 9081

Class Name: SkyPicker

Classes: SkyPicker

The sky picker is used to choose which skybox to use in a sector. It takes 2 arguments.

The first argument is the tid of the SkyViewpoint. If this argument is zero, then the sector will use the default normal sky texture (as defined in MAPINFO) rather than one of the level's skybox. If it is not zero, the SkyViewpoint with the corresponding tid will be used.

The second argument has 3 options:

0: Apply the skybox on both floor and ceiling (if both use the F_SKY1 flat)

1: Apply the skybox only to the ceiling

2: Apply the skybox only to the floor

This way it is possible to place two sky pickers in a single sector, one to chose a skybox for the floor and the other to chose a skybox for the ceiling.

DECORATE definition

ACTOR SkyPicker native

```
{
+NOSECTOR
+NOBLOCKMAP
+NOGRAVITY
+DONTSPASH
}
```

Skybox camera
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9080
Class Name: SkyViewpoint
Classes: Actor→SkyViewpoint
→SkyCamCompat
→StackPoint
→LowerStackLookOnly
→UpperStackLookOnly

The sky viewpoint is used to make skyboxes, it acts as the camera into the skybox. If, when the map is initialized, its TID is zero, it will replace the sky everywhere in the map. If its TID is not zero at that time, then corresponding SkyPickers with the same TID must be placed in the sectors where the sky is to be replaced by the one in the skybox. If the TID changes after map initialization, the skies are not affected.

The sky viewpoint takes one argument: visibility. The visibility for a skybox is equal to one-fourth the value of the first argument, so if you give it a value of 32 the visibility in the skybox will be 8.

DECORATE definition

```
ACTOR SkyViewpoint native
{
    +NOSECTOR
    +NOBLOCKMAP
    +NOGRAVITY
    +DONTSPASH
}
```

Sector Action: "Actor Enters Sector"
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9998
Class Name: SecActEnter
Classes: SectorAction→SecActEnter

This is a ZDoom-specific thing that triggers its special when an actor enters the sector it is present in. The following actor flags, if set, control the trigger-ability of the thing:

AMBUSH → monsters can trigger the special.
DORMANT → projectiles can trigger the special.
FRIENDLY → players cannot trigger the special.
STANDSTILL → the special will be triggered only once.

See also: Thing executed specials

DECORATE definition

ACTOR SecActEnter : SectorAction native {}

Sector Action: "Actor Hits Ceiling"
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9996
Class Name: SecActHitCeil
Classes: SectorAction→SecActHitCeil

This is a ZDoom-specific thing that triggers its special when an actor hits the ceiling of the sector it is present in. The following actor flags, if set, control the trigger-ability of the thing:

AMBUSH → monsters can trigger the special.
DORMANT → projectiles can trigger the special.
FRIENDLY → players cannot trigger the special.
STANDSTILL → the special will be triggered only once.
See also: Thing executed specials

DECORATE definition

ACTOR SecActHitCeil : SectorAction native {}

Sector Action: "Actor Hits Fake Floor"

Actor type: Map spot

Game: ZDoom

DoomEd Number: 9989

Class Name: SecActHitFakeFloor

Classes: SectorAction→SecActHitFakeFloor

This is a ZDoom-specific thing that triggers its special when an actor hits the fake floor of the sector it is present in. The following actor flags, if set, control the trigger-ability of the thing:

AMBUSH → monsters can trigger the special.

DORMANT → projectiles can trigger the special.

FRIENDLY → players cannot trigger the special.

STANDSTILL → the special will be triggered only once.

See also: Thing executed specials

DECORATE definition

ACTOR SecActHitFakeFloor : SectorAction native {}

Sector Action: "Actor Hits Floor"
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9999
Class Name; SecActHitFloor
Classes: SectorAction→SecActHitFloor

This is a ZDoom-specific thing that triggers its special when an actor hits the floor of the sector it is present in. The following actor flags, if set, control the trigger-ability of the thing:

AMBUSH → monsters can trigger the special.
DORMANT → projectiles can trigger the special.
FRIENDLY → players cannot trigger the special.
STANDSTILL → the special will be triggered only once.
See also: Thing executed specials

DECORATE definition

ACTOR SecActHitFloor : SectorAction native {}

Sector Action: "Actor Leaves Sector"
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9997
Class Name: SecActExit
Classes: SectorAction→SecActExit

This is a ZDoom-specific thing that triggers its special when an actor leaves the sector it is present in. The following actor flags, if set, control the trigger-ability of the thing:

AMBUSH → monsters can trigger the special.
DORMANT → projectiles can trigger the special.
FRIENDLY → players cannot trigger the special.
STANDSTILL → the special will be triggered only once.
See also: Thing executed specials

DECORATE definition

ACTOR SecActExit : SectorAction native {}

Sector Action: "Eyes Go Above Ceiling"

Actor type: Map spot

Game: ZDoom

DoomEd Number: 9982

Class Name: SecActEyesAboveC

Classes: SectorAction†'SecActEyesAboveC

This is a Transfer_Heights-specific thing that triggers its special when a player's viewpoint go above the fake ceiling of the sector it is in.

See also: Thing executed specials

DECORATE definition

ACTOR SecActEyesAboveC : SectorAction native {}

Sector Action: "Eyes Go Above Surface"
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9992
Class Name: SecActEyesSurface
Classes: SectorAction†’SecActEyesSurface

This is a Transfer_Heights-specific thing that triggers its special when a player's viewpoint go above the fake floor of the sector it is in.

See also: Thing executed specials

DECORATE definition

ACTOR SecActEyesSurface : SectorAction native {}

Sector Action: "Eyes Go Below Ceiling"

Actor type: Map spot

Game: ZDoom

DoomEd Number: 9983

Class Name: SecActEyesBelowC

Classes: SectorAction†'SecActEyesBelowC

This is a Transfer_Heights-specific thing that triggers its special when a player's viewpoint go below the fake ceiling of the sector it is in.

See also: Thing executed specials

DECORATE definition

ACTOR SecActEyesBelowC : SectorAction native {}

Sector Action: "Eyes Go Below Surface"

Actor type: Map spot

Game: ZDoom

DoomEd Number: 9993

Class Name: SecActEyesDive

Classes: SectorAction†'SecActEyesDive

This is a Transfer_Heights-specific thing that triggers its special when a player's viewpoint go below the fake floor of the sector it is in.

See also: Thing executed specials

DECORATE definition

ACTOR SecActEyesDive : SectorAction native {}

Sector Action: "Player Uses Sector"

Actor type: Map spot

Game: ZDoom

DoomEd Number: 9995

Class Name: SecActUse

Classes: SectorAction†'SecActUse

This is a ZDoom-specific thing that triggers its special when a player presses the "use" key anywhere in the sector except within activation range of a linedef with a special. (If the linedef already has a use-activated special, then this special is activated instead.)

See also: Thing executed specials

DECORATE definition

ACTOR SecActUse : SectorAction native {}

Sector Action: "Player Uses Wall"
Actor type: Map spot
Game: ZDoom
DoomEd Number: 9994
Class Name: SecActUseWall
Classes: SectorAction†'SecActUseWall

This is a ZDoom-specific thing that triggers its special when a player uses one of the sector's linedefs that doesn't, itself, have a special. (If the linedef already has a use-activated special, then this special is activated instead.)

See also: Thing executed specials

DECORATE definition

```
ACTOR SecActUseWall : SectorAction native {}
```

Blood splash
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: BloodSplash
Classes: BloodSplash

Small blood splashes.

DECORATE definition

ACTOR BloodSplash

```
{
  Radius 2
  Height 4
  +NOBLOCKMAP
  +MISSILE
  +DROPOFF
  +NOTELEPORT
  +LOWGRAVITY
  +CANNOTPUSH
  +DONTSPASH
  +DONTBLAST
  States
  {
    Spawn:
      BSPH ABC 8
      BSPH D 16
      Stop
    Death:
      BSPH D 10
      Stop
  }
}
```

Blood splash
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: BloodSplashBase
Classes: BloodSplashBase

Normal (large) blood splashes.

DECORATE definition

```
ACTOR BloodSplashBase
{
    +NOBLOCKMAP
    +NOCLIP
    +NOGRAVITY
    +DONTSPASH
    +DONTBLAST
    States
    {
    Spawn:
        BSPH EFGHIJK 5
        Stop
    }
}
```

Blood splat
Actor type: Gibs
Game: ZDoom
DoomEd Number: None
Class Name: Blood
Spawn ID: 130
Identifier: T_BLOOD
Classes: Blood

The blood splat actor is spawned when an actor is hit and is able to bleed. In IWADs other than Doom's, the blood sprites are actually named BLOOD* and are renamed at load-time by ZDoom.

Blood has special functionality hardcoded into the engine. The blood actor will start at different frames depending on how much damage is done that caused the bleeding.

If Damage is less than 9, the actor begins at the third state.

If Damage is between 9 to 12 (inclusive), it begins at the second state.

If Damage is above 12, it begins at the first state.

If the game is Strife, and damage is greater than 13, it shifts to the Spray state (if it exists), otherwise it adds 2 to the damage check for the previous checks.

DECORATE definition

ACTOR Blood

```
{
  Mass 5
  +NOBLOCKMAP
  +NOTELEPORT
  +ALLOWPARTICLES
  States
  {
    Spawn:
      BLUD CBA 8
      Stop
    Spray:
      SPRY ABCDEF 3
      SPRY G 2
      Stop
  }
}
```

Blood splatter
Actor type: Gibs
Game: ZDoom
DoomEd Number: None
Class Name: BloodSplatter
Classes: BloodSplatter

The blood splatter that is spawned instead of the Blood actor when an actor is hit and is able to bleed and the inflicting weapon or projectile has the +BLOODSPLATTER flag; all the predefined Heretic and Hexen projectiles have this flag implicitly set by the engine and many of the weapons, such as the Mace of Contrition and Hammer of Retribution, set it explicitly. In IWADs other than Doom's, the blood sprites are actually named BLOD* and are renamed at load-time by ZDoom.

DECORATE definition

```
ACTOR BloodSplatter
{
    Radius 2
    Height 4
    +NOBLOCKMAP
    +MISSILE
    +DROPOFF
    +NOTELEPORT
    +CANNOTPUSH
    +ALLOWPARTICLES
    Mass 5
    States
    {
    Spawn:
        BLUD CBA 8
        Stop
    Death:
        BLUD A 6
        Stop
    }
}
```

Dirt
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: Dirt1
Spawn ID: 44
Identifier: T_DIRT1
Classes: Dirt1

Dirt, originally from Hexen, generalized by ZDoom to all games. However, sprites for this dirt aren't included in ZDoom.pk3, and must be provided. In Hexen.wad, the sprite is actually named ROCKD0 and is renamed at load-time by ZDoom.

DECORATE definition

```
ACTOR Dirt1
{
  +NOBLOCKMAP
  +DROPOFF
  +MISSILE
  +NOTELEPORT
  States
  {
    Spawn:
      ROKK D 20
      Loop
    Death:
      ROKK D 10
      Stop
  }
}
```

Dirt
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: Dirt2
Spawn ID: 45
Identifier: T_DIRT2
Classes: Dirt2

Dirt, originally from Hexen, generalized by ZDoom to all games. However, sprites for this dirt aren't included in ZDoom.pk3, and must be provided. In Hexen.wad, the sprite is actually named ROCKE0 and is renamed at load-time by ZDoom.

DECORATE definition

```
ACTOR Dirt2
{
  +NOBLOCKMAP
  +DROPOFF
  +MISSILE
  +NOTELEPORT
  States
  {
    Spawn:
      ROKK E 20
      Loop
    Death:
      ROKK E 10
      Stop
  }
}
```

Dirt3
Actor type: SFX
Game: Zdoom
DoomEd Number: None
Class Name: Dirt3
Spawn ID: 46
Identifier: T_DIRT3
Classes: Dirt3

Dirt, originally from Hexen, generalized by ZDoom to all games. However, sprites for this dirt aren't included in ZDoom.pk3, and must be provided. In Hexen.wad, the sprite is actually named ROCKF0 and is renamed at load-time by ZDoom.

DECORATE definition

```
ACTOR Dirt3
{
  +NOBLOCKMAP
  +DROPOFF
  +MISSILE
  +NOTELEPORT
  States
  {
    Spawn:
      ROKK F 20
      Loop
    Death:
      ROKK F 10
      Stop
  }
}
```


Dirt4
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: Dirt4
Spawn ID: 47
Identifier: T_DIRT4
Classes: Dirt4

Dirt, originally from Hexen, generalized by ZDoom to all games. However, sprites for this dirt aren't included in ZDoom.pk3, and must be provided. In Hexen.wad, the sprite is actually named ROCKG0 and is renamed at load-time by ZDoom.

DECORATE definition

```
ACTOR Dirt4
{
  +NOBLOCKMAP
  +DROPOFF
  +MISSILE
  +NOTELEPORT
  States
  {
    Spawn:
      ROKK G 20
      Loop
    Death:
      ROKK G 10
      Stop
  }
}
```

Dirt5
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: Dirt5
Spawn ID: 48
Identifier: T_DIRT5
Classes: Dirt5

Dirt, originally from Hexen, generalized by ZDoom to all games. However, sprites for this dirt aren't included in ZDoom.pk3, and must be provided. In Hexen.wad, the sprite is actually named ROCKH0 and is renamed at load-time by ZDoom.

DECORATE definition

```
ACTOR Dirt5
{
  +NOBLOCKMAP
  +DROPOFF
  +MISSILE
  +NOTELEPORT
  States
  {
    Spawn:
      ROKK H 20
      Loop
    Death:
      ROKK H 10
      Stop
  }
}
```

Dirt6
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: Dirt6
Spawn ID: 49
Identifier: T_DIRT6
Classes: Dirt6

Dirt, originally from Hexen, generalized by ZDoom to all games. However, sprites for this dirt aren't included in ZDoom.pk3, and must be provided. In Hexen.wad, the sprite is actually named ROCKI0 and is renamed at load-time by ZDoom.

DECORATE definition

```
ACTOR Dirt6
{
    +NOBLOCKMAP
    +DROPOFF
    +MISSILE
    +NOTELEPORT
    States
    {
        Spawn:
            ROKK I 20
            Loop
        Death:
            ROKK I 10
            Stop
    }
}
```

Glass shard base class
Actor type: Internal
Game: ZDoom
DoomEd Number: None
Class Name: GlassShard
Classes: GlassShard
→ SGShard1
→ SGShard2
→ SGShard3
→ SGShard4
→ SGShard5
→ SGShard6
→ SGShard7
→ SGShard8
→ SGShard9
→ SGShard0

The abstract class used to define the code shared by all breakable glasses. It is never used directly; instead, its children classes are called.

DECORATE definition

```
ACTOR GlassShard native
{
    Radius 5
    Mass 5
    Projectile
    -ACTIVATEPCROSS
    -ACTIVATEIMPACT
    BounceType "HexenCompat"
    BounceFactor 0.3
}
```

Item fog
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: ItemFog
Classes: ItemFog

Fog for respawning Doom and Strife items.

DECORATE definition

```
ACTOR ItemFog
{
    +NOBLOCKMAP
    +NOGRAVITY
    States
    {
        Spawn:
            IFOG ABABCDE 6 Bright
        Stop
    }
}
```

Lava smoke
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: LavaSmoke
Classes: LavaSmoke

Smoke for lava or 'hot' rocks.

DECORATE definition

```
ACTOR LavaSmoke
{
    +NOBLOCKMAP
    +NOCLIP
    +NOGRAVITY
    +DONTSPASH
    RenderStyle Translucent
    DefaultAlpha
    States
    {
    Spawn:
        LVAS GHIJK 5 Bright
        Stop
    }
}
```

Lava splash
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: LavaSplash
Classes: LavaSplash

Splash of lava (normal sized).

DECORATE definition

```
ACTOR LavaSplash
{
    +NOBLOCKMAP
    +NOCLIP
    +NOGRAVITY
    +DONTSPASH
    +DONTBLAST
    States
    {
    Spawn:
        LVAS ABCDEF 5 Bright
        Stop
    }
}
```

Pool of crushed gibbs
Actor type: Gibs
Game: ZDoom
DoomEd Number: None
Class Name: RealGibs
Classes: RealGibs
â€¢â†’Gibs

The small pool of blood that replaces a corpse crushed by a crusher or a door.

Sprites for this actor are found in the Doom and Hexen IWADs. In Hexen.wad, the sprite is actually named GIBSA0 and is renamed at load-time by ZDoom.

DECORATE definition

```
ACTOR RealGibs
{
  +DROPOFF
  +CORPSE
  +NOTELEPORT
  +DONTGIB
  States
  {
    Spawn:
      Goto GenericCrush
  }
}
```


Rock1
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: Rock1
Spawn ID: 41
Identifier: T_ROCK1
Classes: Rock1

A rock, originally from Hexen, generalized by ZDoom to all games. However, sprites for this rock aren't included in ZDoom.pk3, and must be provided. In Hexen.wad, the sprite is actually named ROCKA0 and is renamed at load-time by ZDoom.

DECORATE definition

```
ACTOR Rock1
{
    +NOBLOCKMAP
    +DROPOFF
    +MISSILE
    +NOTELEPORT
    States
    {
        Spawn:
            ROKK A 20
            Loop
        Death:
            ROKK A 10
            Stop
    }
}
```

Rock2
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: Rock2
Spawn ID: 42
Identifier: T_ROCK2
Classes: Rock2

A rock, originally from Hexen, generalized by ZDoom to all games. However, sprites for this rock aren't included in ZDoom.pk3, and must be provided. In Hexen.wad, the sprite is actually named ROCKB0 and is renamed at load-time by ZDoom.

DECORATE definition

```
ACTOR Rock2
{
    +NOBLOCKMAP
    +DROPOFF
    +MISSILE
    +NOTELEPORT
    States
    {
        Spawn:
            ROKK B 20
            Loop
        Death:
            ROKK B 10
            Stop
    }
}
```

Rock3
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name Rock3
Spawn ID: 43
Identifier:T_ROCK3
Classes: Rock3

A rock, originally from Hexen, generalized by ZDoom to all games. However, sprites for this rock aren't included in ZDoom.pk3, and must be provided. In Hexen.wad, the sprite is actually named ROCKC0 and is renamed at load-time by ZDoom.

DECORATE definition

```
ACTOR Rock3
{
+NOBLOCKMAP
+DROPOFF
+MISSILE
+NOTELEPORT
States
{
Spawn:
    ROKK C 20
    Loop
Death:
    ROKK C 10
    Stop
}
}
```

Slime chunk
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SlimeChunk
Classes: SlimeChunk

Small nukage chunks.

DECORATE definition

ACTOR SlimeChunk

```
{  
  Radius 2  
  Height 4  
  +NOBLOCKMAP  
  +MISSILE  
  +DROPOFF  
  +NOTELEPORT  
  +LOWGRAVITY  
  +CANNOTPUSH  
  +DONTSPASH  
  States  
  {  
    Spawn:  
      SLIM ABCD 8  
      Stop  
    Death:  
      SLIM D 6  
      Stop  
  }  
}
```

Slime splash
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SlimeSplash
Classes: SlimeSplash

Normal (large) nukage splashes.

DECORATE definition

```
ACTOR SlimeSplash
{
    +NOBLOCKMAP
    +NOCLIP
    +NOGRAVITY
    +DONTSPASH
    States
    {
        Spawn:
            SLIM EFGH 6
        Stop
    }
}
```

Sludge chunk
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SludgeChunk
Classes: SludgeChunk

Small brown sludge splashes.

DECORATE definition

ACTOR SludgeChunk

```
{
  Radius 2
  Height 4
  +NOBLOCKMAP
  +MISSILE
  +DROPOFF
  +NOTELEPORT
  +LOWGRAVITY
  +CANNOTPUSH
  +DONTSPASH
  States
  {
    Spawn:
      SLDG ABCD 8
      Stop
    Death:
      SLDG D 6
      Stop
  }
}
```

Sludge splash
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SludgeSplash
Classes: SludgeSplash

Normal (large) sludge splashes.

DECORATE definition

ACTOR SludgeSplash

```
{
  +NOBLOCKMAP
  +NOCLIP
  +NOGRAVITY
  +DONTSPASH
  States
  {
    Spawn:
      SLDG EFGH 6
      Stop
  }
}
```

Spark generator
Actor type: SFX
Game: ZDoom
DoomEd Number: 9026
Class Name: Spark
Classes: Spark

A spark generator creates spark when activated. The number of sparks created can be set as its first argument; by default it is 32. Its angle is the direction in which the sparks will be thrown.

DECORATE definition

ACTOR Spark native

```
{  
  +NOSECTOR  
  +NOBLOCKMAP  
  +NOGRAVITY  
  +DONTSPASH  
}
```


Stained glass shard
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SGShard0
Spawn ID: 63
Identifier: T_STAINEDGLASS0
Classes: GlassShard → SGShard0

A shard of broken stained glass window.

DECORATE definition

```
ACTOR SGShard0 : GlassShard
{
    States
    {
        Spawn:
            SGSB E 4
        Loop
        Death:
            SGSB E 30
        Stop
    }
}
```

Stained glass shard
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SGShard1
Spawn ID: 54
Identifier: T_STAINEDGLASS1

Classes: GlassShard → SGShard1
A shard of broken stained glass window.

DECORATE definition

```
ACTOR SGShard1 : GlassShard
{
    States
    {
        Spawn:
            SGSA ABCDE 4
        Loop
        Death:
            SGSA E 30
        Stop
    }
}
```

Stained glass shard
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SGShard2
Spawn ID: 55
Identifier: T_STAINEDGLASS2
Classes: GlassShard → SGShard2

A shard of broken stained glass window.

DECORATE definition

```
ACTOR SGShard2 : GlassShard
{
  States
  {
    Spawn:
      SGSA FGHIJ 4
    Loop
    Death:
      SGSA J 30
    Stop
  }
}
```

Stained glass shard
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SGShard3
Spawn ID: 56
Identifier: T_STAINEDGLASS3
Classes: GlassShard â†’ SGShard3

A shard of broken stained glass window.

DECORATE definition

```
ACTOR SGShard3 : GlassShard
{
    States
    {
        Spawn:
            SGSA KLMNO 4
        Loop
        Death:
            SGSA O 30
        Stop
    }
}
```

Stained glass shard
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SGShard4
Spawn ID: 57
Identifier: T_STAINEDGLASS4
Classes: GlassShard → SGShard4

A shard of broken stained glass window.

DECORATE definition

```
ACTOR SGShard4 : GlassShard
{
    States
    {
        Spawn:
            SGSA PQRST 4
        Loop
        Death:
            SGSA T 30
        Stop
    }
}
```

Stained glass shard
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SGShard5
Spawn ID: 58
Identifier: T_STAINEDGLASS5
Classes: GlassShard â†’ SGShard5

A shard of broken stained glass window.

DECORATE definition

```
ACTOR SGShard5 : GlassShard
{
    States
    {
        Spawn:
            SGSA UVWXY 4
        Loop
        Death:
            SGSA Y 30
        Stop
    }
}
```

Stained glass shard
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SGShard6
Spawn ID: 59
Identifier: T_STAINEDGLASS6
Classes: GlassShard → SGShard6

A shard of broken stained glass window.

DECORATE definition

```
ACTOR SGShard6 : GlassShard
{
    States
    {
        Spawn:
            SGSB A 4
        Loop
        Death:
            SGSB A 30
        Stop
    }
}
```

Stained glass shard
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SGShard7
Spawn ID: 60
Identifier: T_STAINEDGLASS7
Classes: GlassShard → SGShard7

A shard of broken stained glass window.

DECORATE definition

```
ACTOR SGShard7 : GlassShard
{
    States
    {
        Spawn:
            SGSB B 4
        Loop
        Death:
            SGSB B 30
        Stop
    }
}
```


Stained glass shard
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SGShard8
Spawn ID: 61
Identifier: T_STAINEDGLASS8
Classes: GlassShard → SGShard8

A shard of broken stained glass window.

DECORATE definition

```
ACTOR SGShard8 : GlassShard
{
    States
    {
        Spawn:
            SGSB C 4
        Loop
        Death:
            SGSB C 30
        Stop
    }
}
```

Stained glass shard
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: SGShard9
Spawn ID: 62
Identifier: T_STAINEDGLASS9
Classes: GlassShard → SGShard9

A shard of broken stained glass window.

DECORATE definition

```
ACTOR SGShard9 : GlassShard
{
    States
    {
        Spawn:
            SGSB D 4
        Loop
        Death:
            SGSB D 30
        Stop
    }
}
```

Teleport fog
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: TeleportFog
Classes: TeleportFog

The Teleport Fog is a special effect flash that is spawned when an actor teleports and, by default, when an actor morphs or unmorphs. Teleport Fogs set the teleporting actor as their target, so actor manipulation via pointers is possible. This includes when an actor (un)morphs.

DECORATE definition

```
ACTOR TeleportFog native
{
+NOBLOCKMAP
+NOTELEPORT
+NOGRAVITY
RenderStyle Add
States
{
Spawn:
    TFOG ABABCDEFBGHIJ 6 Bright
    Stop

Raven:
    TELE ABCDEFGHGFEDC 6 Bright
    Stop

Strife:
    TFOG ABCDEFEDCB 6 Bright
    Stop
}
}
```

Water splash
Actor type: SFX
Game ZDoom
DoomEd Number: None
Class Name: WaterSplashBase
Classes: WaterSplashBase

Normal (large) water splashes.

DECORATE definition

```
ACTOR WaterSplashBase
{
    +NOBLOCKMAP
    +NOCLIP
    +NOGRAVITY
    +DONTSPASH
    +DONTBLAST
    States
    {
    Spawn:
        SPSH EFGHIJK 5
        Stop
    }
}
```

Water splash
Actor type: SFX
Game: ZDoom
DoomEd Number: None
Class Name: WaterSplash
Classes: WaterSplash
Small water splashes.

DECORATE definition

ACTOR WaterSplash

```
{  
  Radius 2  
  Height 4  
  +NOBLOCKMAP  
  +MISSILE  
  +DROPOFF  
  +NOTELEPORT  
  +LOWGRAVITY  
  +CANNOTPUSH  
  +DONTSPASH  
  +DONTBLAST  
  States  
  {  
    Spawn:  
      SPSH ABC 8  
      SPSH D 16  
      Stop  
    Death:  
      SPSH D 10  
      Stop  
  }  
}
```

Backpack

Actor type: Ammo

Game: Doom

DoomEd Number: 8

Class Name: Backpack

Spawn ID: 144

Identifier: T_BACKPACK

Classes: Inventoryâ†’BackpackItemâ†’Backpack

The backpack contains a sample of each ammunition type (the quantity of which depends on the ammunition, not on the backpack) and extends the carrying capacity of each ammunition type (again, this is a factor of the ammunition).

DECORATE definition

ACTOR Backpack : BackpackItem

```
{
  Height 26
  Inventory.PickupMessage "$GOTBACKPACK" // "Picked up a backpack full of ammo!"
  States
  {
    Spawn:
      BPAK A -1
      Stop
  }
}
```

Cell
Actor type: Ammo
Game: Doom
DoomEd Number: 2047
Class Name: Cell
Spawn ID: 75
Identifier: T_CELL
Classes: Inventory→Ammo→Cell
→CellPack

Cell is an ammo type for Doom, used most commonly for energy weapons such as the BFG9000 and the plasma rifle.

DECORATE definition

```
ACTOR Cell : Ammo
{
    Inventory.PickupMessage "$GOTCELL" // "Picked up an energy cell."
    Inventory.Amount 20
    Inventory.MaxAmount 300
    Ammo.BackpackAmount 20
    Ammo.BackpackMaxAmount 600
    Inventory.Icon "CELLA0"
    States
    {
        Spawn:
            CELL A -1
            Stop
    }
}
```

Cell pack

Actor type: Ammo

Game: Doom

DoomEd Number: 17

Class Name: CellPack

Spawn ID: 142

Identifier: T_BATTERY

Classes: Inventoryâ†’Ammoâ†’Cellâ†’CellPack

A cell pack gives an amount of energy ammo five times as large as a cell does.

DECORATE definition

ACTOR CellPack : Cell

```
{
  Inventory.PickupMessage "$GOTCELLBOX" // "Picked up an energy cell pack."
  Inventory.Amount 100
  States
  {
    Spawn:
      CELP A -1
      Stop
  }
}
```


Clip
Actor type: Ammo
Game: Doom
DoomEd Number: 2007
Class Name: Clip
Spawn ID: 11
Identifier: T_CLIP
Classes: Inventory→Ammo→Clip
→ClipBox

Clip is the ammunition for both pistols and chainguns in Doom. A normal clip contains 10 bullets. Not to be confused with Strife's own clip.

DECORATE definition

```
ACTOR Clip : Ammo
{
    Inventory.PickupMessage "$GOTCLIP" // "Picked up a clip."
    Inventory.Amount 10
    Inventory.MaxAmount 200
    Ammo.BackpackAmount 10
    Ammo.BackpackMaxAmount 400
    Inventory.Icon "CLIPA0"
    States
    {
        Spawn:
            CLIP A -1
            Stop
    }
}
```

Clip box

Actor type: Ammo

Game: Doom

DoomEd Number: 2048

Class Name: ClipBox

Spawn ID: 139

Identifier: T_AMMOBOX

Classes: Inventoryâ†’Ammoâ†’Clipâ†’ClipBox

A box of bullets gives five times the amount of bullets a clip does.

DECORATE definition

ACTOR ClipBox : Clip

```
{
  Inventory.PickupMessage "$GOTCLIPBOX" // "Picked up a box of bullets."
  Inventory.Amount 50
  States
  {
    Spawn:
      AMMO A -1
      Stop
  }
}
```

Rocket
Actor type: Ammo
Game: Doom
DoomEd Number: 2010
Class Name: RocketAmmo
Spawn ID: 140
Identifier: T_ROCKETAMMO
Classes: Inventory→Ammo→RocketAmmo
→RocketBox

Ammunition used by the player's rocket launcher in Doom.

DECORATE definition

```
ACTOR RocketAmmo : Ammo
{
    Inventory.PickupMessage "$GOTROCKET" // "Picked up a rocket."
    Inventory.Amount 1
    Inventory.MaxAmount 50
    Ammo.BackpackAmount 1
    Ammo.BackpackMaxAmount 100
    Inventory.Icon "ROCKA0"
    States
    {
        Spawn:
            ROCK A -1
            Stop
    }
}
```

Rocket box
Actor type: Ammo
Game: Doom
DoomEd Number: 2046
Class Name: RocketBox
Spawn ID: 141
Identifier: T_ROCKETBOX
Classes: Inventoryâ†’Ammoâ†’RocketAmmoâ†’RocketBox

A box containing five rockets.

DECORATE definition

```
ACTOR RocketBox : RocketAmmo
{
    Inventory.PickupMessage "$GOTROCKBOX" // "Picked up a box of rockets."
    Inventory.Amount 5
    States
    {
        Spawn:
            BROK A -1
            Stop
    }
}
```

Shells
Actor type: Ammo
Game: Doom
DoomEd Number: 2008
Class Name: Shell
Spawn ID: 12
Identifier: T_SHELLS
Classes: Inventory→Ammo→Shell
→ShellBox

Shell ammo for the Doom shotgun and super shotgun. Gives four shells when picked up.

DECORATE definition

```
ACTOR Shell : Ammo
{
    Inventory.PickupMessage "$GOTSHELLS" // "Picked up 4 shotgun shells."
    Inventory.Amount 4
    Inventory.MaxAmount 50
    Ammo.BackpackAmount 4
    Ammo.BackpackMaxAmount 100
    Inventory.Icon "SHELA0"
    States
    {
        Spawn:
            SHEL A -1
            Stop
    }
}
```

Shell box

Actor type: Ammo

Game: Doom

DoomEd Number: 2049

Class Name: ShellBox

Spawn ID: 143

Identifier: T_SHELLBOX

Classes: Inventoryâ†’Ammoâ†’Shellâ†’ShellBox

A box of shells, for use with the Shotgun or SuperShotgun. Gives five times the amount of ammo from the smaller shell pickup.

DECORATE definition

ACTOR ShellBox : Shell

```
{
  Inventory.PickupMessage "$GOTSHELLBOX" // "Picked up a box of shotgun shells."
  Inventory.Amount 20
  States
  {
    Spawn:
      SBOX A -1
      Stop
  }
}
```

Cacodemon corpse
Actor type: Gore
Game: Doom
DoomEd Number: 22
Class Name: DeadCacodemon
Classes: Cacodemon†’DeadCacodemon

Corpse of a Cacodemon.

DECORATE definition

```
ACTOR DeadCacodemon : Cacodemon
{
  Skip_Super
  States
  {
    Spawn:
      Goto Super::Death+5
  }
}
```

Demon corpse
Actor type: Gore
Game: Doom
DoomEd Number: 21
Class Name: DeadDemon
Classes: Demon+DeadDemon

Corpse of a demon.

DECORATE definition

```
ACTOR DeadDemon : Demon
{
    Skip_Super
    States
    {
        Spawn:
            Goto Super::Death+5
    }
}
```


Imp corpse
Actor type: Gore
Game: Doom
DoomEd Number: 20
Class Name: DeadDoomImp
Classes: DoomImp†'DeadDoomImp

A corpse of an imp.

DECORATE definition

```
ACTOR DeadDoomImp : DoomImp
{
    Skip_Super
    States
    {
        Spawn:
            Goto Super::Death+4
    }
}
```

Lost soul corpse

Actor type: Gore

Game: Doom

DoomEd Number: 23

Class Name: DeadLostSoul

Classes: LostSoul†'DeadLostSoul

Corpse of a lost soul. Considering that the lost soul removes itself when it dies, there really wasn't much point in id including this thing, but they did anyway. (This is a holdover from early alpha versions of Doom where the lost souls left a corpse.)

DECORATE definition

ACTOR DeadLostSoul : LostSoul

```
{
  Skip_Super
  States
  {
    Spawn:
      Goto Super::Death+5
  }
}
```

Space marine corpse
Actor type: Gore
Game: Doom
DoomEd Number: 15
Class Name: DeadMarine

Classes: DeadMarine

Corpse of a space marine.

DECORATE definition

ACTOR DeadMarine

```
{  
  States  
  {  
    Spawn:  
      PLAY N -1  
      Stop  
  }  
}
```

Former human corpse
Actor type: Gore
Game: Doom
DoomEd Number: 19
Class Name: DeadShotgunGuy
Classes: ShotgunGuy†'DeadShotgunGuy

Corpse of a former human sergeant.

DECORATE definition

ACTOR DeadShotgunGuy : ShotgunGuy

```
{  
  Skip_Super  
  DropItem None  
  States  
  {  
    Spawn:  
      Goto Super::Death+4  
  }  
}
```

Former human corpse
Actor type: Gore
Game: Doom
DoomEd Number: 18
Class Name: DeadZombieMan
Classes: ZombieManâ†’DeadZombieMan

Corpse of a former human trooper.

DECORATE definition

ACTOR DeadZombieMan : ZombieMan

```
{  
  Skip_Super  
  DropItem None  
  States  
  {  
    Spawn:  
      Goto Super::Death+4  
  }  
}
```

Former human corpse
Actor type: Gore
Game: Doom
DoomEd Number: 18
Class Name: DeadZombieMan
Classes: ZombieManâ†’DeadZombieMan

Corpse of a former human trooper.

DECORATE definition

```
ACTOR GibbedMarine
{
  States
  {
    Spawn:
      PLAY W -1
      Stop
  }
}
```

Gibbed space marine corpse

Actor type: Gore

Game: Doom

DoomEd Number: 12

Class Name: GibbedMarineExtra

Classes: GibbedMarineâ†’GibbedMarineExtra

A second item of bloody space marine gibbs. It is exactly the same as the first one.

DECORATE definition

ACTOR GibbedMarineExtra : GibbedMarine {}

Arachnotron plasma
Actor type: Explosive
Game: Doom 2
DoomEd Number: None
Class Name: ArachnotronPlasma
Spawn ID: 129
Identifier T_ARACHNOTRONPLASMA

Classes: ArachnotronPlasma

The projectile fired by an Arachnotron.

DECORATE definition

ACTOR ArachnotronPlasma

```
{
  Radius 13
  Height 8
  Speed 25
  Damage 5
  Projectile
  +RANDOMIZE
  RenderStyle Add
  Alpha 0.75
  SeeSound "baby/attack"
  DeathSound "baby/shotx"
  States
  {
  Spawn:
    APLS AB 5 Bright
    Loop
  Death:
    APBX ABCDE 5 Bright
    Stop
  }
}
```


Arch-Vile fire
Actor type: Explosive
Game: Doom 2
DoomEd Number: None
Class Name: ArchvileFire
Spawn ID: 98
Identifier:T_TEMPLARGEFLAME

Classes: ArchvileFire

The temporary flame sprite which follows an Archvile's target while it attacks. Note that this actor does not shoot other actors or players in the air. For that use A_VileAttack.

This is the only standard actor in Doom that calls an action function in its spawn state without looping back to it. As a result, since action functions are called only on state changes and spawning is not a state change, A_StartFire is never actually executed.

DECORATE definition

```
ACTOR ArchvileFire
{
    +NOBLOCKMAP
    +NOGRAVITY
    RenderStyle Add
    Alpha 1
    States
    {
    Spawn:
        FIRE A 2 Bright A_StartFire
        FIRE BAB 2 Bright A_Fire
        FIRE C 2 Bright A_FireCrackle
        FIRE BCBCDCDCDEDED 2 Bright A_Fire
        FIRE E 2 Bright A_FireCrackle
        FIRE FEFEFGHGHGH 2 Bright A_Fire
        Stop
    }
}
```

Baron of Hell plasma
Actor type: Explosive
Game: Doom
DoomEd Number: None
Class Name: BaronBall
Spawn ID: 154
Identifier:T_BARONBALL
Classes: BaronBall

The green energy projectile fired from a Baron of Hell or a Hell Knight.

DECORATE definition

ACTOR BaronBall

```
{
  Radius 6
  Height 16
  Speed 15
  FastSpeed 20
  Damage 8
  Projectile
  +RANDOMIZE
  RenderStyle Add
  Alpha 1
  SeeSound "baron/attack"
  DeathSound "baron/shotx"
  Decal "BaronScorch"
  States
  {
    Spawn:
      BAL7 AB 4 Bright
      Loop
    Death:
      BAL7 CDE 6 Bright
      Stop
  }
}
```

BFG9000 plasma ball
Actor type: Explosive
Game: Doom
DoomEd Number: None
Class Name: BFGBall
Spawn ID: 128
Identifier: T_BFGSHOT
Classes: BFGBall

Projectile that is launched from Doom and Doom 2's BFG9000 weapon. With 100 damage and damaging tracers, this is easily the most powerful projectile in Doom.

The BFG tracers will create green splashes on the monsters, making it easy to see which targets were actually hit. The weapon also defines a damage type, which is "BFGSplash", which is defined in the aforementioned splashes.

DECORATE definition

```
ACTOR BFGBall
{
    Radius 13
    Height 8
    Speed 25
    Damage 100
    Projectile
    +RANDOMIZE
    RenderStyle Add
    Alpha 0.75
    DeathSound "weapons/bfgx"
    Obituary "$OB_MPBFG_BOOM"
    States
    {
        Spawn:
            BFS1 AB 4 Bright
            Loop
        Death:
            BFE1 AB 8 Bright
            BFE1 C 8 Bright A_BFGSpray
            BFE1 DEF 8 Bright
            Stop
    }
}
```

BFG9000 tracer explosion
Actor type: Explosive
Game: Doom
DoomEd Number: None
Class Name: BFGExtra
Classes: BFGExtra

The flash displayed when an actor falls into one of the tracers of a BFG Ball.

DECORATE definition

```
ACTOR BFGExtra
{
    +NOBLOCKMAP
    +NOGRAVITY
    RenderStyle Add
    Alpha 0.75
    DamageType "BFGSplash"
    States
    {
        Spawn:
            BFE2 ABCD 8 Bright
            Stop
    }
}
```

Bullet puff
Actor type: Explosive
Game: Doom
DoomEd Number: None
Class Name: BulletPuff
Spawn ID: 131
Identifier:T_PUFF
Classes: BulletPuff

A puff actor spawned by weapon and monster hitscan attacks (i.e. bullets) in Doom. Note that you do not need to inherit from this class to create your own puffs; puffs are just regular actors and don't need to inherit from a specific class.

DECORATE definition

```
ACTOR BulletPuff
{
  +NOBLOCKMAP
  +NOGRAVITY
  +ALLOWPARTICLES
  +RANDOMIZE
  RenderStyle Translucent
  Alpha 0.5
  VSpeed 1
  Mass 5
  States
  {
    Spawn:
      PUFF A 4 Bright
      PUFF B 4
      // Intentional fall-through
    Melee:
      PUFF CD 4
      Stop
  }
}
```

Cacodemon projectile
Actor type: Explosive
Game: Doom
DoomEd Number: None
Class Name: CacodemonBall
Spawn ID: 126
Identifier:T_CACODEMONSHOT

Classes: CacodemonBall

Lightning ball fired from a Cacodemon's mouth.

DECORATE definition

ACTOR CacodemonBall

```
{  
  Radius 6  
  Height 8  
  Speed 10  
  FastSpeed 20  
  Damage 5  
  Projectile  
  +RANDOMIZE  
  +ZDOOMTRANS  
  RenderStyle Add  
  Alpha 1  
  SeeSound "caco/attack"  
  DeathSound "caco/shotx"  
  States  
  {  
    Spawn:  
      BAL2 AB 4 Bright  
      Loop  
    Death:  
      BAL2 CDE 6 Bright  
      Stop  
  }  
}
```

Imp fireball
Actor type: Explosive
Game: Doom
DoomEd Number: None
Class Name: DoomImpBall
Spawn ID: 10
Identifier: T_IMPFIREBALL
Classes: DoomImpBall

The fireball thrown by an Imp.

DECORATE definition

ACTOR DoomImpBall

```
{
  Radius 6
  Height 8
  Speed 10
  FastSpeed 20
  Damage 3
  Projectile
  +RANDOMIZE
  RenderStyle Add
  Alpha 1
  SeeSound "imp/attack"
  DeathSound "imp/shotx"
  States
  {
  Spawn:
    BAL1 AB 4 Bright
    Loop
  Death:
    BAL1 CDE 6 Bright
    Stop
  }
}
```

Mancubus flamethrower projectile
Actor type: Explosive
Game: Doom 2
DoomEd Number: None
Class Name: FatShot
Spawn ID: 153
Identifier: T_MANCUBUSSHOT
Classes: FatShot

A large fireball shot from a Mancubus.

DECORATE definition

ACTOR FatShot

```
{
  Radius 6
  Height 8
  Speed 20
  Damage 8
  Projectile
  +RANDOMIZE
  RenderStyle Add
  Alpha 1
  SeeSound "fatso/attack"
  DeathSound "fatso/shotx"
  States
  {
    Spawn:
      MANF AB 4 Bright
      Loop
    Death:
      MISL B 8 Bright
      MISL C 6 Bright
      MISL D 4 Bright
      Stop
  }
}
```


Plasma rifle bolt
Actor type: Explosive
Game: Doom
DoomEd Number: None
Class Name: PlasmaBall
Spawn ID: 51
Identifier: T_PLASMABOLT
Classes: PlasmaBall
 â†’PlasmaBall1
 â†’PlasmaBall2

A blue projectile fired from the player's Plasma Rifle.

DECORATE definition

```
ACTOR PlasmaBall
{
    Radius 13
    Height 8
    Speed 25
    Damage 5
    Projectile
    +RANDOMIZE
    +ZDOOMTRANS
    RenderStyle Add
    Alpha 0.75
    SeeSound "weapons/plasmaf"
    DeathSound "weapons/plasmax"
    Obituary "$OB_MPPLASMARIFLE" // "%o was melted by %k's plasma gun."
    States
    {
        Spawn:
            PLSS AB 6 Bright
            Loop
        Death:
            PLSE ABCDE 4 Bright
            Stop
    }
}
```

Revenant homing missile
Actor type: Explosive
Game: Doom 2
DoomEd Number: None
Class Name: RevenantTracer
Spawn ID: 53
Identifier:T_TRACER
Classes: RevenantTracer

The seeking missile fired by a Revenant. Whether it actually homes in on the Revenant's target or not is random (it's about 50% chance, but check A_Tracer for exact numbers), this behavior is caused by the way A_Tracer was written.

From their sprite names, it could be inferred they were originally planned to be the Mancubus' missiles.

DECORATE definition

```
ACTOR RevenantTracer
{
    Radius 11
    Height 8
    Speed 10
    Damage 10
    Projectile
    +SEEKERMISSILE
    +RANDOMIZE
    SeeSound "skeleton/attack"
    DeathSound "skeleton/tracex"
    RenderStyle Add
    States
    {
        Spawn:
            FATB AB 2 Bright A_Tracer
            Loop
        Death:
            FBXP A 8 Bright
            FBXP B 6 Bright
            FBXP C 4 Bright
            Stop
    }
}
```

Revenant homing missile trail
Actor type: Explosive
Game: Doom 2
DoomEd Number: None
Class Name: RevenantTracerSmoke
Classes: RevenantTracerSmoke

The trail of smoke left by a Revenant Tracer. This is almost the same as a Bullet Puff. Remember that the trail is only displayed when the Revenant's Seeker is currently targeting an actor.

DECORATE definition

ACTOR RevenantTracerSmoke

```
{
+NOBLOCKMAP
+NOGRAVITY
+NOTELEPORT
RenderStyle Translucent
Alpha 0.5
States
{
Spawn:
    PUFF ABABC 4
    Stop
}
}
```

Rocket
Actor type: Explosive
Game: Doom
DoomEd Number: None
Class Name: Rocket
Spawn ID: 127
Identifier: T_ROCKET
Classes: Rocket

The projectile fired from a player's rocket launcher or a cyberdemon. The second-most powerful projectile in Doom.

DECORATE definition

```
ACTOR Rocket
{
    Radius 11
    Height 8
    Speed 20
    Damage 20
    Projectile
    +RANDOMIZE
    +DEHEXPLOSION
    +ROCKETTRAIL
    SeeSound "weapons/rocklf"
    DeathSound "weapons/rocklx"
    Obituary "$OB_MPROCKET" // "%o rode %k's rocket."
    States
    {
        Spawn:
            MISL A 1 Bright
            Loop
        Death:
            MISL B 8 Bright A_Explode
            MISL C 6 Bright
            MISL D 4 Bright
            Stop
    }
}
```

Twitching hanged corpse
Actor type: Gore
Game: Doom
DoomEd Number: 49
Class Name: BloodyTwitch
Classes: BloodyTwitch
â€¢â†’NonsolidTwitch

A twitching half-naked corpse with its legs torn off.

DECORATE definition

ACTOR BloodyTwitch

```
{
  Radius 16
  Height 68
  +SOLID
  +NOGRAVITY
  +SPAWNCEILING
  States
  {
  Spawn:
    GOR1 A 10
    GOR1 B 15
    GOR1 C 8
    GOR1 B 6
    Loop
  }
}
```

Pool of blood with spine and brain

Actor type: Gore

Game: Doom 2

DoomEd Number: 81

Class Name: BrainStem

Spawn ID: 150

Identifier: T_BRAINS

Classes: BrainStem

Pool of blood, brain and spinal cord

DECORATE definition

ACTOR BrainStem

```
{
  Radius 20
  Height 4
  +NOBLOCKMAP
  +MOVEWITHSECTOR
  States
  {
    Spawn:
      BRS1 A -1
      Stop
  }
}
```

Pool of blood with torn guts
Actor type: Gore
Game: Doom 2
DoomEd Number: 79
Class Name: ColonGibs
Spawn ID: 147
Identifier: T_BLOODPOOL1
Classes: ColonGibs

Pool of blood and guts.

DECORATE definition

ACTOR ColonGibs

```
{  
  Radius 20  
  Height 4  
  +NOBLOCKMAP  
  +MOVEWITHSECTOR  
  States  
  {  
    Spawn:  
      POB1 A -1  
      Stop  
  }  
}
```

Skewered dead zombie
Actor type: Gore
Game: Doom
DoomEd Number: 25
Class Name: DeadStick
Classes: DeadStick

A dead zombie skewered on a stake.

DECORATE definition

```
ACTOR DeadStick
{
    Radius 16
    Height 64
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            POL1 A -1
            Stop
    }
}
```


Pool of crushed gibbs
Actor type: Gore
Game: Doom
DoomEd Number: 24
Class Name: Gibs
Spawn ID: 146
Identifier: T_BLOODPOOL
Classes: RealGibs†Gibs

Gibs that can be placed on a map.

DECORATE definition

```
ACTOR Gibs : RealGibs
{
    ClearFlags
}
```

Hanging corpse with guts, brain and one leg removed

Actor type: Gore

Game: Doom 2

DoomEd Number: 74

Class Name: HangBNoBrain

Classes: HangBNoBrain

DECORATE definition

ACTOR HangBNoBrain

```
{
  Radius 16
  Height 88
  +SOLID
  +NOGRAVITY
  +SPAWNCEILING
  States
  {
    Spawn:
      HDB2 A -1
      Stop
  }
}
```

Hanging corpse with guts removed
Actor type: Gore
Game: Doom 2
DoomEd Number: 73
Class Name: HangNoGuts
Classes: HangNoGuts

This actor needs a description.

ACTOR HangNoGuts

```
{  
  Radius 16  
  Height 88  
  +SOLID  
  +NOGRAVITY  
  +SPAWNCEILING  
  States  
  {  
    Spawn:  
      HDB1 A -1  
      Stop  
  }  
}
```

Hanging half-torso with head looking down

Actor type: Gore

Game: Doom 2

DoomEd Number: 75

Class Name: HangTLookingDown

Classes: HangTLookingDown

Hanging torso, looking down.

DECORATE definition

ACTOR HangTLookingDown

```
{  
  Radius 16  
  Height 64  
  +SOLID  
  +NOGRAVITY  
  +SPAWNCEILING  
  States  
  {  
    Spawn:  
      HDB3 A -1  
      Stop  
  }  
}
```

Hanging half-torso with head looking up

Actor type: Gore

Game: Doom 2

DoomEd Number: 77

Class Name: HangTLookingUp

Classes: HangTLookingUp

This actor needs a description.

DECORATE definition

ACTOR HangTLookingUp

```
{
  Radius 16
  Height 64
  +SOLID
  +NOGRAVITY
  +SPAWNCEILING
  States
  {
  Spawn:
    HDB5 A -1
    Stop
  }
}
```

Hanging half-torso without brain
Actor type: Gore
Game: Doom 2
DoomEd Number: 78
Class Name: HangTNoBrain
Classes: HangTNoBrain

This actor needs a description.

DECORATE definition

ACTOR HangTNoBrain

```
{  
  Radius 16  
  Height 64  
  +SOLID  
  +NOGRAVITY  
  +SPAWNCEILING  
  States  
  {  
    Spawn:  
      HDB6 A -1  
      Stop  
  }  
}
```

Hanging half-torso with broken skull
Actor type: Gore
Game: Doom 2
DoomEd Number: 76
Class Name: HangTSkull
Classes: HangTSkull

This actor needs a description.

DECORATE definition

ACTOR HangTSkull

```
{  
  Radius 16  
  Height 64  
  +SOLID  
  +NOGRAVITY  
  +SPAWNCEILING  
  States  
  {  
    Spawn:  
      HDB4 A -1  
      Stop  
  }  
}
```

Still-living skewered zombie
Actor type: Gore
Game: Doom
DoomEd Number: 26
Class Name: LiveStick
Classes: LiveStick

A still-twitching zombie skewered on a stake.

DECORATE definition

```
ACTOR LiveStick
{
    Radius 16
    Height 64
    ProjectilePassHeight -16
    +SOLID
    States
    {
    Spawn:
        POL6 A 6
        POL6 B 8
        Loop
    }
}
```


Hanging corpse
Actor type: Gore
Game: Doom
DoomEd Number: 50
Class Name: Meat2
Classes: Meat2
â€¢â†’NonsolidMeat2

This actor needs a description.

DECORATE definition

ACTOR Meat2
{
 Radius 16
 Height 84
 +SOLID
 +NOGRAVITY
 +SPAWNCEILING
 States
 {
 Spawn:
 GOR2 A -1
 Stop
 }
}

One-legged hanging corpse
Actor type: Gore
Game: Doom
DoomEd Number: 51
Class Name: Meat3
Classes: Meat3
â€“,â†’NonsolidMeat3

This actor needs a description.

```
ACTOR Meat3
{
    Radius 16
    Height 84
    +SOLID
    +NOGRAVITY
    +SPAWNCEILING
    States
    {
        Spawn:
            GOR3 A -1
            Stop
    }
}
```

Hanging legs and waist
Actor type: Gore
Game: Doom
DoomEd Number: 52
Class Name: Meat4
Classes: Meat4
â€â†’NonsolidMeat4

This actor needs a description.

```
ACTOR Meat4
{
  Radius 16
  Height 68
  +SOLID
  +NOGRAVITY
  +SPAWNCEILING
  States
  {
    Spawn:
      GOR4 A -1
      Stop
  }
}
```

Hanging leg
Actor type: Gore
Game: Doom
DoomEd Number: 53
Class Name: Meat5
Classes: Meat5
â€â†’NonsolidMeat5

This actor needs a description.

DECORATE definition

ACTOR Meat5
{
 Radius 16
 Height 52
 +SOLID
 +NOGRAVITY
 +SPAWNCEILING
 States
 {
 Spawn:
 GOR5 A -1
 Stop
 }
}

Hanging corpse
Actor type: Gore
Game: Doom
DoomEd Number: 59
Class Name: NonsolidMeat2
Classes: Meat2†NonsolidMeat2

This actor needs a description.

ACTOR NonsolidMeat2 : Meat2
{
-SOLID
Radius 20
}

One-legged hanging corpse
Actor type: Gore
Game: Doom
DoomEd Number: 61
Class Name: NonsolidMeat3
Classes: Meat3†NonsolidMeat3

This actor needs a description.

ACTOR NonsolidMeat3 : Meat3
{
-SOLID
Radius 20
}

Hanging legs and waist
Actor type: Gore
Game: Doom
DoomEd Number: 60
Class Name: NonsolidMeat4
Classes: Meat4†NonsolidMeat4

This actor needs a description.

DECORATE definition

ACTOR NonsolidMeat4 : Meat4
{
-SOLID
Radius 20
}

Hanging leg
Actor type: Gore
Game: Doom
DoomEd Number: 62
Class Name: NonsolidMeat5
Classes: Meat5†NonsolidMeat5

This actor needs a description.

DECORATE definition

```
ACTOR NonsolidMeat5 : Meat5
{
  -SOLID
  Radius 20
}
```


Twitching hanged corpse
Actor type: Gore
Game: Doom
DoomEd Number: 63
Class Name: NonsolidTwitch
Classes: BloodyTwitch†'NonsolidTwitch

This actor needs a description.

DECORATE definition

ACTOR NonsolidTwitch : BloodyTwitch
{
-SOLID
Radius 20
}

Small pool of blood
Actor type: Gore
Game: Doom 2
DoomEd Number: 80
Class Name: SmallBloodPool
Spawn ID: 148
Identifier: T_BLOODPOOL2
Classes: SmallBloodPool

A small pool of blood.

DECORATE definition

```
ACTOR SmallBloodPool
{
    Radius 20
    Height 1
    +NOBLOCKMAP
    +MOVEWITHSECTOR
    States
    {
        Spawn:
            POB2 A -1
            Stop
    }
}
```

Big twisted tree
Actor type: Vegetation
Game: Doom
DoomEd Number: 54
Class Name: BigTree
Classes: BigTree

Large crooked tree.

DECORATE definition

```
ACTOR BigTree
{
    Radius 32
    Height 108
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            TRE2 A -1
            Stop
    }
}
```

Blue torch
Actor type: Light source
Game: Doom
DoomEd Number: 44
Class Name: BlueTorch
Classes: BlueTorch

This is a tall standing torch which is burning with a blue flame.

DECORATE definition

```
ACTOR BlueTorch
{
    Radius 16
    Height 68
    ProjectilePassHeight -16
    +SOLID
    States
    {
    Spawn:
        TBLU ABCD 4 Bright
        Loop
    }
}
```

Candelabra
Actor type: Light source
Game: Doom
DoomEd Number: 35
Class Name: Candelabra
Classes: Candelabra

Candelabra.

DECORATE definition

```
ACTOR Candelabra
{
    Radius 16
    Height 60
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            CBRA A -1 Bright
            Stop
    }
}
```

Candlestick
Actor type: Light source
Game: Doom
DoomEd Number: 34
Class Name:Candlestick
Classes: Candlestick

A simple black candle.

DECORATE definition

```
ACTOR Candlestick
{
    Radius 20
    Height 14
    ProjectilePassHeight -16
    States
    {
        Spawn:
            CAND A -1 Bright
            Stop
    }
}
```

Evil eye
Actor type: Light source
Game: Doom
DoomEd Number: 41
Class Name: EvilEye
Classes: EvilEye

A floating, symbol-framed demonic eye with a candle at the ground.

DECORATE definition

```
ACTOR EvilEye
{
    Radius 16
    Height 54
    ProjectilePassHeight -16
    +SOLID
    States
    {
    Spawn:
        CEYE ABCB 6 Bright
        Loop
    }
}
```

Skulls on a floating lava rock

Actor type: Gore

Game: Doom

DoomEd Number: 42

Class Name: FloatingSkull

Classes: FloatingSkull

A sort of hovering cluster of stubby stalactites with a pair of skulls atop it, all bathed in a yellow-red light, meant to be placed above lava.

DECORATE definition

ACTOR FloatingSkull

```
{
  Radius 16
  Height 26
  ProjectilePassHeight -16
  +SOLID
  States
  {
  Spawn:
    FSKU ABC 6 Bright
    Loop
  }
}
```


Green torch
Actor type: Light source
Game: Doom
DoomEd Number: 45
Class Name GreenTorch
Classes: GreenTorch

This is a tall standing torch which is burning with a green flame.

DECORATE definition

```
ACTOR GreenTorch
{
    Radius 16
    Height 68
    ProjectilePassHeight -16
    +SOLID
    States
    {
    Spawn:
        TGRN ABCD 4 Bright
        Loop
    }
}
```

Skulls with candles
Actor type: Light source
Game: Doom
DoomEd Number: 29
Class Name: HeadCandles
Classes: HeadCandles

3 skulls on short sticks shoved to an organized pile of more skulls. There's a candle on every base skull.

DECORATE definition

```
ACTOR HeadCandles
{
    Radius 16
    Height 42
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            POL3 AB 6 Bright
            Loop
    }
}
```

Head on a stick
Actor type: Gore
Game: Doom
DoomEd Number: 27
Class Name: HeadOnAStick
Classes: HeadOnAStick

A wooden pole with a single human head staked at the top.

DECORATE definition

```
ACTOR HeadOnAStick
{
    Radius 16
    Height 56
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            POL4 A -1
            Stop
    }
}
```

Skewered heads on a stick
Actor type: Gore
Game: Doom
DoomEd Number: 28
Class Name: HeadsOnAStick
Classes: HeadsOnAStick

A wooden pole with several human heads impaled upon it.

DECORATE definition

```
ACTOR HeadsOnAStick
{
    Radius 16
    Height 64
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            POL2 A -1
            Stop
    }
}
```

Green column with still-beating heart

Actor type: Decoration

Game: Doom

DoomEd Number: 36

Class Name: HeartColumn

Classes: HeartColumn

A green marble column decoration with a bas-relief skull pattern and a beating heart sitting upon it.

DECORATE definition

ACTOR HeartColumn

```
{
  Radius 16
  Height 40
  ProjectilePassHeight -16
  +SOLID
  States
  {
  Spawn:
    COL5 AB 14
    Loop
  }
}
```

Red torch
Actor type: Light source
Game: Doom
DoomEd Number: 46
Class Name: RedTorch
Classes: RedTorch

A torch with a red flame.

DECORATE definition

```
ACTOR RedTorch
{
    Radius 16
    Height 68
    ProjectilePassHeight -16
    +SOLID
    States
    {
    Spawn:
        TRED ABCD 4 Bright
        Loop
    }
}
```

Short blue torch
Actor type: Light source
Game: Doom
DoomEd Number: 55
Class Name: ShortBlueTorch
Classes: ShortBlueTorch

This is a short standing torch which is burning with a blue flame.

DECORATE definition

ACTOR ShortBlueTorch

```
{  
  Radius 16  
  Height 37  
  ProjectilePassHeight -16  
  +SOLID  
  States  
  {  
    Spawn:  
      SMBT ABCD 4 Bright  
      Loop  
  }  
}
```

Short green column
Actor type: Decoration
Game: Doom
DoomEd Number: 31
Class Name: ShortGreenColumn
Classes: ShortGreenColumn

A green marble column decoration with a bas-relief skull pattern.

DECORATE definition

ACTOR ShortGreenColumn

```
{  
  Radius 16  
  Height 40  
  ProjectilePassHeight -16  
  +SOLID  
  States  
  {  
    Spawn:  
      COL2 A -1  
      Stop  
  }  
}
```


Short green torch
Actor type: Light source
Game: Doom
DoomEd Number: 56
Class Name: ShortGreenTorch
Classes: ShortGreenTorch

This is a short standing torch which is burning with a green flame.

DECORATE definition

ACTOR ShortGreenTorch

```
{  
  Radius 16  
  Height 37  
  ProjectilePassHeight -16  
  +SOLID  
  States  
  {  
    Spawn:  
      SMGT ABCD 4 Bright  
      Loop  
  }  
}
```

Short red column
Actor type: Decoration
Game: Doom
DoomEd Number: 33
Class Name: ShortRedColumn
Classes: ShortRedColumn

A red marble column decoration with a bas-relief skull pattern.

DECORATE definition

ACTOR ShortRedColumn

```
{  
  Radius 16  
  Height 40  
  ProjectilePassHeight -16  
  +SOLID  
  States  
  {  
    Spawn:  
      COL4 A -1  
      Stop  
  }  
}
```

```
ACTOR ShortRedTorch
{
    Radius 16
    Height 37
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            SMRT ABCD 4 Bright
            Loop
    }
}
```

Red column with skull
Actor type: Decoration
Game: Doom
DoomEd Number: 37
Class Name: SkullColumn
Classes: SkullColumn

A red marble column with a humanoid skull sitting upon it.

DECORATE definition

```
ACTOR SkullColumn
{
    Radius 16
    Height 40
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            COL6 A -1
            Stop
    }
}
```

Stalagmite
Actor type: Decoration
Game: Doom
DoomEd Number: 47
Class Name: Stalagtite
Classes: Stalagtite

A short, spiky brown stalagmite that looks vaguely like a hellish vegetation growth. Do not pay attention to the class name which is neither stalactite (ceiling) nor stalagmite (ground).

Note that there is a stalagmite sprite (SMT2A) that looks more like a real stalagmite, but which wasn't used by a normal actor. See Stalagmite.

DECORATE definition

```
ACTOR Stalagtite
{
    Radius 16
    Height 40
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            SMT A -1
            Stop
    }
}
```

Tall green column
Actor type: Decoration
Game: Doom
DoomEd Number: 30
Class Name: TallGreenColumn

Classes: TallGreenColumn

A tall green marble column decoration with a bas-relief skull pattern.

DECORATE definition

ACTOR TallGreenColumn

```
{  
  Radius 16  
  Height 52  
  ProjectilePassHeight -16  
  +SOLID  
  States  
  {  
    Spawn:  
      COL1 A -1  
      Stop  
  }  
}
```

Tall red column
Actor type: Decoration
Game: Doom
DoomEd Number: 32
Class Name:TallRedColumn

Classes: Actor → TallRedColumn

A tall red marble column decoration found in Doom and Doom 2.

DECORATE definition

ACTOR TallRedColumn

```
{
  Radius 16
  Height 52
  ProjectilePassHeight -16
  +SOLID
  States
  {
  Spawn:
    COL3 A -1
    Stop
  }
}
```

Torched tree
Actor type: Vegetation
Game: Doom
DoomEd Number: 43
Class Name: TorchTree
Classes: TorchTree

Short hollow half-burned tree.

DECORATE definition

```
ACTOR TorchTree
{
    Radius 16
    Height 56
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            TRE1 A -1
            Stop
    }
}
```


Blue keycard
Actor type: Key
Game: Doom
DoomEd Number: 5
Class Name: BlueCard
Spawn ID: 85
Identifier: T_BLUEKEYCARD
Classes: Inventoryâ†’Keyâ†’DoomKeyâ†’BlueCard

Blue keycard.

DECORATE definition

```
ACTOR BlueCard : DoomKey
{
    Inventory.PickupMessage "$GOTBLUECARD"
    Inventory.Icon "STKEYS0"
    States
    {
        Spawn:
            BKEY A 10
            BKEY B 10 Bright
            Loop
    }
}
```

Blue skull key
Actor type: Key
Game: Doom
DoomEd Number: 40
Class Name: BlueSkull
Spawn ID: 90
Identifier: T_BLUESKULLKEY
Classes: Inventoryâ†’Keyâ†’DoomKeyâ†’BlueSkull

Blue skull key.

DECORATE definition

```
ACTOR BlueSkull : DoomKey
{
    Inventory.PickupMessage "$GOTBLUESKUL"
    Inventory.Icon "STKEYS3"
    States
    {
        Spawn:
            BSKU A 10
            BSKU B 10 Bright
        Loop
    }
}
```

Red keycard
Actor type: Key
Game: Doom
DoomEd Number: 13
Class Name: RedCard
Spawn ID: 86
Identifier: T_REDKEYCARD
Classes: Inventoryâ†’Keyâ†’DoomKeyâ†’RedCard

A Red keycard that opens red doors in Doom.

DECORATE definition

```
ACTOR RedCard : DoomKey
{
    Inventory.PickupMessage "$GOTREDCARD"
    Inventory.Icon "STKEYS2"
    States
    {
        Spawn:
            RKEY A 10
            RKEY B 10 Bright
        Loop
    }
}
```

Red skull key
Actor type: Key
Game: Doom
DoomEd Number: 38
Class Name: RedSkull
Spawn ID: 89
Identifier: T_REDSKULLKEY
Classes: Inventoryâ†’Keyâ†’DoomKeyâ†’RedSkull

Red skull key.

DECORATE definition

```
ACTOR RedSkull : DoomKey
{
    Inventory.PickupMessage "$GOTREDSKUL"
    Inventory.Icon "STKEYS5"
    States
    {
        Spawn:
            RSKU A 10
            RSKU B 10 Bright
        Loop
    }
}
```

Yellow keycard
Actor type: Key
Game: Doom
DoomEd Number: 6
Class Name: YellowCard
Spawn ID: 87
Identifier: T_YELLOWKEYCARD
Classes: Inventory†Key†DoomKey†YellowCard

Yellow keycard.

DECORATE definition

```
ACTOR YellowCard : DoomKey
{
    Inventory.PickupMessage "$GOTYELWCARD"
    Inventory.Icon "STKEYS1"
    States
    {
        Spawn:
            YKEY A 10
            YKEY B 10 Bright
        Loop
    }
}
```

Yellow skull key
Actor type: Key
Game: Doom
DoomEd Number: 39
Class Name: YellowSkull
Spawn ID: 88
Identifier: T_YELLOWSKULLKEY
Classes: Inventoryâ†’Keyâ†’DoomKeyâ†’YellowSkull

Yellow skull key.

DECORATE definition

```
ACTOR YellowSkull : DoomKey
{
    Inventory.PickupMessage "$GOTYELWSKUL"
    Inventory.Icon "STKEYS4"
    States
    {
        Spawn:
            YSKU A 10
            YSKU B 10 Bright
        Loop
    }
}
```

Boss brain
Actor type: Monster
Game: Doom 2
DoomEd Number: 88
Class Name: BossBrain
Classes: BossBrain

This actor is used to represent the exposed brain behind the hole in the Icon of Sin's skull. As an easter-egg from id Software, its sprites represent the severed head of John Romero on a pike.

DECORATE definition

```
ACTOR BossBrain
{
    Health 250
    Mass 10000000
    PainChance 255
    +SOLID
    +SHOOTABLE
    +NOICEDDEATH
    +OLDRADIUSDMG
    PainSound "brain/pain"
    DeathSound "brain/death"
    States
    {
        BrainExplode:
            MISL BC 10 Bright
            MISL D 10 A_BrainExplode
            Stop
        Spawn:
            BBRN A -1
            Stop
        Pain:
            BBRN B 36 A_BrainPain
            Goto Spawn
        Death:
            BBRN A 100 A_BrainScream
            BBRN AA 10
            BBRN A -1 A_BrainDie
            Stop
    }
}
```

note: A_BrainScream and A_BrainExplode spawn a rocket and modify it so it results in a projectile like this:

```
ACTOR BossRocket : Rocket
{
    States
    {
        Spawn:
            MISL BC 10 Bright
            MISL D 10 Bright A_BrainExplode
            Stop
    }
}
```


Boss cube shooter
Actor type: Monster
Game: Doom 2
DoomEd Number: 89
Class Name: BossEye
Classes: BossEye

This is an invisible actor that periodically fires the SpawnShot actor, more commonly known as the "Boss Cube," as part of the boss on Doom 2's MAP30. Each cube will fly to a random location on the map (determined by a series of BossTarget actors placed in the map) and spawn an enemy at random.

DECORATE definition

BossEye

This is an invisible actor that periodically fires the SpawnShot actor, more commonly known as the "Boss Cube," as part of the boss on Doom 2's MAP30. Each cube will fly to a random location on the map (determined by a series of BossTarget actors placed in the map) and spawn an enemy at random.

DECORATE definition

ACTOR BossEye

```
{
    Height 32
    +NOBLOCKMAP
    +NOSECTOR
    States
    {
    Spawn:
        SSWV A 10 A_Look
        Loop
    See:
        SSWV A 181 A_BrainAwake
        SSWV A 150 A_BrainSpit // See SpawnShot
        Wait
    }
}
```

Customization

You can customize which actors are spawned by inheriting from the BossEye actor and providing a list of actors using the DropItem property. For example, this code will create a new shooter that will spawn one of five custom monsters:

```
actor CustomEye : BossEye
{
    DropItem "UberZombie"
    DropItem "NaziCommando"
    DropItem "Trite"
    DropItem "D3Imp"
    DropItem "Chicken"
}
```

Like for RandomSpawner, the second optional parameter of DropItem can be used to weigh the list, however the first optional parameter has no effect. The following example accurately replicates the odds of the normal BossEye:

```
actor ICantBelieveItsNotBossEye : BossEye
{
```

```
DropItem "DoomImp"      255 50
DropItem "Demon"        255 40
DropItem "Spectre"      255 30
DropItem "PainElemental" 255 10
DropItem "Cacodemon"    255 30
DropItem "Archvile"     255 2
DropItem "Revenant"     255 10
DropItem "Arachnotron"  255 20
DropItem "Fatso"        255 30
DropItem "HellKnight"   255 24
DropItem "BaronOfHell"  255 10
}
```

Boss cube spawning point

Actor type: Map spot

Game: Doom 2

DoomEd Number: 87

Class Name BossTarget

Classes: SpecialSpot†'BossTarget

When BossEyes are present on a map, they'll shoot their "boss cubes" at these special spots. If no BossTargets are present on the map, A_BrainSpit will fail and do nothing.

DECORATE definition

ACTOR BossTarget : SpecialSpot

```
{  
  Height 32  
  +NOBLOCKMAP  
  +NOSECTOR  
}
```

Commander Keen
Actor type: Monster
Game: Doom 2
DoomEd Number: 72
Class Name CommanderKeen
Classes: CommanderKeen

A character from the Commander Keen series, he just hangs from the ceiling and doesn't do anything. When all Keens present on a level are killed, any door tagged 666 is opened.

DECORATE definition

ACTOR CommanderKeen

```
{
  Health 100
  Radius 16
  Height 72
  Mass 10000000
  PainChance 256
  +SOLID
  +SPAWNCEILING
  +NOGRAVITY
  +SHOOTABLE
  +COUNTKILL
  +NOICEDEATH
  +ISMONSTER
  PainSound "keen/pain"
  DeathSound "keen/death"
  States
  {
    Spawn:
      KEEN A -1
      Loop
    Death:
      KEEN AB 6
      KEEN C 6 A_Scream
      KEEN DEFGH 6
      KEEN I 6
      KEEN J 6
      KEEN K 6 A_KeenDie
      KEEN L -1
      Stop
    Pain:
      KEEN M 4
      KEEN M 8 A_Pain
      Goto Spawn
  }
}
```

Space marine
Actor type: Player
Game: Doom
DoomEd Number: None
Class Name: DoomPlayer
Classes: PlayerPawn â†’ DoomPlayer
â†’ ChexPlayer

The DoomPlayer is who you play as in Doom, Ultimate Doom, Doom II, and Final Doom.

DECORATE Definition

ACTOR DoomPlayer : PlayerPawn

```
{
  Speed 1
  Health 100
  Radius 16
  Height 56
  Mass 100
  PainChance 255
  Player.DisplayName "Marine"
  Player.CrouchSprite "PLYC"
  Player.StartItem "Pistol"
  Player.StartItem "Fist"
  Player.StartItem "Clip", 50
  Player.WeaponSlot 1, Fist, Chainsaw
  Player.WeaponSlot 2, Pistol
  Player.WeaponSlot 3, Shotgun, SuperShotgun
  Player.WeaponSlot 4, Chaingun
  Player.WeaponSlot 5, RocketLauncher
  Player.WeaponSlot 6, PlasmaRifle
  Player.WeaponSlot 7, BFG9000
  Player.ColorRange 112, 127
  Player.ColorSet 0, "Green",    0x70, 0x7F, 0x72
  Player.ColorSet 1, "Gray",    0x60, 0x6F, 0x62 // Called "Indigo" originally so as to have a unique initial
  Player.ColorSet 2, "Brown",   0x40, 0x4F, 0x42
  Player.ColorSet 3, "Red",     0x20, 0x2F, 0x22
  // Doom Legacy additions
  Player.ColorSet 4, "Light Gray", 0x58, 0x67, 0x5A
  Player.ColorSet 5, "Light Brown", 0x38, 0x47, 0x3A
  Player.ColorSet 6, "Light Red",  0xB0, 0xBF, 0xB2
  Player.ColorSet 7, "Light Blue", 0xC0, 0xCF, 0xC2
}
```

States

```
{
  Spawn:
    PLAY A -1
    Loop
  See:
    PLAY ABCD 4
    Loop
  Missile:
    PLAY E 12
}
```

```
Goto Spawn
Melee:
    PLAY F 6 BRIGHT
    Goto Missile
Pain:
    PLAY G 4
    PLAY G 4 A_Pain
    Goto Spawn
Death:
    PLAY H 0 A_PlayerSkinCheck("AltSkinDeath")
Death1:
    PLAY H 10
    PLAY I 10 A_PlayerScream
    PLAY J 10 A_NoBlocking
    PLAY KLM 10
    PLAY N -1
    Stop
XDeath:
    PLAY O 0 A_PlayerSkinCheck("AltSkinXDeath")
XDeath1:
    PLAY O 5
    PLAY P 5 A_XScream
    PLAY Q 5 A_NoBlocking
    PLAY RSTUV 5
    PLAY W -1
    Stop
AltSkinDeath:
    PLAY H 6
    PLAY I 6 A_PlayerScream
    PLAY JK 6
    PLAY L 6 A_NoBlocking
    PLAY MNO 6
    PLAY P -1
    Stop
AltSkinXDeath:
    PLAY Q 5 A_PlayerScream
    PLAY R 0 A_NoBlocking
    PLAY R 5 A_SkullPop
    PLAY STUVWX 5
    PLAY Y -1
    Stop
}
}
```

DeHackEd placeholder
Actor type: Internal
Game: Doom
DoomEd Number: None
Class Name: DoomUnusedStates
Classes: DoomUnusedStates

Doom defined the states S_STALAG, S_DEADTORSO, and S_DEADBOTTOM but never actually used them. For compatibility with DeHackEd patches, they still need to be kept around. This actor serves that purpose.

DECORATE definition

ACTOR DoomUnusedStates

```
{
  States
  {
    Label1:
      SMT2 A -1
      Stop
    Label2:
      PLAY N -1
      Stop
      PLAY S -1
      Stop
    TNT: // MBF compatibility
      TNT1 A -1
      Loop
  }
}
```

Boss spawning fire
Actor type: SFX
Game: Doom 2
DoomEd Number: None
Class Name SpawnFire
Classes: SpawnFire

The brief effect accompanying the spawning of a monster by a boss cube from the boss eye. It reuses the sprites and the A_Fire action function from the Archvile's attack.

DECORATE definition

ACTOR SpawnFire

```
{
  Height 78
  +NOBLOCKMAP
  +NOGRAVITY
  RenderStyle Add
  States
  {
    Spawn:
      FIRE ABCDEFGH 4 Bright A_Fire
      Stop
  }
}
```


Boss cube

Actor type: Explosive

Game: Doom 2

DoomEd Number: None

Class Name: SpawnShot

Classes: Actor + SpawnShot

Projectile fired by a BossEye at a random BossTarget. Once the target is reached, a random monster is spawned. Flies through walls.

Monster list

A custom spawn shot can define its own list of monsters to spawn from its DropItem list. If it does so, it overrides its boss eye's monster list. See BossEye for more information on the syntax.

DECORATE definition

ACTOR SpawnShot

```
{
  Radius 6
  Height 32
  Speed 10
  Damage 3
  Projectile
  +NOCLIP
  -ACTIVATEPCROSS
  +RANDOMIZE
  SeeSound "brain/spit"
  DeathSound "brain/cubeboom"
  States
  {
  Spawn:
    BOSF A 3 Bright A_SpawnSound
    BOSF BCD 3 Bright A_SpawnFly // See SpawnFire
    Loop
  }
}
```

Arachnotron
Actor type: Monster
Game: Doom 2
DoomEd Number: 68
Class Name: Arachnotron
Spawn ID: 6
Identifier: T_ARACHNOTRON
Classes: Arachnotron
â€¢â†’StealthArachnotron

Arachnotrons are spider-like enemies that first appear in Doom 2. Their choice of weapon is plasma, which deals the same damage as the player's plasma rifle, although the colors are different and they do not fire as fast.

DECORATE definition

```
ACTOR Arachnotron
{
    Health 500
    Radius 64
    Height 64
    Mass 600
    Speed 12
    PainChance 128
    Monster
    +FLOORCLIP
    +BOSSDEATH
    SeeSound "baby/sight"
    PainSound "baby/pain"
    DeathSound "baby/death"
    ActiveSound "baby/active"
    Obituary "$OB_BABY"
    States
    {
    Spawn:
        BSPI AB 10 A_Look
        Loop
    See:
        BSPI A 20
        BSPI A 3 A_BabyMetal
        BSPI ABBCC 3 A_Chase
        BSPI D 3 A_BabyMetal
        BSPI DEEFF 3 A_Chase
        Goto See+1
    Missile:
        BSPI A 20 Bright A_FaceTarget
        BSPI G 4 Bright A_BspiAttack
        BSPI H 4 Bright
        BSPI H 1 Bright A_SpidRefire
        Goto Missile+1
    Pain:
        BSPI I 3
        BSPI I 3 A_Pain
        Goto See+1
```

Death:

BSPI J 20 A_Scream

BSPI K 7 A_NoBlocking

BSPI LMNO 7

BSPI P -1 A_BossDeath

Stop

Raise:

BSPI P 5

BSPI ONMLKJ 5

Goto See+1

}

}

Arch-Vile
Actor type: Monster
Game: Doom 2
DoomEd Number: 64
Class Name: Archvile
Spawn ID: 111
Identifier: T_VILE
Classes: Archvile
â€¢,â†’StealthArchvile

The Archvile's primary attack sends a huge flame into its target, which then punts it up into the air and knocking large amounts of health. The only way to dodge it is to get behind a wall fast, or hurt the Archvile before it can complete the attack. The Archvile can also resurrect it's hellish brethren. However, Archviles cannot resurrect the Cyberdemon, Spider Mastermind, Lost Soul, Pain Elemental, or fellow Archviles. The Archvile is lacking from the console versions of Doom.

The Archvile can only resurrect an actor if they have a Raise state in their actor definition. By default, the large enemies (listed above) do not have this state and so the Archvile won't attempt to resurrect them. If you want to create a new actor inherited from one with a Raise state, but don't want the new monster to be resurrectable, you can disable the Raise state using the following code in your actor:

Raise:
Stop

DECORATE definition

```
ACTOR Archvile
{
    Health 700
    Radius 20
    Height 56
    Mass 500
    Speed 15
    PainChance 10
    Monster
    MaxTargetRange 896
    +QUICKTORETALIATE
    +FLOORCLIP
    +NOTARGET
    SeeSound "vile/sight"
    PainSound "vile/pain"
    DeathSound "vile/death"
    ActiveSound "vile/active"
    MeleeSound "vile/stop"
    Obituary "$OB_VILE"
    States
    {
    Spawn:
        VILE AB 10 A_Look
        Loop
    See:
        VILE AABBCCDDEEFF 2 A_VileChase
```

```
    Loop
Missile:
    VILE G 0 Bright A_VileStart
    VILE G 10 Bright A_FaceTarget
    VILE H 8 Bright A_VileTarget
    VILE IJKLMN 8 Bright A_FaceTarget
    VILE O 8 Bright A_VileAttack
    VILE P 20 Bright
    Goto See
Heal:
    VILE "[\]" 10 Bright
    Goto See
Pain:
    VILE Q 5
    VILE Q 5 A_Pain
    Goto See
Death:
    VILE Q 7
    VILE R 7 A_Scream
    VILE S 7 A_NoBlocking
    VILE TUVWXY 7
    VILE Z -1
    Stop
}
}
```

Baron of Hell
Actor type: Monster
Game: Doom
DoomEd Number: 3003
Class Name: BaronOfHell
Spawn ID: 3
Identifier: T_BARON
Classes: BaronOfHell
→HellKnight
→StealthHellKnight
→StealthBaron

The Baron of Hell is akin to the Hell Knight, but has twice as much health. A pair of Barons serve as the endbosses for the first episode of Doom before becoming a far more common foe later on in the series.

DECORATE definition

```
ACTOR BaronOfHell
{
    Health 1000
    Radius 24
    Height 64
    Mass 1000
    Speed 8
    PainChance 50
    Monster
    +FLOORCLIP
    +BOSSDEATH
    SeeSound "baron/sight"
    PainSound "baron/pain"
    DeathSound "baron/death"
    ActiveSound "baron/active"
    Obituary "$OB_BARON"
    HitObituary "$OB_BARONHIT"
    States
    {
    Spawn:
        BOSS AB 10 A_Look
        Loop
    See:
        BOSS AABBCDD 3 A_Chase
        Loop
    Melee:
    Missile:
        BOSS EF 8 A_FaceTarget
        BOSS G 8 A_BruisAttack
        Goto See
    Pain:
        BOSS H 2
        BOSS H 2 A_Pain
        Goto See
    Death:
        BOSS I 8
        BOSS J 8 A_Scream
        BOSS K 8
```

BOSS L 8 A_NoBlocking
BOSS MN 8
BOSS O -1 A_BossDeath
Stop

Raise:

BOSS O 8
BOSS NMLKJI 8
Goto See

}
}

Cacodemon
Actor type: Monster
Game: Doom
DoomEd Number: 3005
Class Name: Cacodemon
Spawn ID: 19
Identifier: T_CACODEMON
Classes: Cacodemon
→DeadCacodemon
→StealthCacodemon

The Cacodemon has nothing better to do than to fly around and shoot purple balls at you through their mouth. They like biting up close so it's a good idea to keep your distance.

DECORATE definition

```
ACTOR Cacodemon
{
    Health 400
    Radius 31
    Height 56
    Mass 400
    Speed 8
    PainChance 128
    Monster
    +FLOAT
    +NOGRAVITY
    SeeSound "caco/sight"
    PainSound "caco/pain"
    DeathSound "caco/death"
    ActiveSound "caco/active"
    Obituary "$OB_CACO"
    HitObituary "$OB_CACOHIT"
    States
    {
        Spawn:
            HEAD A 10 A_Look
            Loop
        See:
            HEAD A 3 A_Chase
            Loop
        Missile:
            HEAD BC 5 A_FaceTarget
            HEAD D 5 Bright A_HeadAttack
            Goto See
        Pain:
            HEAD E 3
            HEAD E 3 A_Pain
            HEAD F 6
            Goto See
        Death:
            HEAD G 8
            HEAD H 8 A_Scream
            HEAD IJ 8
            HEAD K 8 A_NoBlocking
            HEAD L -1 A_SetFloorClip
```



```
    Stop  
  Raise:  
    HEAD L 8 A_UnSetFloorClip  
    HEAD KJIHG 8  
    Goto See  
  }  
}
```

Heavy Weapon Dude
Actor type: Monster
Game: Doom 2
DoomEd Number: 65
Class Name: ChaingunGuy
Spawn ID: 2
Identifier: T_CHAINGUY
Classes: ChaingunGuy
â€¢,â†’StealthChaingunGuy

The Chaingun Zombie, aka "Former Human Commando", totes a huge chaingun and can whittle a player's health down in seconds. Luckily, they can't take much damage and can be easily dispatched. They drop their chaingun when they die. They will even mow through their own ranks if you can get around them.

DECORATE definition

```
ACTOR ChaingunGuy
{
    Health 70
    Radius 20
    Height 56
    Mass 100
    Speed 8
    PainChance 170
    Monster
    +FLOORCLIP
    SeeSound "chainguy/sight"
    PainSound "chainguy/pain"
    DeathSound "chainguy/death"
    ActiveSound "chainguy/active"
    AttackSound "chainguy/attack"
    Obituary "$OB_CHAINGUY"
    Dropitem "Chaingun"
    States
    {
    Spawn:
        CPOS AB 10 A_Look
        Loop
    See:
        CPOS AABBCDD 3 A_Chase
        Loop
    Missile:
        CPOS E 10 A_FaceTarget
        CPOS FE 4 Bright A_CPosAttack
        CPOS F 1 A_CPosRefire
        Goto Missile+1
    Pain:
        CPOS G 3
        CPOS G 3 A_Pain
        Goto See
    Death:
        CPOS H 5
```

```
CPOS I 5 A_Scream
CPOS J 5 A_NoBlocking
CPOS KLM 5
CPOS N -1
Stop
XDeath:
CPOS O 5
CPOS P 5 A_XScream
CPOS Q 5 A_NoBlocking
CPOS RS 5
CPOS T -1
Stop
Raise:
CPOS N 5
CPOS MLKJIH 5
Goto See
}
}
```

Cyberdemon
Actor type: Monster
Game: Doom
DoomEd Number: 16
Class Name: Cyberdemon
Spawn ID: 114
Identifier: T_CYBERDEMON
Classes: Cyberdemon

With 4000 health, the Cyberdemon is the strongest enemy in Doom. They can take as many as four BFG shots to kill; with lesser weapons, this can require a lot of strafing around to dodge their rockets. It is recommended also to keep away from walls in their presence, as their rockets often inflict a lot of splash damage. As cyberdemons do not have a Raise state, they cannot be resurrected by archviles.

DECORATE definition

```
ACTOR Cyberdemon
{
    Health 4000
    Radius 40
    Height 110
    Mass 1000
    Speed 16
    PainChance 20
    Monster
    MinMissileChance 160
    +BOSS
    +MISSILEMORE
    +FLOORCLIP
    +NORADIUSDMG
    +DONTMORPH
    +BOSSDEATH
    SeeSound "cyber/sight"
    PainSound "cyber/pain"
    DeathSound "cyber/death"
    ActiveSound "cyber/active"
    Obituary "$OB_CYBORG"
    States
    {
    Spawn:
        CYBR AB 10 A_Look
        Loop
    See:
        CYBR A 3 A_Hoof
        CYBR ABBCC 3 A_Chase
        CYBR D 3 A_Metal
        CYBR D 3 A_Chase
        Loop
    Missile:
        CYBR E 6 A_FaceTarget
        CYBR F 12 A_CyberAttack
        CYBR E 12 A_FaceTarget
        CYBR F 12 A_CyberAttack
```

CYBR E 12 A_FaceTarget
CYBR F 12 A_CyberAttack
Goto See

Pain:

CYBR G 10 A_Pain
Goto See

Death:

CYBR H 10
CYBR I 10 A_Scream
CYBR JKL 10
CYBR M 10 A_NoBlocking
CYBR NO 10
CYBR P 30
CYBR P -1 A_BossDeath
Stop

}

}

Demon
Actor type: Monster
Game: Doom
DoomEd Number: 3002
Class Name: Demon
Spawn ID: 8
Identifier: T_DEMON
Classes: Demon
→DeadDemon
→Spectre
→StealthDemon

The Demon also known as Pinky Demon, is the first and only monster which won't shoot or fire at you from a distance. Instead, it will hunt you down and get up close and personal to bite you. Can be a low to high threat, depending on the number of Demons.

DECORATE definition

```
ACTOR Demon
{
    Health 150
    PainChance 180
    Speed 10
    Radius 30
    Height 56
    Mass 400
    Monster
    +FLOORCLIP
    SeeSound "demon/sight"
    AttackSound "demon/melee"
    PainSound "demon/pain"
    DeathSound "demon/death"
    ActiveSound "demon/active"
    Obituary "$OB_DEMONHIT" // "%o was bit by a demon."
    States
    {
        Spawn:
            SARG AB 10 A_Look
            Loop
        See:
            SARG AABBCDD 2 Fast A_Chase
            Loop
        Melee:
            SARG EF 8 Fast A_FaceTarget
            SARG G 8 Fast A_SargAttack
            Goto See
        Pain:
            SARG H 2 Fast
            SARG H 2 Fast A_Pain
            Goto See
        Death:
            SARG I 8
            SARG J 8 A_Scream
            SARG K 4
            SARG L 4 A_NoBlocking
            SARG M 4
```

SARG N -1

Stop

Raise:

SARG N 5

SARG MLKJI 5

Goto See

}

}

Imp
Actor type: Monster
Game: Doom
DoomEd Number: 3001
Class Name: DoomImp
Spawn ID: 5
Identifier: T_IMP
Classes: DoomImp
→DeadDoomImp
→StealthDoomImp

Imps are common enemies throughout the game. They slash up close which is a pain because they always seem to get close to you when there's more than one. The fireballs they throw do average damage. Luckily they don't have that much health, and one shotgun blast can put them to sleep.

DECORATE definition

```
ACTOR DoomImp
{
    Health 60
    Radius 20
    Height 56
    Mass 100
    Speed 8
    PainChance 200
    Monster
    +FLOORCLIP
    SeeSound "imp/sight"
    PainSound "imp/pain"
    DeathSound "imp/death"
    ActiveSound "imp/active"
    HitObituary "$OB_IMPHIT"
    Obituary "$OB_IMP"
    States
    {
        Spawn:
            TROO AB 10 A_Look
            Loop
        See:
            TROO AABCCDD 3 A_Chase
            Loop
        Melee:
        Missile:
            TROO EF 8 A_FaceTarget
            TROO G 6 A_TroopAttack
            Goto See
        Pain:
            TROO H 2
            TROO H 2 A_Pain
            Goto See
        Death:
            TROO I 8
            TROO J 8 A_Scream
            TROO K 6
            TROO L 6 A_NoBlocking
            TROO M -1
```



```
    Stop
XDeath:
    TROO N 5
    TROO O 5 A_XScream
    TROO P 5
    TROO Q 5 A_NoBlocking
    TROO RST 5
    TROO U -1
    Stop
Raise:
    TROO ML 8
    TROO KJI 6
    Goto See
}
}
```

Mancubus
Actor type: Monster
Game: Doom 2
DoomEd Number: 67
Class Name: Fatso
Spawn ID: 112
Identifier: T_MANCUBUS
Classes: Fatso
â€,â€StealthFatso

The mancubus has a lot of health so killing them won't be easy, especially when they can shoot 6 fireballs at a time every time they attack! The fireballs take lots of health off too, and can be hard to dodge because they are huge. Get lots of cover before challenging these huge things.

DECORATE definition

```
ACTOR Fatso
{
    Health 600
    Radius 48
    Height 64
    Mass 1000
    Speed 8
    PainChance 80
    Monster
    +FLOORCLIP
    +BOSSDEATH
    SeeSound "fatso/sight"
    PainSound "fatso/pain"
    DeathSound "fatso/death"
    ActiveSound "fatso/active"
    Obituary "$OB_FATSO"
    States
    {
    Spawn:
        FATT AB 15 A_Look
        Loop
    See:
        FATT AABBBCCDDEEFF 4 A_Chase
        Loop
    Missile:
        FATT G 20 A_FatRaise
        FATT H 10 Bright A_FatAttack1
        FATT IG 5 A_FaceTarget
        FATT H 10 Bright A_FatAttack2
        FATT IG 5 A_FaceTarget
        FATT H 10 Bright A_FatAttack3
        FATT IG 5 A_FaceTarget
        Goto See
    Pain:
        FATT J 3
        FATT J 3 A_Pain
        Goto See
```

Death:

FATT K 6

FATT L 6 A_Scream

FATT M 6 A_NoBlocking

FATT NOPQRS 6

FATT T -1 A_BossDeath

Stop

Raise:

FATT R 5

FATT QPONMLK 5

Goto See

}

}

Hell Knight
Actor type: Monster
Game: Doom 2
DoomEd Number: 69
Class Name: HellKnight
Spawn ID: 113
Identifier: T_HELLKNIGHT
Classes: BaronOfHell'HellKnight
â€,â'StealthHellKnight

Hell Knights are a satyr-like monster similar to the Baron of Hell. Like the Baron, their primary attack is a green fireball. However, they only have half as much health as their big brothers.

DECORATE definition

```
ACTOR HellKnight : BaronOfHell
{
    Health 500
    -BOSSDEATH
    SeeSound "knight/sight"
    ActiveSound "knight/active"
    PainSound "knight/pain"
    DeathSound "knight/death"
    HitObituary "$OB_KNIGHTHIT"
    Obituary "$OB_KNIGHT"
    States
    {
    Spawn:
        BOS2 AB 10 A_Look
        Loop
    See:
        BOS2 AABBCDD 3 A_Chase
        Loop
    Melee:
    Missile:
        BOS2 EF 8 A_FaceTarget
        BOS2 G 8 A_BruisAttack
        Goto See
    Pain:
        BOS2 H 2
        BOS2 H 2 A_Pain
        Goto See
    Death:
        BOS2 I 8
        BOS2 J 8 A_Scream
        BOS2 K 8
        BOS2 L 8 A_NoBlocking
        BOS2 MN 8
        BOS2 O -1
        Stop
    Raise:
        BOS2 O 8
        BOS2 NMLKJI 8
```

Goto See

}
}
}

Lost Soul
Actor type: Monster
Game: Doom
DoomEd Number: 3006
Class Name: LostSoul
Spawn ID: 110
Identifier: T_LOSTSOUL
Classes: Actor → LostSoul
→BetaSkull
→DeadLostSoul

Lost souls have nearly twice as much health as an imp, but they don't shoot fireballs. They fly around and then charge at you for a big bite. These can really be a pain if there are lots of them flying around. In Doom 2, they are spat out by pain elementals as a form of offense.

DECORATE definition

```
ACTOR LostSoul
{
    Health 100
    Radius 16
    Height 56
    Mass 50
    Speed 8
    Damage 3
    PainChance 256
    Monster
    +FLOAT
    +NOGRAVITY
    +MISSILEMORE
    +DONTFALL
    +NOICEDATH
    AttackSound "skull/melee"
    PainSound "skull/pain"
    DeathSound "skull/death"
    ActiveSound "skull/active"
    RenderStyle SoulTrans
    Obituary "$OB_SKULL" // "%o was spooked by a lost soul."
    States
    {
        Spawn:
            SKUL AB 10 Bright A_Look
            Loop
        See:
            SKUL AB 6 Bright A_Chase
            Loop
        Missile:
            SKUL C 10 Bright A_FaceTarget
            SKUL D 4 Bright A_SkullAttack
            SKUL CD 4 Bright
            Goto Missile+2
        Pain:
            SKUL E 3 Bright
            SKUL E 3 Bright A_Pain
            Goto See
        Death:
```

SKUL F 6 Bright
SKUL G 6 Bright A_Scream
SKUL H 6 Bright
SKUL I 6 Bright A_NoBlocking
SKUL J 6
SKUL K 6
Stop

}

}

Pain Elemental
Actor type: Monster
Game: Doom 2
DoomEd Number: 71
Class Name: PainElemental
Spawn ID: 115
Identifier: T_PAINELEMENTAL
Classes: PainElemental

Pain elementals are similar in appearance to cacodemons, but are brown and have stubby arms. The pain elemental is unique in that it does not have an attack of its own, instead it spits lost souls to do its bidding for them. Note that when a pain elemental dies, three lost souls spawn burst out of it.

In vanilla Doom, if the total number of lost souls in the level was greater than 20, the pain elemental's attack would fail, putting a limit at 21 lost souls. This did not differentiate between lost souls spawned by pain elementals and "independent" lost souls placed directly in the map. In ZDoom, this is subjected to a compatibility option, accessible through the `compat_limitpain` console variable or the menu system.

DECORATE definition

```
ACTOR PainElemental
{
    Health 400
    Radius 31
    Height 56
    Mass 400
    Speed 8
    PainChance 128
    Monster
    +FLOAT
    +NOGRAVITY
    SeeSound "pain/sight"
    PainSound "pain/pain"
    DeathSound "pain/death"
    ActiveSound "pain/active"
    States
    {
    Spawn:
        PAIN A 10 A_Look
        Loop
    See:
        PAIN AABBC 3 A_Chase
        Loop
    Missile:
        PAIN D 5 A_FaceTarget
        PAIN E 5 A_FaceTarget
        PAIN F 5 Bright A_FaceTarget
        PAIN F 0 Bright A_PainAttack
        Goto See
    Pain:
        PAIN G 6
        PAIN G 6 A_Pain
        Goto See
    }
```



```
Death:
  PAIN H 8 Bright
  PAIN I 8 Bright A_Scream
  PAIN JK 8 Bright
  PAIN L 8 Bright A_PainDie
  PAIN M 8 Bright
  Stop
Raise:
  PAIN MLKJIH 8
  Goto See
}
```

Revenant
Actor type: Monster
Game: Doom 2
DoomEd Number: 66
Class Name: Revenant
Spawn ID: 20
Identifier: T_REVENANT
Classes: Revenant
â€¢,â†’StealthRevenant

The Revenant, a bag of bones with armour, will send heatseeking missiles into its targets from a distance and punch them when up close.

DECORATE definition

ACTOR Revenant

```
{
  Health 300
  Radius 20
  Height 56
  Mass 500
  Speed 10
  PainChance 100
  Monster
  MeleeThreshold 196
  +MISSILEMORE
  +FLOORCLIP
  SeeSound "skeleton/sight"
  PainSound "skeleton/pain"
  DeathSound "skeleton/death"
  ActiveSound "skeleton/active"
  MeleeSound "skeleton/melee"
  HitObituary "$OB_UNDEADHIT" // "%o was punched by a revenant."
  Obituary "$OB_UNDEAD" // "%o couldn't evade a revenant's fireball."
  States
  {
  Spawn:
    SKEL AB 10 A_Look
    Loop
  See:
    SKEL AABBCCDDEEFF 2 A_Chase
    Loop
  Melee:
    SKEL G 0 A_FaceTarget
    SKEL G 6 A_SkelWhoosh
    SKEL H 6 A_FaceTarget
    SKEL I 6 A_SkelFist
    Goto See
  Missile:
    SKEL J 0 Bright A_FaceTarget
    SKEL J 10 Bright A_FaceTarget
    SKEL K 10 A_SkelMissile
    SKEL K 10 A_FaceTarget
```

```
Goto See
Pain:
  SKEL L 5
  SKEL L 5 A_Pain
  Goto See
Death:
  SKEL LM 7
  SKEL N 7 A_Scream
  SKEL O 7 A_NoBlocking
  SKEL P 7
  SKEL Q -1
  Stop
Raise:
  SKEL Q 5
  SKEL PONML 5
  Goto See
}
}
```

Sergeant
Actor type: Monster
Game: Doom
DoomEd Number: 9
Class Name: ShotgunGuy
Spawn ID: 1
Identifier: T_SHOTGUY
Classes: ShotgunGuy
→StealthShotgunGuy
→DeadShotgunGuy

The Shotgun Zombie, more commonly known as Sergeants or Shotgunners, are zombies with a shotgun. They can be particularly dangerous to the player at close range.

DECORATE definition

```
ACTOR ShotgunGuy
{
    Health 30
    Radius 20
    Height 56
    Mass 100
    Speed 8
    PainChance 170
    Monster
    +FLOORCLIP
    SeeSound "shotguy/sight"
    AttackSound "shotguy/attack"
    PainSound "shotguy/pain"
    DeathSound "shotguy/death"
    ActiveSound "shotguy/active"
    Obituary "$OB_SHOTGUY"
    DropItem "Shotgun"
    States
    {
        Spawn:
            SPOS AB 10 A_Look
            Loop
        See:
            SPOS AABCCDD 3 A_Chase
            Loop
        Missile:
            SPOS E 10 A_FaceTarget
            SPOS F 10 Bright A_SPosAttackUseAtkSound
            SPOS E 10
            Goto See
        Pain:
            SPOS G 3
            SPOS G 3 A_Pain
            Goto See
        Death:
            SPOS H 5
            SPOS I 5 A_Scream
            SPOS J 5 A_NoBlocking
            SPOS K 5
            SPOS L -1
    }
}
```

```
    Stop
XDeath:
    SPOS M 5
    SPOS N 5 A_XScream
    SPOS O 5 A_NoBlocking
    SPOS PQRST 5
    SPOS U -1
    Stop
Raise:
    SPOS L 5
    SPOS KJIH 5
    Goto See
}
}
```

Spectre
Actor type: Monster
Game: Doom
DoomEd Number: 58
Class Name: Spectre
Spawn ID: 9
Identifier: T_SPECTRE
Classes: Demon†Spectre

The spectre is effectively the same as the demon, with the difference that it is rendered with the "fuzzy" partial invisibility effect. This can optionally be changed with the `r_drawfuzz` and (in OpenGL renderers) `gl_fuzztype` console variables.

DECORATE definition

```
ACTOR Spectre : Demon
{
+SHADOW
RenderStyle OptFuzzy
Alpha 0.5
SeeSound "spectre/sight"
AttackSound "spectre/melee"
PainSound "spectre/pain"
DeathSound "spectre/death"
ActiveSound "spectre/active"
HitObituary "$OB_SPECTREHIT" // "%o was eaten by a spectre."
}
```

Spider Mastermind
Actor type: Monster
Game: Doom
DoomEd Number: 7
Class Name: SpiderMastermind
Spawn ID: 7
Identifier: T_SPIDERMASTERMIND
Classes: SpiderMastermind

The Spider Mastermind (a.k.a. the Spiderdemon) is the endboss of Doom. Although with 3000 HPs, it doesn't have as much health as the Cyberdemon, but its chaingun is essentially a sped-up version of Sergeant's shotgun (firing 3 bullets per shot), making this enemy especially deadly.

Doom 2 manual jokingly describes Spider Mastermind as Arachnotron's "mom" and implies this demon is actually female.

DECORATE definition

ACTOR SpiderMastermind

```
{
  Health 3000
  Radius 128
  Height 100
  Mass 1000
  Speed 12
  PainChance 40
  Monster
  MinMissileChance 160
  +BOSS
  +MISSILEMORE
  +FLOORCLIP
  +NORADIUSDMG
  +DONTMORPH
  +BOSSDEATH
  SeeSound "spider/sight"
  AttackSound "spider/attack"
  PainSound "spider/pain"
  DeathSound "spider/death"
  ActiveSound "spider/active"
  Obituary "$OB_SPIDER"
  States
  {
  Spawn:
    SPID AB 10 A_Look
    Loop
  See:
    SPID A 3 A_Metal
    SPID ABB 3 A_Chase
    SPID C 3 A_Metal
    SPID CDD 3 A_Chase
    SPID E 3 A_Metal
    SPID EFF 3 A_Chase
    Loop
}
```

Missile:

SPID A 20 Bright A_FaceTarget

SPID G 4 Bright A_SPosAttackUseAtkSound

SPID H 4 Bright A_SPosAttackUseAtkSound

SPID H 1 Bright A_SpidRefire

Goto Missile+1

Pain:

SPID I 3

SPID I 3 A_Pain

Goto See

Death:

SPID J 20 A_Scream

SPID K 10 A_NoBlocking

SPID LMNOPQR 10

SPID S 30

SPID S -1 A_BossDeath

Stop

}

}

Wolfenstein SS
Actor type: Monster
Game: Doom 2
DoomEd Number: 84
Class Name: WolfensteinSS
Spawn ID: 116
Identifier: T_WOLFSS
Classes: WolfensteinSS

The Wolfenstein SS guard ("WolfSS") is an enemy from Wolfenstein 3D, which appears in the secret levels of Doom 2. He is armed with a machine gun and leaves a clip after death.

Difficulty levels in Wolfenstein 3D were implemented by controlling the speed at which the enemies react and fire. For Doom 2, id simply placed 2-4 of these enemies everywhere that one guard appeared in Wolfenstein. Unfortunately, because enemies in Doom can also hurt each other, this just made them tend to infight with each other more.

The attack frames for this enemy are single-rotation only (as in Wolfenstein 3D), so that he always appears to be firing directly at the player viewing him, no matter what direction he's actually shooting in.

DECORATE definition

```
ACTOR WolfensteinSS
{
    Health 50
    Radius 20
    Height 56
    Speed 8
    PainChance 170
    Monster
    +FLOORCLIP
    SeeSound "wolfss/sight"
    PainSound "wolfss/pain"
    DeathSound "wolfss/death"
    ActiveSound "wolfss/active"
    AttackSound "wolfss/attack"
    Obituary "$OB_WOLFSS"
    Dropitem "Clip"
    States
    {
        Spawn:
            SSWV AB 10 A_Look
            Loop
        See:
            SSWV AABBCDD 3 A_Chase
            Loop
        Missile:
            SSWV E 10 A_FaceTarget
            SSWV F 10 A_FaceTarget
            SSWV G 4 Bright A_CPosAttack
            SSWV F 6 A_FaceTarget
            SSWV G 4 Bright A_CPosAttack
            SSWV F 1 A_CPosRefire
            Goto Missile+1
    }
```

Pain:

SSWV H 3

SSWV H 3 A_Pain

Goto See

Death:

SSWV I 5

SSWV J 5 A_Scream

SSWV K 5 A_NoBlocking

SSWV L 5

SSWV M -1

Stop

XDeath:

SSWV N 5

SSWV O 5 A_XScream

SSWV P 5 A_NoBlocking

SSWV QRSTU 5

SSWV V -1

Stop

Raise:

SSWV M 5

SSWV LKJI 5

Goto See

}

}

Former Human
Actor type: Monster
Game: Doom
DoomEd Number: 3004
Class Name: ZombieMan
Spawn ID: 4
Identifier: T_ZOMBIE
Classes: Actor → ZombieMan
→StealthZombieMan
→DeadZombieMan

The Pistol Zombie, also known as Troopers or Former Humans, are the least threatening monster in Doom. They are very weak, terrible shots, and their weapons are not particularly damaging even in instances when they do hit.

DECORATE definition

```
ACTOR ZombieMan
{
    Health 20
    Radius 20
    Height 56
    Speed 8
    PainChance 200
    Monster
    +FLOORCLIP
    SeeSound "grunt/sight"
    AttackSound "grunt/attack"
    PainSound "grunt/pain"
    DeathSound "grunt/death"
    ActiveSound "grunt/active"
    Obituary "$OB_ZOMBIE" // "%o was killed by a zombieman."
    DropItem "Clip"
    States
    {
        Spawn:
            POSS AB 10 A_Look
            Loop
        See:
            POSS AABCCDD 4 A_Chase
            Loop
        Missile:
            POSS E 10 A_FaceTarget
            POSS F 8 A_PosAttack
            POSS E 8
            Goto See
        Pain:
            POSS G 3
            POSS G 3 A_Pain
            Goto See
        Death:
            POSS H 5
            POSS I 5 A_Scream
            POSS J 5 A_NoBlocking
            POSS K 5
            POSS L -1
    }
```

```
    Stop
XDeath:
    POSS M 5
    POSS N 5 A_XScream
    POSS O 5 A_NoBlocking
    POSS PQRST 5
    POSS U -1
    Stop
Raise:
    POSS K 5
    POSS JIH 5
    Goto See
}
}
```

Computer area map
Actor type: Powerup
Game: Doom
DoomEd Number: 2026
Class Name: Allmap
Spawn ID: 137
Identifier: T_COMPUTERMAP
Classes: Inventoryâ†’MapRevealerâ†’Allmap

The computer area map reveals all the unexplored areas of a level with gray lines in the automap. Contrarily to the iddt cheat, the computer area map shows neither hidden linedefs nor actors.

```
ACTOR Allmap : MapRevealer
{
+COUNTITEM
+INVENTORY.FANCYPICKUPSOUND
+INVENTORY.ALWAYSPICKUP
Inventory.MaxAmount 0
Inventory.PickupSound "misc/p_pkup"
Inventory.PickupMessage "$GOTMAP" // "Computer Area Map"
States
{
Spawn:
    PMAP ABCDCB 6 Bright
    Loop
}
}
```

Armor bonus
Actor type: Armor
Game: Doom
DoomEd Number: 2015
Class Name: ArmorBonus
Spawn ID: 22
Identifier: T_ARMORBONUS
Classes: Inventory†Armor†BasicArmorBonus†ArmorBonus

Armor bonuses add 1% of additional armor to the player, up to 200%. If the player already had armor left, the protection percent is unchanged. Otherwise, the player is granted 33% protection (same as the Green Armor).

DECORATE definition

```
ACTOR ArmorBonus : BasicArmorBonus
{
    Radius 20
    Height 16
    Inventory.PickupMessage "$GOTARMBONUS" // "Picked up an armor bonus."
    Inventory.Icon "ARM1A0"
    Armor.SavePercent 33.335
    Armor.SaveAmount 1
    Armor.MaxSaveAmount 200
    +COUNTITEM
    +INVENTORY.ALWAYSPICKUP
    States
    {
        Spawn:
            BON2 ABCDCB 6
            Loop
    }
}
```

Berserk
Actor type: Powerup
Game: Doom
DoomEd Number: 2023
Class Name: Berserk
Spawn ID: 134
Identifier: T_BERSERK
Classes: Inventory†CustomInventory†Berserk

When a berserk pack (or inherited actor) is picked up, it heals 100 health and switches the player's weapon to the fist (if available). The fist attack does 10 times as much damage as it would otherwise for the remainder of the level. However this doesn't apply when using a A_CustomPunch and chainsaw's A_Saw.

DECORATE definition

```
ACTOR Berserk : CustomInventory
{
+COUNTITEM
+INVENTORY.ALWAYSPICKUP
Inventory.PickupMessage "$GOTBERSERK" // "Berserk!"
Inventory.PickupSound "misc/p_pkup"
States
{
Spawn:
PSTR A -1
Stop
Pickup:
TNT1 A 0 A_GiveInventory("PowerStrength")
TNT1 A 0 HealThing(100, 0)
TNT1 A 0 A_SelectWeapon("Fist")
Stop
}
}
```

Combat armor
Actor type: Armor
Game: Doom
DoomEd Number: 2019
Class Name: BlueArmor
Spawn ID: 69
Identifier: T_BLUEARMOR
Classes: Inventory→Armor→BasicArmorPickup→BlueArmor
→BlueArmorForMegasphere

Doom's blue armor pickup grants the player 200% of armor that resists 50% of the damage taken. If the player is already at 200% armor, the item will not be picked up.

DECORATE definition

```
ACTOR BlueArmor : BasicArmorPickup
{
    Radius 20
    Height 16
    Inventory.PickupMessage "$GOTMEGA" // "Picked up the MegaArmor!"
    Inventory.Icon "ARM2A0"
    Armor.SavePercent 50
    Armor.SaveAmount 200
    States
    {
        Spawn:
            ARM2 A 6
            ARM2 B 6 Bright
        Loop
    }
}
```


Blur sphere
Actor type: Powerup
Game: Doom
DoomEd Number: 2024
Class Name: BlurSphere
Spawn ID: 135
Identifier: T_INVISIBILITY
Classes: Inventoryâ†’PowerupGiverâ†’BlurSphere

This is the invisibility sphere powerup from Doom and Doom 2. When picked up, it causes the player to appear translucent (or fuzzy, depending on the user's settings) to other players, making him difficult to see. Enemies which have not yet woken may fail to see a player with this powerup active, and enemies that are firing on the player will miss by a random amount. While this powerup is in effect, the player's weapons will appear translucent/fuzzy on the HUD.

DECORATE definition

```
ACTOR BlurSphere : PowerupGiver
{
+COUNTITEM
+VISIBILITYPULSE
+INVENTORY.AUTOACTIVATE
+INVENTORY.ALWAYSPICKUP
+INVENTORY.BIGPOWERUP
Inventory.MaxAmount 0
Powerup.Type "Invisibility"
RenderStyle Translucent
Inventory.PickupMessage "$GOTINVIS" // "Partial Invisibility"
States
{
Spawn:
    PINS ABCD 6 Bright
    Loop
}
}
```

Security armor
Actor type: Armor
Game:Doom
DoomEd Number: 2018
Class Name: GreenArmor
Spawn ID: 68
Identifier: T_GREENARMOR
Classes: Inventoryâ†’Armorâ†’BasicArmorPickupâ†’GreenArmor

Doom's green armor pickup grants the player 100% of armor that resists 33% of the damage taken. If the player is already at or above 100% armor, the item will not be picked up.

DECORATE definition

```
ACTOR GreenArmor : BasicArmorPickup
{
    Radius 20
    Height 16
    Inventory.PickupMessage "$GOTARMOR" // "Picked up the armor."
    Inventory.Icon "ARM1A0"
    Armor.SavePercent 33.335
    Armor.SaveAmount 100
    States
    {
        Spawn:
            ARM1 A 6
            ARM1 B 7 Bright
        Loop
    }
}
```

Health bonus
Actor type: Health
Game: Doom
DoomEd Number: 2014
Class Name: HealthBonus
Spawn ID: 152
Identifier: T_HEALTHBONUS
Classes: Inventory'Health'HealthBonus

A health bonus is an item in Doom which gives the player one health point. However, unlike medikits and stimpacks, the health bonus can go over the 100 HP limit, up to 200. A related item that has a larger effect is the soul sphere.

DECORATE definition

```
ACTOR HealthBonus : Health
{
+COUNTITEM
+INVENTORY.ALWAYSPICKUP
Inventory.Amount 1
Inventory.MaxAmount 200
Inventory.PickupMessage "$GOTHTHBONUS" // "Picked up a health bonus."
States
{
Spawn:
    BON1 ABCDCB 6
    Loop
}
}
```

Light-amplification visor
Actor type: Powerup
Game: Doom
DoomEd Number: 2045
Class Name: Infrared
Spawn ID: 138
Identifier: T_LIGHTAMP
Classes: Inventoryâ†’PowerupGiverâ†’Infrared

The light amplification goggles works like Heretic's torch by brightening all sector light to 255. In GZDoom and Skulltag, the OpenGL renderer's option of "enhanced light vision mode" turns some actors (monsters and pickups) an inverted white palette as in the effect of the invulnerability sphere, and gives a strong green hue to everything else. If that option is off, or when using the software renderer, the goggle effect is exactly the same as the torch effect.

DECORATE definition

```
ACTOR Infrared : PowerupGiver
{
+COUNTITEM
+INVENTORY.AUTOACTIVATE
+INVENTORY.ALWAYSPICKUP
Inventory.MaxAmount 0
Powerup.Type "LightAmp"
Inventory.PickupMessage "$GOTVISOR" // "Light Amplification Visor"
States
{
Spawn:
    PVIS A 6 Bright
    PVIS B 6
    Loop
}
}
```

Invulnerability sphere
Actor type: Powerup
Game: Doom
DoomEd Number: 2022
Class Name: InvulnerabilitySphere
Spawn ID: 133
Identifier: T_INVULNERABILITY
Classes: Inventoryâ†’PowerupGiverâ†’InvulnerabilitySphere

The invulnerability sphere is a green sphere with a grinning, red-eyed face appearing on it. When picked up, the player becomes invulnerable and sees everything through an inverted white map.

If the item is imported in Heretic or Hexen, its effect when picked up will vary because it depends on the game and on the player class's Player.InvulnerabilityMode feature rather than by the item itself. See [here](#) for the details. If the item is imported in Strife, it'll behave exactly as in Doom.

DECORATE definition

```
ACTOR InvulnerabilitySphere : PowerupGiver
{
+COUNTITEM
+INVENTORY.AUTOACTIVATE
+INVENTORY.ALWAYSPICKUP
+INVENTORY.BIGPOWERUP
Inventory.MaxAmount 0
Powerup.Type "Invulnerable"
Powerup.Color InverseMap
Inventory.PickupMessage "$GOTINVUL" // "Invulnerability!"
States
{
Spawn:
    PINV ABCD 6 Bright
    Loop
}
}
```

Medical kit
Actor type: Health
Game: Doom
DoomEd Number: 2012
Class Name: Medikit
Spawn ID: 24
Identifier: T_MEDKIT
Classes: Inventory'Health'Medikit

A health-restoring item in Doom. It restores 25 hit points.

DECORATE definition

ACTOR Medikit : Health

```
{
  Inventory.Amount 25
  Inventory.PickupMessage "$GOTMEDIKIT" // "Picked up a medikit."
  Health.LowMessage 25, "$GOTMEDINEED" // "Picked up a medikit that you REALLY need!"
  States
  {
    Spawn:
      MEDI A -1
      Stop
  }
}
```

Megasphere
Actor type: Powerup
Game: Doom
DoomEd Number: 83
Class Name: Megasphere
Spawn ID: 132
Identifier: T_MEGASPHERE
Classes: Inventoryâ†’CustomInventoryâ†’Megasphere

A powerup found in Doom 2. While the Soulsphere gives 100 health points up to a maximum of 200, the megasphere boosts the player's health to 200% instantly, no matter how low it was. In addition, it also provides the player with 200% armor and sets their armor absorption rate to 50% as if they had picked up a blue armor vest item.

DECORATE definition

```
ACTOR Megasphere : CustomInventory
{
+COUNTITEM
+INVENTORY.ALWAYSPICKUP
Inventory.PickupMessage "$GOTMSPHERE" // "MegaSphere!"
Inventory.PickupSound "misc/p_pkup"
States
{
Spawn:
    MEGA ABCD 6 Bright
    Loop
Pickup:
    TNT1 A 0 A_GiveInventory("BlueArmorForMegasphere", 1)
    TNT1 A 0 A_GiveInventory("MegasphereHealth", 1)
    Stop
}
}
```

Radiation-shielding suit
Actor type: Powerup
Game: Doom
DoomEd Number: 2025
Class Name: RadSuit
Spawn ID: 136
Identifier: T_IRONFEET
Classes: Inventory†PowerupGiver†RadSuit

This is the radiation suit powerup from Doom and Doom 2. It grants the wearer immunity from most damaging sector effects, but may randomly fail to prevent against injury from a sector effect that deals high damage (See Classes:PowerIronFeet for a full list of sector effects). The user's screen is tinted green while the powerup is in effect. Effective for 1 minute (2100 tics).

DECORATE definition

```
ACTOR RadSuit : PowerupGiver
{
    Height 46
    +INVENTORY.AUTOACTIVATE
    +INVENTORY.ALWAYSPICKUP
    Inventory.MaxAmount 0
    Inventory.PickupMessage "$GOTSUIT" // "Radiation Shielding Suit"
    Powerup.Type "IronFeet"
    States
    {
        Spawn:
            SUIT A -1 Bright
            Stop
    }
}
```


Soul sphere
Actor type: Health
Game: Doom
DoomEd Number: 2013
Class Name: Soulsphere
Spawn ID: 25
Identifier: T_SOULSPHERE
Classes: Inventory'Health'Soulsphere

The soul sphere is a blue orb with a humanoid face inside. When picked up, it grants the player a mighty 100 point health boost. Unlike the medikit and the stimpack, the soul sphere's health boost can go above the 100 health point limit, up to 200 points. A related, yet less powerful, item with a similar effect is the health bonus.

DECORATE definition

```
ACTOR Soulsphere : Health
{
+COUNTITEM
+INVENTORY.AUTOACTIVATE
+INVENTORY.ALWAYSPICKUP
+INVENTORY.FANCYPICKUPSOUND
Inventory.Amount 100
Inventory.MaxAmount 200
Inventory.PickupMessage "$GOTSUPER" // "Supercharge!"
Inventory.PickupSound "misc/p_pkup"
States
{
Spawn:
    SOUL ABCDCB 6 Bright
    Loop
}
}
```

Stimulant pack
Actor type: Health
Game: Doom
DoomEd Number: 2011
Class Name: Stimpack
Spawn ID: 23
Identifier: T_STIMPACK
Classes: Inventoryâ†’Healthâ†’Stimpack

A health restoring item in Doom. It restores 10 hit points.

DECORATE definition

```
ACTOR Stimpack : Health
{
    Inventory.Amount 10
    Inventory.PickupMessage "$GOTSTIM" // "Picked up a stimpack."
    States
    {
        Spawn:
            STIM A -1
            Stop
    }
}
```

Stealth Arachnotron
Actor type: Monster
Game Doom 2
DoomEd Number: 9050
Class Name: StealthArachnotron
Spawn ID: 117
Identifier: T_STEALTHARACHNOTRON
Classes: Arachnotron†StealthArachnotron

A stealth version of the arachnotron.

DECORATE definition

```
ACTOR StealthArachnotron : Arachnotron
{
+STEALTH
RenderStyle Translucent
Alpha 0
Obituary "$OB_STEALTHBABY"
}
```

Stealth Arch-Vile

Actor type: Monster

Game: Doom 2

DoomEd Number: 9051

Class Name: StealthArchvile

Spawn ID: 118

Identifier: T_STEALTHARCHVILE

Classes: Archvile†StealthArchvile

A stealth version of the arch-vile.

DECORATE definition

ACTOR StealthArchvile : Archvile

```
{  
+STEALTH  
RenderStyle Translucent  
Alpha 0  
Obituary "$OB_STEALTHVILE"  
}
```

Stealth Baron of Hell
Actor type: Monster
Game: Doom
DoomEd Number: 9052
Class Name: StealthBaron
Spawn ID: 100
Identifier: T_STEALTHBARON
Classes: BaronOfHell†StealthBaron

A stealth version of the Baron of Hell.

DECORATE definition

```
ACTOR StealthBaron : BaronOfHell
{
+STEALTH
RenderStyle Translucent
Alpha 0
Obituary "$OB_STEALTHBARON"
HitObituary "$OB_STEALTHBARON"
}
```

Stealth Cacodemon

Actor type Monster

Game: Doom

DoomEd Number: 9053

Class Name: StealthCacodemon

Spawn ID: 119

Identifier: T_STEALTHCACODEMON

Classes: Cacodemonâ†’StealthCacodemon

A stealth version of the cacodemon.

DECORATE definition

ACTOR StealthCacodemon : Cacodemon

```
{  
+STEALTH  
RenderStyle Translucent  
Alpha 0  
Obituary "$OB_STEALTHCACO"  
HitObituary "$OB_STEALTHCACO"  
}
```

Stealth Chaingunner
Actor type: Monster
Game: Doom 2
DoomEd Number: 9054
Class Name: StealthChaingunGuy
Spawn ID: 120
Identifier: T_STEALTHCHAINGUY
Classes: ChaingunGuy†'StealthChaingunGuy

A stealth version of the chaingunner.

DECORATE definition

```
ACTOR StealthChaingunGuy : ChaingunGuy
{
+STEALTH
RenderStyle Translucent
Alpha 0
Obituary "$OB_STEALTHCHAINGUY"
}
```

Stealth Demon

Actor type: Monster

Game: Doom

DoomEd Number: 9055

Class Name: StealthDemon

Spawn ID: 121

Identifier:T_STEALTHSERGEANT

Classes: Demon+StealthDemon

A stealth version of the demon, not to be confused with a spectre: the Spectre is always "fuzzy", but the stealth demon is totally invisible except when attacking or taking damage.

DECORATE definition

ACTOR StealthDemon : Demon

```
{
+STEALTH
RenderStyle Translucent
Alpha 0
Obituary "$OB_STEALTHDEMON"
HitObituary "$OB_STEALTHDEMON"
}
```


Stealth Imp
Actor type: Monster
Game: Doom
DoomEd Number: 9057
Class Name: StealthDoomImp
Spawn ID: 122
Identifier: T_STEALTHIMP
Classes: Actorâ†’DoomImpâ†’StealthDoomImp

A stealth version of the imp.

DECORATE definition

```
ACTOR StealthDoomImp : DoomImp
{
+STEALTH
RenderStyle Translucent
Alpha 0
Obituary "$OB_STEALTHIMP"
HitObituary "$OB_STEALTHIMP"
}
```

Stealth Mancubus
Actor type: Monster
Game: Doom 2
DoomEd Number: 9058
Class Name: StealthFatso
Spawn ID: 123
Identifier: T_STEALTHMANCUBUS
Classes: Fatso†StealthFatso

A stealth version of the mancubus.

DECORATE definition

```
ACTOR StealthFatso : Fatso
{
+STEALTH
RenderStyle Translucent
Alpha 0
Obituary "$OB_STEALTHFATSO"
}
```

Stealth Hell Knight
Actor type: Monster
Game: Doom 2
DoomEd Number: 9056
Class Name: StealthHellKnight
Spawn ID: 101
Identifier: T_STEALTHKNIGHT
Classes: BaronOfHellâ†’HellKnightâ†’StealthHellKnight

A stealth version of the hell knight.

DECORATE definition

```
ACTOR StealthHellKnight : HellKnight
{
+STEALTH
RenderStyle Translucent
Alpha 0
Obituary "$OB_STEALTHKNIGHT"
HitObituary "$OB_STEALTHKNIGHT"
}
```

Stealth Revenant

Actor type: Monster

Game: Doom 2

DoomEd Number: 9059

Class Name: StealthRevenant

Spawn ID: 124

Identifier: T_STEALTHREVENANT

Classes: Revenant†StealthRevenant

A stealth version of the revenant.

DECORATE definition

ACTOR StealthRevenant : Revenant

```
{
+STEALTH
RenderStyle Translucent
Alpha 0
Obituary "$OB_STEALTHUNDEAD"
HitObituary "$OB_STEALTHUNDEAD"
}
```

Stealth Sergeant
Actor type: Monster
Game: Doom
DoomEd Number: 9060
Class Name: StealthShotgunGuy
Spawn ID: 103
Identifier: T_STEALTHSHOTGUY
Classes: ShotgunGuy†StealthShotgunGuy

A stealth version of the shotgun guy.

DECORATE definition

```
ACTOR StealthShotgunGuy : ShotgunGuy
{
+STEALTH
RenderStyle Translucent
Alpha 0
Obituary "$OB_STEALTHSHOTGUNGUY"
}
```

Stealth Zombie

Actor type: Monster

Game: Doom

DoomEd Number: 9061

Class Name: StealthZombieMan

Spawn ID: 102

Identifier: T_STEALTHZOMBIE

Classes: ZombieMan+StealthZombieMan

A stealth version of zombie man.

DECORATE definition

ACTOR StealthZombieMan : ZombieMan

```
{  
+STEALTH  
RenderStyle Translucent  
Alpha 0  
Obituary "$OB_STEALTHZOMBIE"  
}
```

Burning barrel
Actor type: Light source
Game: Doom 2
DoomEd Number: 70
Class Name: BurningBarrel
Spawn ID: 149
Identifier: T_FLAMINGBARREL
Classes: BurningBarrel

A metal barrel similar to the explosive barrels, but its content is burning with large flames. It does not explode.

DECORATE definition

```
ACTOR BurningBarrel
{
    Radius 16
    Height 32
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            FCAN ABC 4 Bright
            Loop
    }
}
```

Light column
Actor type: Light source
Game: Doom
DoomEd Number: 2028
Class Name: Column
Classes: Column

An illuminated bollard, metallic with a yellow lamp.

DECORATE definition

```
ACTOR Column
{
    Radius 16
    Height 48
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            COLU A -1 Bright
            Stop
    }
}
```


Explosive barrel
Actor type: Hazard
Game: Doom
DoomEd Number: 2035
Class Name: ExplosiveBarrel
Spawn ID: 125
Identifier: T_BARREL
Classes: ExplosiveBarrel

A barrel full of green goop that can be destroyed, causing explosive damage. Several barrels near each other can be used to create large chain-reactions, as shown in the Doom 2 map "Barrels O' Fun". Barrels will respawn in multiplayer games if the sv_barrelrespawn CVAR is set.

DECORATE definition

```
ACTOR ExplosiveBarrel
{
    Health 20
    Radius 10
    Height 42
    +SOLID
    +SHOOTABLE
    +NOBLOOD
    +ACTIVATEMCROSS
    +DONTGIB
    +NOICEDDEATH
    +OLDRADIUSDMG
    DeathSound "world/barrelx"
    Obituary "$OB_BARREL" // "%o went boom."
    States
    {
        Spawn:
            BAR1 AB 6
            Loop
        Death:
            BEXP A 5 Bright
            BEXP B 5 Bright A_Scream
            BEXP C 5 Bright
            BEXP D 5 Bright A_Explode
            BEXP E 10 Bright
            TNT1 A 1050 Bright A_BarrelDestroy
            TNT1 A 5 A_Respawn
            Wait
    }
}
```

Tech lamp
Actor type: Light source
Game: Doom 2
DoomEd Number: 85
Class Name: TechLamp
Classes: TechLamp

A tall metallic lamp that's glowing with bright white to blue light.

DECORATE definition

```
ACTOR TechLamp
{
    Radius 16
    Height 80
    ProjectilePassHeight -16
    +SOLID
    States
    {
    Spawn:
        TLMP ABCD 4 Bright
        Loop
    }
}
```

Tech lamp
Actor type: Light source
Game: Doom 2
DoomEd Number: 86
Class Name: TechLamp2
Classes: TechLamp2

A short metallic lamp that's glowing with bright white to blue light.

DECORATE definition

```
ACTOR TechLamp2
{
    Radius 16
    Height 60
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            TLP2 ABCD 4 Bright
            Loop
    }
}
```

Tech pillar
Actor type: Decoration
Game: Doom
DoomEd Number: 48
Class Name: TechPillar
Classes: TechPillar

A tall column covered with electronics.

DECORATE definition

```
ACTOR TechPillar
{
    Radius 16
    Height 128
    ProjectilePassHeight -16
    +SOLID
    States
    {
        Spawn:
            ELEC A -1
            Stop
    }
}
```

Space Marine with Berserk Pack
Actor type: Monster
Game: Doom
DoomEd Number: 9102
Class Name: MarineBerserk
Classes: ScriptedMarine†'MarineBerserk

A scripted marine with a fist weapon in berserk mode.

DECORATE definition

ACTOR MarineBerserk: ScriptedMarine

```
{
  States
  {
    Melee:
      Goto Super::Melee.Berserk
    Missile:
      Stop
  }
}
```

Space Marine with BFG
Actor type: Monster
Game: Doom
DoomEd Number: 9111
Class Name: MarineBFG
Classes: ScriptedMarine†’MarineBFG

A scripted marine with a BFG 9000

DECORATE definition

```
ACTOR MarineBFG: ScriptedMarine
{
    States
    {
        Missile:
            Goto Super::Missile.BFG
    }
}
```

Space Marine with Chaingun
Actor type: Monster
Game: Doom
DoomEd Number: 9107
Class Name: MarineChaingun
Classes: ScriptedMarine†MarineChaingun

A scripted marine with a chaingun.

DECORATE definition

```
ACTOR MarineChaingun: ScriptedMarine
{
    States
    {
        Missile:
            Goto Super::Missile.Chaingun
    }
}
```

Space Marine with Chainsaw
Actor type: Monster
Game: Doom
DoomEd Number: 9103
Class Name: MarineChainsaw
Classes: ScriptedMarine†’MarineChainsaw

A scripted marine with a chainsaw.

DECORATE definition

ACTOR MarineChainsaw: ScriptedMarine
{
 States
 {
 Melee:
 Goto Super::Melee.Chainsaw
 Missile:
 Stop
 }
}

Bare-Handed Space Marine
Actor type: Monster
Game: Doom
DoomEd Number: 9101
Class Name: MarineFist
Classes: ScriptedMarine†’MarineFist

A scripted marine with just his fists.

DECORATE definition

```
ACTOR MarineFist: ScriptedMarine
{
    States
    {
        Melee:
            Goto Super::Melee.Fist
        Missile:
            Stop
    }
}
```

Space Marine with Pistol
Actor type: Monster
Game: Doom
DoomEd Number: 9104
Class Name: MarinePistol
Classes: ScriptedMarine†'MarinePistol

A scripted marine with a standard-issue pea shooter.

DECORATE definition

```
ACTOR MarinePistol: ScriptedMarine
{
    States
    {
        Missile:
            Goto Super::Missile.Pistol
    }
}
```

Space Marine with Plasma Rifle
Actor type: Monster
Game: Doom
DoomEd Number: 9109
Class Name MarinePlasma
Classes: ScriptedMarine†'MarinePlasma

A scripted marine with a plasma rifle.

DECORATE definition

```
ACTOR MarinePlasma: ScriptedMarine
{
    States
    {
        Missile:
            Goto Super::Missile.Plasma
    }
}
```

Space Marine with Railgun
Actor type: Monster
Game: Doom
DoomEd Number: 9110
Class Name MarineRailgun
Classes: ScriptedMarine†’MarineRailgun

A scripted marine toting a railgun.

DECORATE definition

```
ACTOR MarineRailgun: ScriptedMarine
{
    States
    {
        Missile:
            Goto Super::Missile.Railgun
    }
}
```

Space Marine with Rocket Launcher
Actor type:Monster
Game: Doom
DoomEd Number: 9108
Class Name: MarineRocket
Classes: ScriptedMarine†MarineRocket

A scripted marine with a rocket launcher.

DECORATE definition

```
ACTOR MarineRocket : MarineFist
{
    States
    {
        Missile:
            Goto Super::Missile.Rocket
    }
}
```

Space Marine with Shotgun

Actor type: Monster

Game: Doom

DoomEd Number: 9105

Class Name: MarineShotgun

Classes: ScriptedMarine†MarineShotgun

A scripted marine with a pump-action shotgun.

DECORATE definition

ACTOR MarineShotgun : ScriptedMarine

```
{
  States
  {
    Missile:
      Goto Super::Missile.Shotgun
  }
}
```

Space Marine with Super Shotgun
Actor type: Monster
Game: Doom
DoomEd Number: 9106
Class Name: MarineSSG
Classes: ScriptedMarine†MarineSSG

A scripted marine with a double-barreled sawed-off shotgun.

DECORATE definition

```
ACTOR MarineSSG : ScriptedMarine
{
    States
    {
        Missile:
            Goto Super::Missile.SSG
    }
}
```

Dog
Actor type: Monster
Game: ZDoom
DoomEd Number: 888
Class Name: MBFHelperDog
Classes: MBFHelperDog

The Marines' Best Friends, who gave their name to the MBF port, intended as an alternative to coop bots. They have an extremely high amount of hit points (as many as a Hell Knight!) to compensate for their inability to pick up health items such as medikits. This makes them unbalanced if used as enemies; but despite that, they are not allied to the player by default.

DECORATE definition

```
ACTOR MBFHelperDog
{
    Health 500
    Speed 10
    PainChance 180
    Radius 12
    Height 28
    Mass 100
    Monster
    +JUMPDOWN
    ActiveSound "dog/active"
    AttackSound "dog/attack"
    DeathSound "dog/death"
    PainSound "dog/pain"
    SeeSound "dog/sight"
    Obituary "$OB_DOG" // "%o was mauled by a dog."
    States
    {
    Spawn:
        DOGS AB 10 A_Look
        Loop
    See:
        DOGS AABBCDD 2 A_Chase
        Loop
    Melee:
        DOGS EF 8 A_FaceTarget
        DOGS G 8 A_SargAttack
        Goto See
    Pain:
        DOGS H 2
        DOGS H 2 A_Pain
        Goto See
    Death:
        DOGS I 8
        DOGS J 8 A_Scream
        DOGS K 4
        DOGS L 4 A_Fall
        DOGS M 4
        DOGS N -1
```


Raise:
DOGS NMLKJI 5
Goto See
}
}

Marine
Actor type: Monster
Game: Doom
DoomEd Number: 9100
Class Name: ScriptedMarine
Spawn ID: 151
Identifier: T_SCRIPTEDMARINE
Classes: ScriptedMarine
→MarineBFG
→MarineBerserk
→MarineChaingun
→MarineChainsaw
→MarineFist
→MarinePistol
→MarinePlasma
→MarineRailgun
→MarineRocket
→MarineSSG
→MarineShotgun

The Scripted Marine is a duplication of the Doom player which can be controlled via scripting during cutscenes or gameplay, to simulate the player or other marines wearing the same armor. Though the actor has several attacks coded, he does not actually ever use any by default. These attack states are used, however, by its children classes.

As with all marines, you can use the special ACS functions SetMarineSprite and SetMarineWeapon to change this actor's appearance and weapon.

DECORATE definition

ACTOR ScriptedMarine native

```
{
  Health 100
  Radius 16
  Height 56
  Mass 100
  Speed 8
  PainChance 160
  Monster
  -COUNTKILL
  Translation 0
  Damage 100
  DeathSound "*death"
  PainSound "*pain50"

  action native A_M_Refire(bool ignoremissile=false);
  action native A_M_CheckAttack();
  action native A_MarineChase();
  action native A_MarineLook();
  action native A_MarineNoise();
  action native A_M_Punch(int force);
  action native A_M_SawRefire();
  action native A_M_FirePistol(bool accurate);
  action native A_M_FireShotgun();
  action native A_M_FireShotgun2();
  action native A_M_FireCGun(bool accurate);
```

```
action native A_M_FireMissile();
action native A_M_FirePlasma();
action native A_M_FireRailgun();
action native A_M_BFGSound();
action native A_M_FireBFG();
```

States

```
{
```

Spawn:

```
PLAY A 4 A_MarineLook
PLAY A 4 A_MarineNoise
Loop
```

Idle:

```
PLAY A 4 A_MarineLook
PLAY A 4 A_MarineNoise
PLAY A 4 A_MarineLook
PLAY B 4 A_MarineNoise
PLAY B 4 A_MarineLook
PLAY B 4 A_MarineNoise
Loop
```

See:

```
PLAY ABCD 4 A_MarineChase
Loop
```

Melee.Fist:

```
PLAY E 4 A_FaceTarget
PLAY E 4 A_M_Punch(1)
PLAY A 9
PLAY A 0 A_M_Refire(1)
Loop
PLAY A 5 A_FaceTarget
Goto See
```

Melee.Berserk:

```
PLAY E 4 A_FaceTarget
PLAY E 4 A_M_Punch(10)
PLAY A 9
PLAY A 0 A_M_Refire(1)
Loop
PLAY A 5 A_FaceTarget
Goto See
```

Melee.Chainsaw:

```
PLAY E 4 A_MarineNoise
PLAY E 4 A_M_Saw
PLAY E 0 A_M_SawRefire
Goto Melee.Chainsaw+1
PLAY A 0
Goto See
```

Missile:

Missile.None:

```
PLAY E 12 A_FaceTarget
Goto Idle
PLAY F 6 Bright
Loop
```

Missile.Pistol:

```
PLAY E 4 A_FaceTarget
PLAY F 6 Bright A_M_FirePistol(1)
```

PLAY A 4 A_FaceTarget
PLAY A 0 A_M_Refire
PLAY A 5
Goto See
Fireloop.Pistol:
PLAY F 6 Bright A_M_FirePistol(0)
PLAY A 4 A_FaceTarget
PLAY A 0 A_M_Refire
Goto Fireloop.Pistol
PLAY A 5
Goto See
Missile.Shotgun:
PLAY E 3 A_M_CheckAttack
PLAY F 7 Bright A_M_FireShotgun
Goto See
Missile.SSG:
PLAY E 3 A_M_CheckAttack
PLAY F 7 Bright A_M_FireShotgun2
Goto See
Missile.Chaingun:
PLAY E 4 A_FaceTarget
PLAY FF 4 Bright A_M_FireCGun(1)
PLAY FF 4 Bright A_M_FireCGun(0)
PLAY A 0 A_M_Refire
Goto Missile.Chaingun+3
PLAY A 0
Goto See
Missile.Rocket:
PLAY E 8
PLAY F 6 Bright A_M_FireMissile
PLAY E 6
PLAY A 0 A_M_Refire
Loop
PLAY A 0
Goto See
Missile.Plasma:
PLAY E 2 A_FaceTarget
PLAY E 0 A_FaceTarget
PLAY F 3 Bright A_M_FirePlasma
PLAY A 0 A_M_Refire
Goto Missile.Plasma+1
PLAY A 0
Goto See
Missile.Railgun:
PLAY E 4 A_M_CheckAttack
PLAY F 6 Bright A_M_FireRailgun
Goto See
Missile.BFG:
PLAY E 5 A_M_BFGSound
PLAY EEEEE 5 A_FaceTarget
PLAY F 6 Bright A_M_FireBFG
PLAY A 4 A_FaceTarget
PLAY A 0 A_M_Refire
Loop
PLAY A 0
Goto See

SkipAttack:

PLAY A 1

Goto See

Pain:

PLAY G 4

PLAY G 4 A_Pain

Goto Idle

Death:

PLAY H 10

PLAY I 10 A_Scream

PLAY J 10 A_NoBlocking

PLAY KLM 10

PLAY N -1

Stop

XDeath:

PLAY O 5

PLAY P 5 A_XScream

PLAY Q 5 A_NoBlocking

PLAY RSTUV 5

PLAY W -1

Stop

Raise:

PLAY MLKJIH 5

Goto See

}

}

BFG 9000

Actor type: Weapon

Game: Doom

DoomEd Number: 2006

Class Name: BFG9000

Spawn ID: 31

Identifier: T_BFG

Classes: Inventoryâ†’Weaponâ†’DoomWeaponâ†’BFG9000

The BFG9000 is the deadliest and strongest weapon available in Doom. The weapon itself has a few effects not found in other weapons, such as the 40 hitscan tracers that fire out when the weapon hits a target, each doing 100 damage (cumulative with the damage from the projectile) to anything in the way. It uses 40 cell ammo per shot to function.

Note: The BFGBall class contains the obituary for the direct hit. For the tracers' obituary, you may add this line to the weapon's definition.

Obituary "\$OB_MPBFG_SPLASH" // "%o couldn't hide from %k's BFG"

DECORATE definition

ACTOR BFG9000 : DoomWeapon

```
{
  Height 20
  Weapon.SelectionOrder 2800
  Weapon.AmmoUse 40
  Weapon.AmmoGive 40
  Weapon.AmmoType "Cell"
  +WEAPON.NOAUTOFIRE
  Inventory.PickupMessage "$GOTBFG9000"
  Tag "$TAG_BFG9000"
  States
  {
    Ready:
      BFGG A 1 A_WeaponReady
      Loop
    Deselect:
      BFGG A 1 A_Lower
      Loop
    Select:
      BFGG A 1 A_Raise
      Loop
    Fire:
      BFGG A 20 A_BFGSound
      BFGG B 10 A_GunFlash
      BFGG B 10 A_FireBFG
      BFGG B 20 A_ReFire
      Goto Ready
    Flash:
      BFGF A 11 Bright A_Light1
      BFGF B 6 Bright A_Light2
      Goto LightDone
    Spawn:
      BFUG A -1
```

```
Stop
OldFire:
BFGG A 10 A_BFGSound
BFGG BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB 1 A_FireOldBFG
BFGG B 0 A_Light0
BFGG B 20 A_ReFire
Goto Ready
}
}
```

Chaingun
Actor type: Weapon
Game: Doom
DoomEd Number: 2002
Class Name: Chaingun
Spawn ID: 28
Identifier: T_CHAINGUN
Classes: Inventoryâ†’Weaponâ†’DoomWeaponâ†’Chaingun

The chaingun is a rapid fire bullet weapon. It is dropped by chaingunners and uses clips for ammo.

DECORATE definition

```
ACTOR Chaingun : DoomWeapon
{
    Weapon.SelectionOrder 700
    Weapon.AmmoUse 1
    Weapon.AmmoGive 20
    Weapon.AmmoType "Clip"
    Inventory.PickupMessage "$GOTCHAINGUN" // "You got the chaingun"
    Obituary "$OB_MPCHAINGUN" // "%o was mowed down by %k's chaingun."
    Tag "$TAG_CHAINGUN"
    States
    {
        Ready:
            CHGG A 1 A_WeaponReady
            Loop
        Deselect:
            CHGG A 1 A_Lower
            Loop
        Select:
            CHGG A 1 A_Raise
            Loop
        Fire:
            CHGG AB 4 A_FireCGun
            CHGG B 0 A_ReFire
            Goto Ready
        Flash:
            CHGF A 5 Bright A_Light1
            Goto LightDone
            CHGF B 5 Bright A_Light1
            Goto LightDone
        Spawn:
            MGUN A -1
            Stop
    }
}
```


Chainsaw
Actor type: Weapon
Game: Doom
DoomEd Number: 2005
Class Name: Chainsaw
Spawn ID: 32
Identifier: T_CHAINSAW
Classes: Inventory†Weapon†Chainsaw

The Chainsaw is a fast melee weapon.

DECORATE definition

```
ACTOR Chainsaw : Weapon
{
    Weapon.Kickback 0
    Weapon.SelectionOrder 2200
    Weapon.UpSound "weapons/sawup"
    Weapon.ReadySound "weapons/sawidle"
    Inventory.PickupMessage "$GOTCHAINSAW"
    Obituary "$OB_MPCHAINSAW"
    Tag "$TAG_CHAINSAW"
    +WEAPON.MELEEWEPON
    States
    {
        Ready:
            SAWG CD 4 A_WeaponReady
            Loop
        Deselect:
            SAWG C 1 A_Lower
            Loop
        Select:
            SAWG C 1 A_Raise
            Loop
        Fire:
            SAWG AB 4 A_Saw
            SAWG B 0 A_ReFire
            Goto Ready
        Spawn:
            CSAW A -1
            Stop
    }
}
```

Fist
Actor type: Weapon
Game: Doom
DoomEd Number: None
Class Name: Fist
Classes: Inventoryâ†’Weaponâ†’Fist

Your (hairy), brass-knuckled fist. Used as a melee weapon. Automatically selected as a last resort if you run out of ammo and do not have the chainsaw. Picking up the berserk pack also switches to this weapon.

DECORATE definition

```
ACTOR Fist : Weapon
{
    Weapon.SelectionOrder 3700
    Weapon.Kickback 100
    Obituary "$OB_MPFIST"
    Tag "$TAG_FIST"
    +WEAPON.WIMPY_WEAPON
    +WEAPON.MELEEWEPON
    States
    {
        Ready:
            PUNG A 1 A_WeaponReady
            Loop
        Deselect:
            PUNG A 1 A_Lower
            Loop
        Select:
            PUNG A 1 A_Raise
            Loop
        Fire:
            PUNG B 4
            PUNG C 4 A_Punch
            PUNG D 5
            PUNG C 4
            PUNG B 5 A_ReFire
            Goto Ready
    }
}
```

Pistol
Actor type: Weapon
Game: Doom
DoomEd Number: 5010
Class Name: Pistol
Classes: Inventoryâ†’Weaponâ†’DoomWeaponâ†’Pistol

The pistol. The weapon you start off with and the weakest long-ranged weapon in Doom. Uses clips for ammo.

DECORATE definition

```
ACTOR Pistol : DoomWeapon
{
    Weapon.SelectionOrder 1900
    Weapon.AmmoUse 1
    Weapon.AmmoGive 20
    Weapon.AmmoType "Clip"
    Obituary "$OB_MPPISTOL"
    +WEAPON.WIMPY_WEAPON
    Inventory.Pickupmessage "$PICKUP_PISTOL_DROPPED"
    Tag "$TAG_PISTOL"
    States
    {
        Ready:
            PISG A 1 A_WeaponReady
            Loop
        Deselect:
            PISG A 1 A_Lower
            Loop
        Select:
            PISG A 1 A_Raise
            Loop
        Fire:
            PISG A 4
            PISG B 6 A_FirePistol
            PISG C 4
            PISG B 5 A_ReFire
            Goto Ready
        Flash:
            PISF A 7 Bright A_Light1
            Goto LightDone
            PISF A 7 Bright A_Light1
            Goto LightDone
        Spawn:
            PIST A -1
            Stop
    }
}
```

Plasma rifle
Actor type: Weapon
Game: Doom
DoomEd Number: 2004
Class Name: PlasmaRifle
Spawn ID: 30
Identifier: T_PLASMAGUN
Classes: Inventoryâ†’Weaponâ†’DoomWeaponâ†’PlasmaRifle

The plasma rifle. A fast projectile-launching weapon which uses cells for ammo.

DECORATE definition

```
ACTOR PlasmaRifle : DoomWeapon
{
    Weapon.SelectionOrder 100
    Weapon.AmmoUse 1
    Weapon.AmmoGive 40
    Weapon.AmmoType "Cell"
    Inventory.PickupMessage "$GOTPLASMA"
    Tag "$TAG_PLASMARIFLE"
    States
    {
        Ready:
            PLSG A 1 A_WeaponReady
            Loop
        Deselect:
            PLSG A 1 A_Lower
            Loop
        Select:
            PLSG A 1 A_Raise
            Loop
        Fire:
            PLSG A 3 A_FirePlasma
            PLSG B 20 A_ReFire
            Goto Ready
        Flash:
            PLSF A 4 Bright A_Light1
            Goto LightDone
            PLSF B 4 Bright A_Light1
            Goto LightDone
        Spawn:
            PLAS A -1
            Stop
    }
}
```

Rocket launcher
Actor type: Weapon
Game: Doom
DoomEd Number: 2003
Class Name: RocketLauncher
Spawn ID: 29
Identifier: T_ROCKETLAUNCHER
Classes: Inventory†Weapon†DoomWeapon†RocketLauncher

The rocket launcher. It fires projectiles which cause splash damage. Not recommended for close range. Uses rockets for ammo.

DECORATE definition

```
ACTOR RocketLauncher : DoomWeapon
{
    Weapon.SelectionOrder 2500
    Weapon.AmmoUse 1
    Weapon.AmmoGive 2
    Weapon.AmmoType "RocketAmmo"
    +WEAPON.NOAUTOFIRE
    Inventory.PickupMessage "$GOTLAUNCHER"
    Tag "$TAG_ROCKETLAUNCHER"
    States
    {
        Ready:
            MISG A 1 A_WeaponReady
            Loop
        Deselect:
            MISG A 1 A_Lower
            Loop
        Select:
            MISG A 1 A_Raise
            Loop
        Fire:
            MISG B 8 A_GunFlash
            MISG B 12 A_FireMissile
            MISG B 0 A_ReFire
            Goto Ready
        Flash:
            MISF A 3 Bright A_Light1
            MISF B 4 Bright
            MISF CD 4 Bright A_Light2
            Goto LightDone
        Spawn:
            LAUN A -1
            Stop
    }
}
```

Shotgun
Actor type: Weapon
Game: Doom
DoomEd Number: 2001
Class Name: Shotgun
Spawn ID: 27
Identifier: T_SHOTGUN
Classes: Inventoryâ†’Weaponâ†’DoomWeaponâ†’Shotgun

The shotgun. A hitscan weapon which has a large spread. Uses shells for ammo.

DECORATE definition

```
ACTOR Shotgun : DoomWeapon
{
    Weapon.SelectionOrder 1300
    Weapon.AmmoUse 1
    Weapon.AmmoGive 8
    Weapon.AmmoType "Shell"
    Inventory.PickupMessage "$GOTSHOTGUN"
    Obituary "$OB_MPSHOTGUN"
    Tag "$TAG_SHOTGUN"
    States
    {
        Ready:
            SHTG A 1 A_WeaponReady
            Loop
        Deselect:
            SHTG A 1 A_Lower
            Loop
        Select:
            SHTG A 1 A_Raise
            Loop
        Fire:
            SHTG A 3
            SHTG A 7 A_FireShotgun
            SHTG BC 5
            SHTG D 4
            SHTG CB 5
            SHTG A 3
            SHTG A 7 A_ReFire
            Goto Ready
        Flash:
            SHTF A 4 Bright A_Light1
            SHTF B 3 Bright A_Light2
            Goto LightDone
        Spawn:
            SHOT A -1
            Stop
    }
}
```

Super shotgun
Actor type: Weapon
Game: Doom 2
DoomEd Number: 82
Class Name: SuperShotgun
Spawn ID: 33
Identifier: T_SUPERSHOTGUN
Classes: Inventoryâ†’Weaponâ†’DoomWeaponâ†’SuperShotgun

The super shotgun. Also known as the double-barreled shotgun. A powerful hitscan weapon with a large spread. Uses shells for ammo. This is the only weapon to not appear in the original Doom.

DECORATE definition

```
ACTOR SuperShotgun : DoomWeapon
{
    Weapon.SelectionOrder 400
    Weapon.AmmoUse 2
    Weapon.AmmoGive 8
    Weapon.AmmoType "Shell"
    Inventory.PickupMessage "$GOTSHOTGUN2"
    Obituary "$OB_MPSSHOTGUN"
    Tag "$TAG_SUPERSHOTGUN"
    States
    {
        Ready:
            SHT2 A 1 A_WeaponReady
            Loop
        Deselect:
            SHT2 A 1 A_Lower
            Loop
        Select:
            SHT2 A 1 A_Raise
            Loop
        Fire:
            SHT2 A 3
            SHT2 A 7 A_FireShotgun2
            SHT2 B 7
            SHT2 C 7 A_CheckReload
            SHT2 D 7 A_OpenShotgun2
            SHT2 E 7
            SHT2 F 7 A_LoadShotgun2
            SHT2 G 6
            SHT2 H 6 A_CloseShotgun2
            SHT2 A 5 A_ReFire
            Goto Ready
        // unused states
        SHT2 B 7
        SHT2 A 3
        Goto Deselect
    }
    Flash:
        SHT2 I 4 Bright A_Light1
        SHT2 J 3 Bright A_Light2
    }
```

```
    Goto LightDone
Spawn:
    SGN2 A -1
    Stop
}
}
```


Actor flags

Flags control various characteristics of your actor, varying from appearance to physical properties. Remember that you can get a basic set of flags by using the Monster or Projectile property keywords. However, you will invariably feel the need to dabble with at least a few of these flags. To use, simply specify the flag within the actor's DECORATE definition and outside the state block. You can set or clear a flag as follows:

+FLAGNAME sets a flag
-FLAGNAME clears a flag

General flags

The following flags can be used with any actor.

Renderer

INTERPOLATEANGLES

Actor angles are constantly interpolated over the span of one tic. This includes angle, pitch and roll. Interpolation smoothly transitions from the previous angles to the next instead of instantly snapping.

FLATSPRITE

Actor becomes a flat sprite which can be tilted with the use of the Pitch actor property.

NOTE: This flag is not compatible with WALLSPRITE.

ROLLSPRITE

Actor sprite is affected by roll, rotating the sprite.

WALLSPRITE

Similar to FLATSPRITE but is not affected by pitch.

NOTE: This flag is not compatible with FLATSPRITE.

ROLLCENTER

Prior to this flag's introduction, rolling occurred at the very center of the sprite no matter what offsets were used. Offsets are now used instead of the center. Using this flag restores the original behavior of displacing via the center instead of offsets.

SPRITEANGLE

Enables the use of the SpriteAngle actor property.

SPRITEFLIP

Actor sprite is flipped on the x-axis in a special way. If the current sprite is mirrored (such as POSSA2A8 versus CYBRA2), this flag has no effect.

XFLIP

Actor sprite is flipped on the x-axis.

YFLIP

Actor sprite is flipped on the y-axis.

MASKROTATION

Enables rendering of a sprite based on defined angles through the use of VisibleAngles and VisiblePitch.

ABSMASKANGLE

The visible angle becomes absolute -- not offset by the actor's current angle.

ABSMASKPITCH

Same as ABSMASKANGLE but for pitch.

DONTINTERPOLATE

Position interpolation is disabled for this actor.

ZDOOMTRANS

Marks the actor's RenderStyle as being a ZDoom graphical enhancement over the original. An actor with this flag will change its RenderStyle to Normal (opaque) when r_vanillatrans is set to 1 or 3. Custom content should generally not use this flag, though it can be desired for visual consistency with vanilla content for actors such as fireball projectiles or teleportation flashes.

ABSVIEWANGLES

The ViewAngle/Pitch/Roll properties are an offset to the current actor's angle/pitch/roll. With this flag, they become absolute.

CASTSPRITESHADOW

Forces the actor to cast sprite shadows, this can be used to make a monster cast a sprite shadow if the setting is set to default, or to make non-monster actors such as decorations cast shadows as well.

NOSPRITESHADOW

Entirely disables shadow casting for the actor regardless of what the setting is set to, can be used to make a monster not cast shadows, can't be used to turn shadows off for players.

MASTERNOSEE

Actors with this flag on will not appear on the view of the actor who they have set as their master. For example, an orb with this flag that orbits around the player, will not render on the players' first person view, but can be seen if the player sees themselves through another camera, or looks at themselves through a mirror or a portal.

ADDLIGHTLEVEL

Makes the actors' LightLevel be added on top of the existing sector light level. Instead of being an absolute value.

INVISIBLEINMIRRORS

The actor will not render on mirrors.

ONLYVISIBLEINMIRRORS

The actor will only be visible when seen through mirrors.

Physics

SOLID

Set when the object should be solid (blocking). The size of the blocking is defined using the height and radius properties.

→ Automatically given by the Monster combo

SHOOTABLE

Object can be damaged. If health goes below 0 it enters its death state.

→ Automatically given by the Monster combo

FLOAT

Floating actor that can change height at will (usually used for monsters). The speed of the float can be defined with its FloatSpeed property. Actors will not be able to float properly unless it has NOGRAVITY set. If set with FLOORCLIP, the actor will be able to walk off edges from heights.

NOGRAVITY

Actor is not subject to gravity.

→ Automatically given by the Projectile combo

WINDTHRUST

Actor is thrust by Heretic/Hexen wind sector specials, and pushers/pullers.

PUSHABLE

Actor can be pushed.

DONTFALL

Doesn't fall down after being killed.

CANPASS

Actor uses height sensitive collision detection. Use with care! This only makes sense on actors that can move by themselves. This flag is needed so that an actor can stand on sprite bridges.

→ Automatically given by the Monster combo

ACTLIKEBRIDGE

Uses the special collision logic for bridge things. Most importantly bridge things allow everything else to pass above and beneath regardless of the setting of the CANPASS flag.

NOBLOCKMAP

This object is excluded from passive collision detection. Nothing else can run into a NOBLOCKMAP object but the object itself can run into others. Actors with this flag cannot be hit by hitscan/trace functions such as A_FireBullets, A_RailAttack, etc.

Note: If an object had this flag set needs to be restored to being a solid object, A_ChangeLinkFlags(0) may need to be called manually to reset its blockmap data.

→ Automatically given by the Projectile combo

MOVEWITHSECTOR

Actor follows sector height changes unconditionally. Normally actors with the NOBLOCKMAP flag are excluded from moving with sectors. Another use is to ensure that an actor remains on the floor of extremely fast moving lifts.

RELATIVETO FLOOR

An actor not bound by gravity with this flag retains its z-height relative to a moving floor.

NOLIFTDROP

Does not drop when a lift under it lowers.

SLIDESONWALLS

Actor can slide along walls. Used for players and actors that are pushed against walls.

NODROPOFF

Makes tall dropoffs block any movement by this actor. Normally this is only being done for monster movement but not for thrust by damage or scrolling sectors.

NOFORWARDFALL

On random occasions and under certain circumstances, an actor on high ledges will be pushed and will fall forward when being damaged by an attack. This flag, when set, will disable that behavior. Alternatively, the flag can be set on the damage inflictor (e.g., a projectile or a puff for a hitscan attack). In this case, the behavior is only disabled if the actor got damaged by an inflictor which has the flag set.

NOTRIGGER

Disables all line action (crossing, impacting, using, etc.) for the actor.

BLOCKEDBYSOLIDACTORS

An actor with this flag is blocked by solid actors, even if itself non-solid. If the actor does not have the CANPASS flag as well, solid things are infinitely tall for it. An effect of this flag is that PoisonCloud actors spawned from a ZPoisonShroom are stuck in the mushroom (which is solid) and unable to be pushed away by the Disc of Repulsion, whereas the same actor spawned from a PoisonBag can be blasted away.

BLOCKASPLAYER

A non-player actor (including projectiles) with this flag is blocked by the same lines that block a player.

NOFRICTION

All friction effects are disabled on the actor with this flag set, including the speed cap from water and crouching.

NOFRICTIONBOUNCE

Actor does not bounce off walls as a result of sliding into them on slippery floors.

FALLDAMAGE

Monster can be damaged by falling. A falling monster always dies upon hitting the floor if its downward velocity is high enough for it to be considered for damage, unless the ProperMonsterFallingDamage MAPINFO flag is also set on the map.

To enable falling damage for all monsters on a map, set the MonsterFallingDamage MAPINFO flag on that map.

ALLOWTHRUBITS

Enables the use of the ThruBits property.

CROSSLINECHECK

Enables the use of the CanCrossLine virtual function.

Behavior

ALWAYSRESPAWN

A monster with this flag will respawn in any skill level as though the skill level was set to Nightmare. Note, this flag is for monsters, not for inventory items; Inventory items use INVENTORY.ALWAYSRESPAWN. If dealing with an Inventory pointer (or derivative), you will need to cast to Actor first in order to access this flag. The only case when this distinction is not relevant is in a DECORATE definition, since prefixes are optional and using +ALWAYSRESPAWN will set the correct flag depending on the type of actor.

AMBUSH

Monster is set to ambush players: The monster will not start chasing the player after hearing player weapons until it has a direct line of sight to him (after being woken up, however, it does not need to be facing him). Normally this is set in an editor on a per-object basis. Note that some editors call this flag "Deaf," but this is a misnomer. The monster can still hear the player with this flag set!

AVOIDMELEE

Monster backs away from melee combat when too close to its target, provided it has a valid Missile state. A bot with this flag needs to have a ranged weapon selected.

To enable this behavior for all monsters on a map, set the AvoidMelee MAPINFO flag on that map.

BOSS

Actor is a boss, and has the following special properties:

Plays See sound and Death sound at full volume (regardless of distance), and play these sounds in surround mode during the cast endgame.

Cannot be squashed (normally instant death) by Heretic's powered-up Mace.

Cannot be teleported by Hexen's Banishment Device.

Cannot be pushed by items such as Hexen's Disc of Repulsion, or damaged by other monsters thrown at the boss by these items.

Takes only one-quarter of the damage from Hexen's Wraithverge spirits, burns said spirit out faster than other monsters, and reflects them against the attacker if the boss also has the REFLECTIVE flag.

Has a chance to resist Hexen's Arcs of Death.

Does not take fire damage when indirectly affected by Hexen's Bloodscourge's projectiles.

Takes only 50 damage from a poison bolt, instead of instantly killed.

DONTCORPSE

Monsters with this flag will not automatically get the CORPSE flag when killed. It is the modder's duty to explicitly set the flag in the monster's death sequence. This has two main effects: first, the actor cannot be immediately passed through, and secondly it will not immediately enter a Crash state if it is on the floor or another actor. The purpose is to allow modders greater control on death animations and special effects with flying monsters.

DONTFACETALKER

Actor does not turn to face the player in a conversation.

DORMANT

Actor is dormant and has to be activated with Thing_Activate. Dormant actors can't do anything and don't take damage.

FRIENDLY

This monster doesn't target the player. Instead it attacks other monsters. Currently friendly monsters only target unfriendly monsters when they see the Player, and unfriendly monsters will never attack friendly monsters unless they are attacked first or have the SEEFRIENDLYMONSTERS flag on. Also note that monsters resurrected or spawned by friendly monsters (Arch-Vile, Pain Elementals, Icon of Sin, etc.) are friendly; and inversely a dead friendly monster resurrected by an unfriendly Arch-Vile is no longer friendly.

JUMPDOWN

Actors with this flag set will often (about 92% chance) decide to jump down tall ledges to pursue their target, if said target is hostile and is relatively close (horizontally within 144 map units).

LOOKALLAROUND

Looks in all directions for targets, not just in front of itself when A_Look or one of its variants is called.

MISSILEEVENMORE

Increases the probability of a missile attack from farther away even more than MISSILEMORE. Both flags can be combined for extra aggressive monsters.

MISSILEMORE

Increases the probability of a missile attack from farther away, though less than MISSILEEVENMORE. Both flags can be combined for extra aggressive monsters.

NEVERRESPAWN

A monster with this flag will never respawn, even in Nightmare.

NOSPLASHALERT

This monster is not alerted by terrain splash sounds like landing in water. Splashes are defined in TERRAIN and are not always literally "splashes".

NOTARGETSWITCH

A monster with this flag never switches its target unless its current target is dead.

NOVERTICALMELEERANGE

A monster with this flag ignores vertical distance to its target when checking for melee range. Used, for example, by the Strife Stalker to drop on its target from the ceiling, regardless of ceiling height.

QUICKTORETALIATE

Normally, when an actor acquires a target, it is prevented from switching to a new target for until its Threshold value gets reduced to 0 by repeated calls to A_Chase. If this flag is set, the monster is exempted from that code and is free to switch targets repeatedly. Monsters with this set will tend to turn on a new attacker immediately, rather than focusing on their current target. The arch-vile uses this flag.

STANDSTILL

It is used to prevent inactive monsters from walking around without having something to attack. Strife's special logic for humanoids and ZDoom's friendly monsters use this.

AVOIDHAZARDS

The monster will actively try to avoid being harmed by crushing ceilings. Note: This flag ONLY works for

crushing ceilings ! It will not make the actor react to other level hazards.

STAYONLIFT

Will stay still on moving platforms, instead of moving around on them, and begin moving again once they stop.

DONTFOLLOWPLAYERS

Friendly monsters with this flag will not follow the player they are allied to, when they have no target or goal left to deal with.

SEEFRIENDLYMONSTERS

Hostile monsters with this flag on will attack friendly monsters on sight. The sight behavior is identical to that of friendly monsters.

(In)Abilities

CANNOTPUSH

This actor cannot push pushable objects.

NOTELEPORT

Actor cannot teleport.

→ Automatically given by the Projectile combo

ACTIVATEIMPACT

Upon hitting a wall this actor can activate G1/GR lines.

→ Automatically given by the Projectile combo

CANPUSHWALLS

Upon hitting a wall this actor can activate P1/PR lines.

→ Automatically given by the Monster combo

CANUSEWALLS

This actor can activate unlocked doors and lifts. (Original Doom monster behavior)

→ Automatically given by the Monster combo

ACTIVATEMCROSS

This actor can activate 'Monster crosses' lines.

→ Automatically given by the Monster combo

ACTIVATEPCROSS

This actor can activate 'Projectile crosses' lines.

→ Automatically given by the Projectile combo

CANTLEAVEFLOORPIC

This actor cannot cross into a sector with a different floor texture.

TELESTOMP

This actor can telefrag others. Actors without this flag will fail to teleport if another actor is blocking the destination.

NOTELESTOMP

This actor cannot telefrag others under any circumstances, even if the AllowMonsterTelefrags map definition flag is set. This also affects actors spawned via the Icon of Sin Cubes.

STAYMORPHED

If morphed this actor cannot revert to its original form. Does not appear to work on playerclasses.

CANBLAST

Can be blasted by Hexen's Disc of Repulsion. For monsters this is implicit.

NOBLOCKMONST

Actor can walk through monster blocking lines.

ALLOWTHRUFLAGS

Only useful for actors used as puffs. Enables a variety of THRU flag behavior for actors used as puffs. Flags not listed do not require ALLOWTHRUFLAGS to work.

Bullets - Enables THRUACTORS and THRUSPECIES on puffs.

Rails - Enables THRUACTORS, THRUSPECIES and THRUGHOST on puffs.

THRUGHOST

Missiles and puffs pass through ghosts (set with the GHOST flag). See ALLOWTHRUFLAGS for details on puffs.

THRUACTORS

This actor passes through all other actors. See ALLOWTHRUFLAGS for details on puffs.

THRUSPECIES

This actor passes through other actors of the same species. See ALLOWTHRUFLAGS for details on puffs.

MTHRUSPECIES

A missile or puff with this flag makes the attack (the missile itself, or the puff's hitscan) passes through actors of the same species as the actor that shot it, if any.

A rail attack goes harmlessly through actors which have the same species as that of the attacker's, provided that the puff has this flag set. Additionally, A_BFGSpray has no effect on actors which have the same species as that of the associated flash, provided, again, that the flash also has this flag set. (The actors' species needs to match the shooter's, not the flash's)

SPECTRAL

A monster with this flag can only be hurt by a missile or puff (bullet attacks only; not rail attacks) that also has this flag set.

FRIGHTENED

Monster runs away from player (but still fights back).

FRIGHTENING

Monster runs away from its target if said target has this flag set, attacking less frequently.

NOTARGET

Actor cannot be targeted by other monsters. This does not prevent hostile monsters from being targeted by friendly monsters.

NEVERTARGET

Actor cannot be targeted at all. This prevents hostile monsters from being targeted by friendly monsters.

NOINFIGHTSPECIES

Actor does not infight with others belonging to the same species as it.

FORCEINFIGHTING

A monster ignores the map's setting of no infighting, and infights like normal.

NOINFIGHTING

The inverse of NOTARGET. This actor will never turn on another monster when provoked.

NOTIMEFREEZE

This actor is not affected by time freeze powerups or the "freeze" console cheat.

NOFEAR

This actor is not affected by frightener powerups or the "anubis" console cheat.

CANTSEEK

This actor cannot be tracked by seeker missiles. See also the DONTSEEKINVISIBLE flag.

SEEINVISIBLE

Monsters with this flag can see and aim at invisible opponents (players or other monsters) without their aim being degraded.

DONTTHRUST

Actor is excluded from damage and radius thrusting of all sorts by explosions or damage of any kind. It also never deals impact damage to other actors, nor does it damage itself from being too close to a wall.

ALLOWPAIN

Actors with this flag can enter pain states regardless of invulnerability or damage absorption, if the incoming damage is greater than zero.

USEKILLSCRIPTS

Actors with this flag will execute KILL-type ACS scripts upon death.

NOKILLSCRIPTS

Actors with this flag will never execute KILL-type ACS scripts upon death, even if ForceKillScripts is enabled in the GameInfo definition.

STOPRAILS

Piercing rail attacks stop at and do not go through an actor with this flag. The actor is still subject to damage from the attack, however.

MINVISIBLE (development version 4475240 only)

Monsters will treat actors with this flag as if they were invisible, regardless of their rendering properties.

MVISBLOCKED (development version 4475240 only)

Actors with this flag are entirely blocked from monster sight checks. This can be used to guarantee full invisibility, without any random chance of monsters being able to spot the actor.

Defenses

INVULNERABLE

Actor is invulnerable. It cannot be hurt in normal situations, will never enter its Pain state, and will not change targets unless damaged by FOILINVUL projectiles. Use ALLOWPAIN to let the actor feel pain while

invulnerable.

Note: an invulnerable actor can still be hurt by damage that exceeds a very high threshold set at 1000000 (one million). This ensures that certain "instakill" attacks such as the mdk console command or telefragging always work, even on invulnerable actors. Invulnerability also ignores damagefactors; the raw damage must be a million points or greater in order to foil it. NODAMAGE is the only flag to ignore these rules.

BUDDHA

Actor is given an indestructible hit point: no normal attack can remove that last hit point from the actor. They can still take damage and enter any pain state. However, if the actor is hit with an attack bearing the FOILBUDDHA flag or was subjected to telefragging or "instakill" attacks, the actor will still die as a result.

REFLECTIVE

Actor reflects missiles shot at it. By default projectiles are reflected at a random angle. This also causes the reflecting actor to be considered as the original shooter once reflected, and seeker missiles will seek after the original shooter.

SHIELDREFLECT

Used in conjunction with the REFLECTIVE flag. It changes the reflection behavior so that only projectiles hitting the monster from the front side are reflected. With this flag the reflection angle is always +/-45 degrees.

DEFLECT

Used in conjunction with the REFLECTIVE flag. It changes the reflection behavior so that the reflection angle is always +/-45 degrees.

MIRRORREFLECT

Used in conjunction with the REFLECTIVE flag. It changes the reflection behavior so the projectile is always turned 180 degrees around and the speed is inversed. Takes precedence over SHIELDREFLECT and DEFLECT.

AIMREFLECT

Used in conjunction with the REFLECTIVE flag. It changes the reflection behavior so the projectile is always turned and aimed at the original shooter. Takes precedence over SHIELDREFLECT, DEFLECT and MIRRORREFLECT flags. If there is no shooter to aim at, it will fall back on the flags it takes precedence over. I.e. if MIRRORREFLECT is specified, it will simply use that behavior instead.

THRUREFLECT

Used in conjunction with the REFLECTIVE flag. It changes the reflection behavior so the projectile passes through without changing angle or speed, but is still considered reflected. Takes precedence over all reflective modifying flags.

NORADIUSDMG

Actor cannot be hurt by radius (explosive) damage.

DONTBLAST

Cannot be blasted by Hexen's Disc of Repulsion.

SHADOW

Makes monsters trying to target this actor to be inaccurate. Unlike regular Doom this does not automatically imply fuzziness. For that you have to specify RenderStyle Fuzzy.

GHOST

Actor is a ghost. This does not imply translucency etc. Some of Heretic's projectiles and weapons cannot attack ghosts.

DONTMORPH

Cannot be morphed into a chicken, pig or any other morphs.

DONTSQUASH

This actor cannot be instantly killed by Heretic's powered up Mace. BOSS implies this flag.

NOTELEOTHER

Cannot be teleported by Hexen's banishment device.

HARMFRIENDS

Projectile attacks from a friendly monster with this flag will hurt other friendly monsters of the same species.

DOHARMSPECIES

Monsters with this flag will be able to receive damage from projectile attacks from other monsters sharing the same species. By default projectiles fired by monsters belonging to the same species can only deal damage and trigger infighting via A_Explode but they deal no impact damage; this flag overrides that behavior, forcing the monster to receive impact damage from projectiles as well.

The flag's name is a misnomer: it sounds like it's the opposite of DONTHARMSPECIES, but the effect is

entirely different. This flag only affects the damage received by the monster, not the damage it deals, as opposed to DONTTHARMSPECIES.

DONTTHARMCLASS

Prevents monsters with this flag from dealing explosion damage to other monsters of the same class. By default, projectiles fired by monsters of the same class don't deal impact damage but can deal damage and trigger infighting via A_Explode. This flag overrides that behavior, blocking explosion damage between the same class as well. (This also means that monsters with this flag can't blow themselves up with their own explosive projectiles.)

Note, if DONTTHARMSPECIES is used on the monster, adding this flag is pointless because it's implied.

This flag was previously named DONTTHURTSPECIES, which was a misleading name that could lead to believe its effect was that of DONTTHARMSPECIES.

DONTTHARMSPECIES

Works the same way as the DONTTHARMCLASS flag, except it checks for species, not class name. Prevents monsters with this flag from dealing explosion damage to other monsters that have the same species. By default, projectiles fired by monsters belonging to the same species don't deal impact damage but can deal damage and trigger infighting via A_Explode. This flag overrides that behavior, blocking explosion damage between monsters of the same species as well. (This also means that monsters with this flag can't blow themselves up with their own explosive projectiles.)

This flag implies DONTTHARMCLASS.

NODAMAGE

The actor reduces ALL damage to 0, but still reacts to getting hurt (allowing infighting).

NOTE: Unlike INVULNERABLE, this flag carries a higher degree of protection against all effects. They cannot be killed with MDK, the "kill monsters" command, Thing_Destroy, telefrag damage, and are completely impervious to the FOILINVUL flag. If the actor has both INVULNERABLE and NODAMAGE, then FOILINVUL actors can still be used to draw their attention.

DONTRIP

Ripping projectiles die upon hitting this actor as though they were non-ripping projectiles. This takes precedence over ripping levels.

NOTELEFRAG

Actor cannot be telefragged. This only implies the actual teleport effect, not by damaging functions. Actors simply cannot teleport into that position at all.

ALWAYSTELEFRAG

An actor with this flag will unconditionally be telefragged by others when in the way.

DONTDRAIN

Actor's health cannot be drained with weapons that drain health from the victim such as Heretic's gauntlets, or with weapons using the lifestea! capability of A_Saw and A_CustomPunch.

LAXTELEFRAGDMG

By default, when an actor receives telefrag-esque damage (million points) from a single attack, damagefactors are ignored. This flag forces the damage to be mitigated or amplified regardless, based on damagetype and factor. Invulnerable and buddha capabilities are still foiled nonetheless.

Appearance and sound

BRIGHT

All of the actor's frames are rendered at full-bright, as if every frame had the bright keyword present.

INVISIBLE

Actor is invisible. This disables the rendering of actor's sprites, similarly to RenderStyle None, but in addition makes it actually invisible to monsters.

NOBLOOD

Actor does not bleed when hurt. Additionally, an actor with this flag is immune to poison bolts. You can also specify a blood type (without this flag) using the BloodType actor property.

NOBLOODDECALS

Actor does not generate blood decals when shot (but still generates blood sprites). Not needed if NOBLOOD is already specified.

STEALTH

Actor is a stealth monster. Stealth monsters are monsters that are visible only when they are in pain or attack; other than that, they are invisible.

FLOORCLIP

Actor's lower part is clipped when standing in a sector with a liquid texture (defined in the TERRAIN lump).

SPAWNFLOAT

Actor is spawned at a random height in the sector.

SPAWNCEILING

Actor is spawned hanging from the ceiling as opposed to standing on the floor. Use in conjunction with NOGRAVITY.

FLOATBOB

Use float bobbing z movement like Heretic/Hexen's powerups. Generally used with NOGRAVITY.

NOICEDEATH

Monster cannot be frozen, used to prevent the generic ice death.

DONTGIB

Actor cannot be crushed to a pile of blood by crushing ceilings. Also, dropped items with this flag cannot be crushed into nothingness as they usually do.

DONTSPASH

Actor does not create any terrain splashes.

DONTOVERLAP

Two actors with this flag cannot occupy the same x/y-position. Note that this flag will short-circuit a large part of the collision detection code, so it is not recommended to be used unless unavoidable!

RANDOMIZE

Randomizes the duration for its first death and spawn frame of a projectile by removing up to 3 tics. Most of Doom's (but not Heretic's and Hexen's) projectiles use this.

FIXMAPTHINGPOS

Move thing out of walls. For torches and similar things that tend to be placed directly on a wall.

FULLVOLACTIVE

Plays its active sound at full volume.

FULLVOLDEATH

Plays its death sound at full volume. This only works for projectiles.

FULLVOLSEE

Plays its see sound at full volume.

NOWALLBOUNCESND

Does not play a sound when bouncing off a wall. Normally the SeeSound is played in this case.

VISIBILITYPULSE

The actor's translucency is pulsing between 25% and fully opaque.

ROCKETTRAIL

This object has a rocket particle trail. It will spawn a RocketSmokeTrail actor at its position every four tics.

GRENADETRAIL

This object has a grenade particle trail (from Skulltag's grenades). It will spawn a GrenadeSmokeTrail actor at its position every eight tics. This flag is ignored if the actor also has the ROCKETTRAIL flag, which gets priority.

NOBOUNCESOUND

Don't make a bouncing sound.

NOSKIN

Don't allow skins for this actor. Useful only for player classes.

DONTTRANSLATE

Prevents this actor from being translated when used as a blood class (using the BloodType property) or when spawned from another actor using A_SpawnItemEx.

On a player class, this prevents player translations from being applied.

Note: Changing this flag during play has no effect; this flag is only checked when the actor is spawned and is not referenced afterwards.

NOPAIN

The actor will never enter its pain state. Contrarily to merely giving a PainChance of 0, this overrides the FORCEPAIN flag on the inflictor. Note that it will still take damage, however.

FORCEYBILLBOARD OpenGL.png (OpenGL only: not supported by ZDoom)

Added for GZDoom, Skulltag and Zandronum compatibility. Forces the actor's sprite to "billboard" to the screen on the Y-axis when using the OpenGL renderer. Does nothing in ZDoom.

FORCEXYBILLBOARD OpenGL.png (OpenGL only: not supported by ZDoom)

Added for GZDoom, Skulltag and Zandronum compatibility. Forces the actor's sprite to "billboard" to the

screen on the X- and Y-axes when using the OpenGL renderer. Does nothing in ZDoom (Also ZDoom already "billboards" sprites on the X- and Y-axes).

Projectile

MISSILE

Actor is a projectile. Actors with this flag set will enter their death state when hitting a solid wall or an actor, and constantly move at their speed value (without the need of any actor functions). Actors with this flag will also be able to go through impassable linedefs, unless they're set to block projectiles. This flag is automatically unset once the projectile dies upon hitting an actor or an obstacle, thus checking for it is the best generic way to check if the projectile is still active.

→ Automatically given by the Projectile combo

RIPPER

For projectiles that can rip through monsters and players. Ripping projectiles constantly cause their damage amount when ripping through actors, so low damage values are recommended. Bleeding actors also spew blood when being ripped. They will never be reflectable by any actor unless it comes into contact with an actor containing DONTRIP.

NOBOSSRIP

Ripper projectiles with this flag will not rip through monsters with the +BOSS flag.

NODAMAGETHRUST

An actor with that flag does not thrust away its victims when it inflicts damage.

DONTREFLECT

Will not reflect off of actors with the REFLECTIVE flag set, instead exploding on impact. If the projectile also includes the ability to bounce off of actors with special conditions, it will do so without being considered reflected and maintain who shot it and who it is seeking if set to do so. THRUREFLECT takes priority over this.

NOSHIELDREFLECT

Will not reflect off of actors with the SHIELDREFLECT flag set, instead exploding on impact. If the projectile also includes the ability to bounce off of actors with special conditions, it will do so without being considered reflected and maintain who shot it and who it is seeking if set to do so. THRUREFLECT takes priority over this.

FLOORHUGGER

Defines a projectile that is moving along the floor. Floor hugging projectiles pass over all obstacles until they hit a solid wall (not a raised floor).

CEILINGHUGGER

Defines a projectile that is moving along the ceiling.

BLOODLESSIMPACT

No blood is spawned when this projectile hits something. This also prevents hitscan weapons from spawning blood.

BLOODSPATTER

Spawns blood splatter sprites when hitting a bleeding actor. This can also be used for hitscan weapons.

→ Automatically given by the Projectile combo when playing Heretic or Hexen

FOILINVUL

Can hurt invulnerable monsters (but not players).

FOILBUDDHA

Ignores the BUDDHA flag behavior and kills them if applicable. Damage factors and types still take priority.

Players are unaffected by this flag.

SEEKERMISSILE

Actor is a homing missile. This is only used as a hint to the game. For a homing missile to be effective it also has to use one of the seeker missile code pointers in its moving states.

SCREENSEEKER

A seeker missile with this flag can only acquire a new target (in its tracer field) if the latter is in the field of vision of the missile's shooter (more exactly, within an about 84° cone from the shooter).

SKYEXPLODE

Projectile explodes and puff spawns when hitting a sky instead of vanishing.

A puff with this flag spawns on linedefs which have the Line_Horizon special set.

NOEXPLODEFLOOR

When hitting the floor the projectile just stops instead of exploding.

STRIFEDAMAGE

Strife uses a different damage calculation for its projectiles which results in lower damage; the damage is multiplied by a random value between 1 and 4, inclusive.

EXTREMEDEATH

This projectile or puff always gibs its victim.

NOEXTREMEDEATH

This projectile or puff never gibs its victim.

DEHEXPLOSION

Draw the missile's Death state translucent or additive depending on the addrocketexplosion console variable and the 'Rocket Explosion Style' and 'Rocket Explosion Alpha' settings in DeHackEd.

PIERCEARMOR

Armor does not alter the damage done by a projectile with this flag set.

FORCERADIUSDMG

All actors will take radius damage from this projectile, even boss enemies that are normally immune to radius damage.

FORCEZERORADIUSDMG

Forces splash damage of zero to pass through to the code responsible for damaging actors. The purpose of this is to give the projectile the chance to execute its special damage code.

SPAWNSOUNDSOURCE

This projectile will play its seesound from the originating actor.

PAINLESS

Actors will never enter their Pain state if hit by a projectile with this flag set.

FORCEPAIN

Actors will always enter the Pain state appropriate to the type of damage received if hit by a projectile with this flag set, even if the projectile does zero damage. The PAINLESS flag has precedence (a projectile with both PAINLESS and FORCEPAIN will not cause actors to enter their Pain state), and pain cannot be forced on actors with the NOPAIN flag set.

CAUSEPAIN

Actors will always be subject to their pain chance, even if said actors are invulnerable or the amount of damage inflicted is zero.

DONTSEEKINVISIBLE

Seeker missiles with this flag will not home in on invisible actors. See also the CANTSEEK flag.

STEPMISSILE

Missiles with this flag can climb up steps.

ADDITIVEPOISONDAMAGE

Several hits from missiles or puffs with the PoisonDamage property and this flag cause the poison damage suffered by the victim to increase.

ADDITIVEPOISONDURATION

Several hits from missiles or puffs with the PoisonDamage property and this flag cause the poisoning suffered by the victim to be prolonged.

POISONALWAYS

(Note: this flag does not apply to the Hexen style of poison which affects players only.)

Poison is inflicted even if the victim is invulnerable.

HITTARGET

Projectiles that die from hitting an actor will set that actor as its target pointer. Bouncing missiles can make use of this to immediately change their pointers upon bouncing. Bullets and rail attacks can benefit from this flag as well.

HITMASTER

Projectiles that die from hitting an actor will set that actor as its master pointer. Bouncing missiles can make use of this to immediately change their pointers upon bouncing. Bullets and rail attacks can benefit from this flag as well.

HITTRACER

Projectiles that die from hitting an actor will set that actor as its tracer pointer. Bouncing missiles can make use of this to immediately change their pointers upon bouncing. Bullets and rail attacks can benefit from this flag as well.

HITOWNER

(Note: despite having similar naming, this flag does not share the same functionality as the above three flags.)

Normally, projectiles go harmlessly through their shooters. This flag disables that behavior, and instead allows projectiles to collide with their shooters. This means, a projectile could potentially explode immediately upon being fired, possibly damaging its shooter, so care should be exercised when using the flag. A useful application of this flag is with bouncing projectiles, so that they collide with their shooters when bouncing back instead of going through them.

Bouncing

See also the bounce properties.

BOUNCEONWALLS

Missile bounces off of walls. Implied by Doom and Hexen bounce types.

BOUNCEONFLOORS

Missile bounces off of floors. Implied by all three bounce types.

BOUNCEONCEILINGS

Missile bounces off of ceilings. Implied by all three bounce types.

ALLOWBOUNCEONACTORS

Missile bounces off of non-bleeding actors. Implied by Doom and Hexen bounce types. Note that this needs to be set for BOUNCEONACTORS below to have any effect.

BOUNCEAUTOOFF

Missile stops bouncing if the velocity is too low after a bounce. Always triggered by a bounce on the ceiling, since moving downwards will always count as not moving upwards enough. Implied by the Doom bounce type.

BOUNCEAUTOOFFFLOORONLY

Missile stops bouncing if the velocity is too low after a bounce on the floor.

BOUNCELIKEHERETIC

Missile only bounces once, and never on walls, like Heretic's Firemace projectiles. Implied by Heretic bounce type.

BOUNCEONACTORS

Missile bounces off of all actors, even if they bleed and aren't reflective.

BOUNCEONUNRIPPABLES

Ripping missile bounces off of shootable actors with the DONTRIP flag set.

NOWALLBOUNCESND

Missile doesn't play a sound when hitting a wall. The Heresiarch's bouncing flaming skulls use this.

NOBOUNCESOUND

Missile doesn't make any sound at all when bouncing.

EXPLODEONWATER

Missile explodes when landing on a liquid surface. Usually they vanish in this case.

CANBOUNCEWATER

Missile bounces off of liquids normally. Usually they vanish in this case.

MBFBOUNCER

Missile is considered an MBF bouncer. They behave differently from normal bouncers in many situations: They are considered to be missiles for several tests, even if they do not have the MISSILE flag. For example, a monster with that flag is able to cross impassable linedefs that do not block missiles. They do not lose their bouncing properties when slowing down and coming to a rest.

USEBOUNCESTATE

Missile enters the most appropriate Bounce state available upon bouncing. Possible bounce states include Bounce, Bounce.Floor, Bounce.Ceiling, Bounce.Wall, Bounce.Actor, and Bounce.Actor.Creature. The Bounce.Actor.Creature state is used for bouncing over a shootable actor without the NOBLOOD flag.

DONTBOUNCEONSHOOTABLES

Missile does not bounce off shootable actors, and instead, it explodes.

DONTBOUNCEONSKY

Missile does not bounce off skies, and instead, it disappears.

Miscellaneous

ICESHATTER

An actor with this flag, when causing damage, can shatter ice corpses regardless of damage type. Note that initially the damage which kills and freezes the actor at first will not shatter it, but any damage done afterwards will.

DROPPED

Actor always acts as if it was dropped. Dropped items have two properties: They never respawn, and they will be crushed by doors and moving sectors, unless they have the DONTGIB flag.

ISMONSTER

Actor is classed as a monster.

→ Automatically given by the Monster combo

CORPSE

Actor is a corpse. For normal actors there is no need to set this but in combination with the Crash state it might be useful.

COUNTITEM

Counts toward item percentage.

COUNTKILL

Counts toward kill percentage.

→ Automatically given by the Monster combo

COUNTSECRET

Counts toward secret percentage.

NOTDMATCH

Actor is not spawned in deathmatch games.

NONSHOOTABLE

Actor cannot be hit (projectiles pass through).

DROPOFF

Actor can walk over ledges/taller steps. This would allow a monster to travel through a map with impunity.

→ Automatically given by the Projectile combo

PUFFONACTORS

Used on puff objects only. A puff with this flag is spawned even if the actor being hit bleeds. Normally in such a case only blood is spawned but no puff.

ALLOWPARTICLES

Used on puff and blood objects only. This flag allows the sprite to be replaced by particles.

ALWAYSPUFF

Used on puff objects only. Makes the puff appear even when nothing was hit. Puffs spawn on the floor/ceiling via rails if this flag is provided.

PUFFGETSOWNER

Used on puff objects. If set, the puff's target is set to the actor who fired the shot once it is spawned.

Usually used in conjunction with A_GiveToTarget to achieve certain special effect. The same can be set with BFG Spray actors.

It can also be used on Blood: in that case the actor that is bleeding will be considered the Blood actor's target (by default player is considered any non-friendly actor's target). If A_FaceTarget function is added, the Blood actor will face the bleeding actor.

FORCEDECAL

A puff with this flag has its decal used instead of the one defined by the weapon.

NODECAL

A hitscan attack using a puff with this flag does not generate decals.

SYNCHRONIZED

Normally all actors spawned at level start are slightly randomized to avoid having all of them have the exact same appearance at the same time. This flag disables that randomization.

ALWAYSFAST

The monster with this flag does not take a random number of steps before attempting an attack. In addition, the reaction time is reduced and actor states marked as FAST will have their durations halved.

NEVERFAST

The monster with this flag will always take a random number of steps before attempting to attack. Opposite of ALWAYSFAST.

OLDRADIUSDMG

Use old radius damage code (for barrels and boss brain). Actors with this flag do not take z-height into account when calculating radius damage.

USESPECIAL

The actor's special will be triggered when the player presses use while facing this actor. The player will be considered the activator of the special.

Note: This default behavior can be modified with the Activation property.

BUMPSPECIAL

The actor's special will be triggered when the player touches this actor. The player will be considered the activator of the special.

Note: This default behavior can be modified with the Activation property.

BOSSDEATH

Makes A_FreezeDeathChunks and A_Burst call A_BossDeath.

NOINTERACTION

The actor is purely decorative. It will not interact with any other actor and will ignore all game physics (including solid walls and floors!), moving only according to its own momentum. (i.e. The actor will not be affected by gravity, but its movement can still be directly changed using ThrustThing.) All AI routines will ignore this actor's existence, speeding up processing time.

Note: Setting this flag implies removing the actor from the blockmap. If an actor had this flag set needs to be restored to being a solid object, A_ChangeLinkFlags(0) must be called manually to reset its blockmap data. Keep in mind NOBLOCKMAP will only apply on the next actor tick when used, meaning the actor is still subject to repositioning (i.e. if a puff without the flag hits a ledge, the engine may move it above the ledge). Including NOBLOCKMAP as part of the actor's definition will prevent this behavior, since the engine won't link it to begin with and save processing time.

NOTAUTOAIMED

The actor is ignored by autoaim. Can be overridden by setting cl_doautoaim to true.

NOMENU

A player class with this flag set is excluded from being listed in the class selection menu.

PICKUP

Tells the engine whether or not this class can pick up items. Mostly useful for preventing morphed players from using items.

TOUCHY

The actor dies at the slightest touch. This includes contact with another actor, sudden contact with the ceiling (closing door, raising elevator, floor or ceiling crusher, and so on), falling, and being blasted by an effect such as the disc of repulsion.

VULNERABLE

An actor with this flag can be affected by area-of-effect damage. Only matters for actors without the SHOOTABLE flag.

NOTONAUTOMAP

An actor with this flag will not be shown on the automap while the scanner powerup is in effect. The cheat code can still reveal the actors however.

WEAPONSPAWN

Actors with this flag are affected by cooperative weapon spawning filter DMFlag. Actors inheriting from Weapon will have this set.

Limited use

GETOWNER

Only useful with FastProjectiles. By default, the actors spawned by the FastProjectiles will set the projectile as the target pointer. Use this flag to get the original shooter as the owner for proper damage methods.

SEESDAGGERS

Strife specific flag. Any actor with this flag will react to a player using Strife's dagger which normally doesn't alert monsters. You can clear it from Strife's Acolytes though to prevent them from responding to dagger attacks.

INCOMBAT

Used for Strife's dialog system. An actor that has this flag set will not respond to the player's attempts to converse, even if it has a ConversationID set. Not particularly useful to have an actor spawn with, but can be set later if the actor should become hostile to the player.

NOCLIP

Actor is totally excluded from collision detection and can walk through walls etc.

NOSECTOR

Object is not linked into the sector. This makes it invisible and excludes it from certain physics checks. It is recommended not to use this flag due to its side effects. There are better ways to make an actor invisible, e.g. the INVISIBLE flag.

ICECORPSE

Actor is a frozen corpse.

JUSTHIT

Try to attack right back (used in monster AI, probably not particularly useful in actor definitions).

JUSTATTACKED

Take at least one step before attacking (also not particularly useful in an actor definition).

TELEPORT

Used when teleporting an actor. When something is teleported, this flag is set, it gets moved to its new position, it's checked for collision, then the flag is cleared (again, not useful for actor definition).

BLASTED

Actor takes temporary damage from impact. Related to the Disc of Repulsion. Not useful in actor definitions.

EXPLOCOUNT

A missile with this flag that should normally explode only does so if its increased special2 field is greater than its special1 field.

SKULLFLY

Actor is a charging monster. Like a MISSILE, its movement will not be damped, and it will cause damage on impact. In addition, the actor will not go into pain states, will also be blocked by items, and on impact SKULLFLY will be cleared and the actor will return to its initial state. It can be dynamically set like any ordinary flag, but the actor needs to have a velocity for it to do anything. A_SkullAttack and A_MinotaurCharge set this flag along with thrusting the actor.

RETARGETAFTERSLAM

A charging actor with this flag set enters the Idle state instead of the See state upon slamming into other actors. This flag is only used with the lost soul to restore its original behavior of reacquiring its target upon slamming into other actors.

ONLYSLAMSOLID (New from 4.10.0)

A charging actor with this flag will only collide with solid actors, instead of also colliding with non-solid actors like items. Solid actors are actors that have SOLID or SHOOTABLE.

SPECIALFIREDAMAGE

A projectile with this flag that causes fire damage will not inflict a fire death when killing an actor.

SPECIALFLOORCLIP

An actor with this flag is considered to use the floor clipping feature to achieve special effects, such as the buried reiver raising from the ground; so its floorclip is not adjusted by the engine depending on which terrain type it is standing.

SUMMONEDMONSTER

This flag's use is limited to the Minotaur actor class and derivatives; it basically alters the outcome of the functions exclusive to that class such as A_MinotaurDecide, resulting in minor behavioral changes. The only actor class that makes use of this flag is MinotaurFriend.

SPECIAL

Tells the engine that this actor can be picked up. The base class SpectralMonster has a special use for this flag that allows actors inheriting from it to hurt the player when they come in contact with said actors. Unsetting this on inventory-derived classes does nothing, as the engine will automatically assign this flag for all defined inventory.

Boss event triggers

E1M8BOSS

When all actors with this flag in E1M8 are killed, the death sequence that lowers all floors with a tag of 666 will trigger.

E2M8BOSS

When all actors with this flag in E2M8 are killed, the map will automatically end.

E3M8BOSS

When all actors with this flag in E3M8 are killed, the map will automatically end like in E2M8.

E4M6BOSS

When all actors with this flag in E4M6 are killed, all sectors with a tag of 666 will quickly open.

E4M8BOSS

When all actors with this flag in E4M8 are killed, all the floors with a tag of 666 will lower to the closest floor.

MAP07BOSS1

When all actors with this flag are killed on MAP07, all floors with a tag of 666 will lower to the closest floor.

MAP07BOSS2

When all actors with this flag are killed on MAP07, all floors with a tag of 667 will raise by the height of the lowest texture.

Internal flags

These flags are used internally by GZDoom. But have also been exposed to ZScript to be usable for scripts too. These flags cannot be added directly to an actors' definition.

INCHASE

Used by A_Chase and A_Look/A_LookEx to prevent infinite recursion if the actor is already running the A_Chase function.

UNMORPHED

Marks the player or monster as being the original, unmorphed version of another player/monster.

FLY

Marks players as currently being able to fly, for as long as they have the flight powerup.

ONMOBJ

The actor is currently standing on top of another actor.

ARGSDEFINED

If the actor has args defined in its' DECORATE/ZScript definition, then this flag makes GZDoom ignore the map defined args for the actor.

NOSIGHTCHECK

Used by Thing_Hate types 2, 4, and 6, to make monsters attack other actors they've been set to hate, even if they currently have no line of sight to them.

CRASHED

The actor has entered its' Crash state.

WARNBOT

Used to make projectiles warn bots.

HUNTPLAYERS

Used by Thing_Hate types 2 and 4, to make monsters chase other monsters with a certain TID, but also still be able to chase players.

NOHATEPLAYERS

Used by Thing_Hate types 5 and 6, to make monsters only chase other monsters, and ignore players.

SCROLLMOVE

Marks the actor as being thrust by a scrolling sector.

VFRICITION

Used exclusively by A_PainAttack to make any monster with the FLOAT flag that called the function be thrust downwards once killed.

BOSSPAWNED

Used by the boss cubes fired by the Icon of Sin, to mark the monsters it spawns as having been spawned by the IoS.

AVOIDINGDROPOFF

Used to make monsters avoid dropoffs.

CHASEGOAL

Set by Thing_SetGoal if the "Don't Chase Target" parameter is set to true. To make the monster only follow its' patrol route but still be able to attack its' enemies.

INCONVERSATION

Is turned on while the actor is engaging in conversation with the player.

ARMED

Used by MBF to mark actors with the TOUCHY flag that are alive and have a See state, as being armed mines.

FALLING

Does nothing.

LINEDONE

Used by the deprecated A_LineEffect function to make the actor not be able to execute another special. After it executes a special that is not repeating.

SHATTERING

Marks an actor as needing to forcibly shatter eventually after becoming an ice corpse.

KILLED

Turned on when an actor dies, to mark it as having been killed. This does not necessarily mean that the actor is a corpse.

BOSSCUBE

Used to mark boss cubes spawned by A_BrainSpit as such. To prevent the cubes from accelerating while they are frozen in place.

INTRYMOVE

Turned on when the actor is running P_TryMove().

HANDLENODELAY

This flag is enabled right at the start of the PostBeginPlay() virtual of all actors, to make them obey the NoDelay state keyword.

FLYCHEAT

This flag along with the FLY flag is turned on if the player starts flying using the fly cheat.

RESPAWNINVUL

Enables the invulnerability visual effect respawned players get in Deathmatch when sv_respawnprotect is on.

Deprecated flags

These flags should no longer be used. They are documented here for reference purposes.

LOWGRAVITY (deprecated)

Actor is subject to low gravity.

Superseded by the Gravity actor property. Use Gravity 0.125 instead.

QUARTERGRAVITY (deprecated)

Actor is subject to 1/4th of normal gravity.

This flag was only added for compatibility with certain Skulltag definitions. Use Gravity 0.25 instead.

LONGMELEERANGE (deprecated)

When closer than 196 map units to its target, this monster does not start its missile attacks.

Superseded by the MeleeThreshold actor property. Use MeleeThreshold 196 instead.

SHORTMISSILERANGE (deprecated)

Has a limited missile attack range of 896 map units.

Superseded by the MaxTargetRange actor property. Use MaxTargetRange 896 instead.

HIGHERMPROB (deprecated)

Has a higher chance of performing a ranged attack.

Superseded by the MinMissileChance actor property. Use MinMissileChance 160 instead.

FIRERESIST (deprecated)

Actor takes one half of the normal damage from fire.

Superseded by the DamageFactor actor property. Use DamageFactor "Fire", 0.5 instead.

DONTHURTSPECIES (deprecated)

This is an alternate name for DONTARMCLASS. This name has been deprecated because it is misleading, as its effect is unrelated to the actual definition of Species in the ZDoom engine.

FIREDDAMAGE (deprecated)

Actor inflicts fire damage.

Superseded by the DamageType actor property. Use DamageType "Fire" instead.

ICEDAMAGE (deprecated)

Actor inflicts ice damage.

Superseded by the DamageType actor property. Use DamageType "Ice" instead.

HERETICBOUNCE (deprecated)

Heretic-style bouncing (objects only bounce off planes).

Use BounceType "Heretic" or the following flags instead: BOUNCEONFLOORS, BOUNCELIKEHERETIC.

HEXENBOUNCE (deprecated)

Hexen-style bouncing (objects bounce off planes and walls).

Use BounceType "Hexen" or the following flags instead: BOUNCEONWALLS, BOUNCEONFLOORS, ALLOWBOUNCEONACTORS.

DOOMBOUNCE (deprecated)

ZDoom-style bouncing (like Hexen but stops when losing a certain amount of momentum).

Use BounceType "Doom" or the following flags instead: BOUNCEONWALLS, BOUNCEONFLOORS, ALLOWBOUNCEONACTORS, BOUNCEAUTOOFF.

FASTER (deprecated)

The monster with this flag had the duration of the states in its See state sequence halved in nightmare skill or with fast monsters on. This flag has no effect anymore and the Fast state keyword should be used instead.

FASTMELEE (deprecated)

The duration of the states in the the monster's Melee state sequence was halved in nightmare mode or with fast monsters on. This flag has no effect anymore and the Fast state keyword keyword should be used instead.

Additional flags

Some subclasses of Actor define their own additional flags. They are listed here:

Inventory

INVENTORY.QUIET

When this item is picked up, its pickup sound is not played. Additionally, neither its pickup message nor the pickup palette flash are displayed.

INVENTORY.AUTOACTIVATE

This item activates automatically when being picked up.

INVENTORY.UNDROPPABLE

This item cannot be dropped once it has been picked up. Note that this also prevents the actor from being removed when ClearInventory or ClearActorInventory is called.

INVENTORY.UNCLEARABLE

This item cannot be removed by ClearInventory or ClearActorInventory, but can be dropped/tossed.

INVENTORY.INVBAR

This item is placed into the visible inventory when picked up.

INVENTORY.HUBPOWER

This item is kept when travelling between levels of the same hub.

INVENTORY.PERSISTENTPOWER

This item is kept when travelling between levels, even outside of a hub.

INVENTORY.INTERHUBSTRIP (deprecated)

This item is taken away when traveling between hubs or single levels. Replaced by the more flexible

Inventory.InterHubAmount property.

INVENTORY.PICKUPFLASH (deprecated)

When being picked up a PickupFlash actor is spawned. This is the blue effect you can observe on Heretic and Hexen. This flag has been deprecated and it is recommended that you use the Inventory.PickupFlash property instead.

INVENTORY.ALWAYSPICKUP

This item is always picked up no matter whether the player can use it or not. This only applies to items that activate automatically.

INVENTORY.FANCYPICKUPSOUND (deprecated)

The pickup sound is supposed to be played in surround mode. This flag does not actually have any effect.

INVENTORY.NOATTENPICKUPSOUND

The pickup sound is played with no attenuation, which means it is played at full volume and is heard clearly regardless of distance.

INVENTORY.BIGPOWERUP

Marks this item as a 'powerful' item which is controlled by the 'mega powerups respawn' dmflag option.

INVENTORY.NEVERRESPAWN

An item with this flag will never respawn in any circumstance.

INVENTORY.KEEPDEPLETED

This item will remain in the player's inventory bar even after the last one is used. If the item also has an inventory icon, it will be drawn darkened when the quantity is 0.

INVENTORY.IGNORESKILL

Normally, the amount of ammo picked up from an inventory item is doubled on the easiest and hardest skill levels. If this flag is set, the item will ignore the skill setting and only give the specified ammo amount. This flag is typically used on a magazine for reloading weapons so that only one round is inserted into the magazine per round of ammunition depleted from the main ammo pool.

INVENTORY.ADDITIVETIME

If set, when a player picks up a second powerup of this type before the first has worn off, the new powerup's duration will be added to the old, rather than overwriting it. For example, if a powerup has a duration of 60 seconds, and a player who currently has the powerup with 21 seconds left picks up a second one, normally the new powerup will override the old and the duration will be reset to 60 seconds. With this flag set, the duration would be extended to 81 seconds instead. This flag can either be set directly on a powerup or its giver.

INVENTORY.UNTOSSABLE

If set, the item cannot be tossed with the drop console command.

INVENTORY.RESTRICTABSOLUTELY

If set, the Inventory.ForbiddenTo and Inventory.RestrictedTo will prevent player classes not allowed to pick up a weapon from attempting to pick it up for ammo.

INVENTORY.NOSCREENFLASH

Upon picking up an item with this flag, the associated pickup palette flash will not be displayed.

INVENTORY.TOSSED

An item with this flag is treated as being dropped.

INVENTORY.ALWAYSRESPAWN

An item with this flag will always respawn, regardless of the dmflag option. Note, this flag is different from the regular ALWAYSRESPAWN flag (without prefix): the latter one is for monsters, while this one is for items. In ZScript, where flag prefixes are not optional, you have to use +INVENTORY.ALWAYSRESPAWN, not +ALWAYSRESPAWN to make it work for Inventory items. Also, if dealing with a non-Inventory pointer, you will need to cast to Inventory first in order to access this flag.

INVENTORY.TRANSFER

An item with this flag will transfer all items in its inventory to the acquiring actor's own.

INVENTORY.NOTELEPORTFREEZE

Normally, a teleporting player loses the ability to move for about half a second before regaining it; an active power with this flag overrides this behavior. This flag can either be set directly on a powerup or its giver.

INVENTORY.NOSCREENBLINK

Powerups will never show the fade blinking every 1/4 of a second, which serves as an expiration warning. Useful when using scripting to allow special effects to happen without the blinking causing visual interference.

INVENTORY.ISHEALTH

An item with this flag set does not spawn if the sv_nohealth console variable is true.

INVENTORY.ISARMOR

An item with this flag set does not spawn if the sv_noarmor console variable is true.

Weapons

WEAPON.NOAUTOFIRE

Does not fire when selected automatically and the fire button is still pressed. Used to prevent dangerous weapons from firing accidentally.

WEAPON.READYSDHALF

The ready sound is played only with 50% probability.

WEAPON.DONTBOB

The weapon sprite does not bob.

WEAPON.AXEBLOOD

This weapon spawns the special AxeBlood type when hitting something that bleeds. This only has an effect for hitscan and melee weapons.

WEAPON.NOALERT

Does not alert nearby monsters when being fired.

WEAPON.AMMO_OPTIONAL

Tells the engine that this weapon doesn't require ammo to work.

WEAPON.ALT_AMMO_OPTIONAL

The same for the alternate attack.

WEAPON.AMMO_CHECKBOTH

The weapon can be selected if either the primary or the alternate fire have enough ammo. Both weapon properties Weapon.AmmoUse1 and Weapon.AmmoUse2 must have a positive value for this flag to work correctly.

WEAPON.PRIMARY_USES_BOTH

The primary attack uses both ammo types.

WEAPON.ALT_USES_BOTH

Like WEAPON.PRIMARY_USES_BOTH, makes the AltFire (secondary) attack use both ammo types.

WEAPON.WIMPY_WEAPON

A small weapon with limited capabilities. If ammo for something better gets picked up the game will automatically switch weapons unless the weapon to switch to has the NOAUTOSWITCHTO flag set.

WEAPON.POWERED_UP

This is a powered up weapon. Powered up weapons cannot exist by themselves. They always are linked to a normal weapon via the weapon.sisterweapon property and are only activated by using Heretic's Tome of Power or another artifact based on PowerWeaponLevel2.

WEAPON.STAFF2_KICKBACK

Uses the special kickback formula of Heretic's powered up staff. Unlike normal kickback this is a fixed thrust value.

WEAPON.EXPLOSIVE

Signifies to bots that the weapon fires explosive projectiles.

WEAPON.MELEEWEPON

Signifies to bots, and to monsters with the AVOIDMELEE flag, that the weapon is a melee weapon.

WEAPON.BFG

Signifies to bots that the weapon causes a lot of damage.

WEAPON.CHEATNOTWEAPON

Weapon is not given by the 'give weapons' cheat.

WEAPON.NO_AUTO_SWITCH

When the player picks this weapon up, they will never automatically switch to it.

WEAPON.NOAUTOSWITCHTO

A weapon with this flag set cannot be switched to automatically upon obtaining ammo for it. Typically, if the player had run out of ammo for a weapon, and the weapon that was automatically switched to as a result is a weak weapon, obtaining ammo for the other weapon automatically switches away from the weak weapon and back to it, provided the player had not manually switched weapons in the interim, and the other weapon has a higher selection priority than the weak one.

WEAPON.NOAUTOAIM

A weapon with that flag will not adjust the aim of an attack, no matter the player's autoaim settings. This is intended to be used only for projectiles affected by gravity, such as grenades, where the player will usually want to aim higher than in a straight line, though it does also affect hitscan and railgun attacks.

WEAPON.NODEATHDESELECT

The weapon will not jump to the deselect state when the player dies.

WEAPON.NODEATHINPUT

The weapon cannot act on any input if the player is dead.

PowerSpeed

POWERSPEED.NOTRAIL

If set, it disables the speed trail effect that is created while the player is moving.

Players

PLAYERPAWN.NOTHRUSTWHENINVUL

Player is not thrust by attacks when invulnerable.

PLAYERPAWN.CANSUPERMORPH

If this flag is given to a player morph, the morphed player receives a tome of power if they are attempted to be morphed again. This is used by the chicken morph to reproduce the original behavior of Heretic and its "super chickens".

PLAYERPAWN.CROUCHABLEMORPH

Enables crouching for morphed player classes.

PLAYERPAWN.WEAPONLEVEL2ENDED

Signals that an enhanced weapon power has expired, so the code responsible switches the player's powered-up weapon back to its normal form.

This flag was introduced as part of a solution to make switching from the powered-up weapon to its normal form seamless upon the power's expiration, so it may not be used in user code.

Actor pointer

Actor pointers are used to keep track of an actor's relationships with others, such as which one spawned them, or which ones they spawned.

There are three actor pointers that are generally used:

target
tracer
master

Some additional pointers also exist, such as lastenemy, but are not normally exposed to modders.

Depending on the type of actor, different pointers are used.

Projectiles

target: A projectile's target is not the actor that it will seek after. Instead, this is the actor who fired the projectile (though a little counter-intuitive at that).

tracer: This is used only if the projectile has the +SEEKERMISILE flag. It will lock onto an enemy and seek towards whoever is in this field.

master: Unused.

Monsters

master: This refers to the monster that spawns them using A_SpawnItemEx with the SXF_SETMASTER flag, or is transferred over with the SXF_TRANSFERPOINTERS flag.

target: This refers to whom the monster is chasing and will attack when called upon with A_Chase.

tracer: Unused.

Special cases

Many special actors have different handling for these pointers which do not fit in the generic categories above. They include:

Anything that explodes will use its target as the source responsible for damage inflicted by the explosion.

SpawnShot (A_SpawnFly and A_SpawnSound will move it towards its target, not its tracer as with other projectiles).

Archvile (A_VileTarget assigns its tracer to the spawned ArchvileFire, and A_VileAttack detonates the fire)

MinotaurFriend uses the tracer pointer to keep track of the player who summoned it.

HolyTail (A_CHolyTail uses the tracer pointer to manage its trail).

Lightning and derived classes use the tracer, target and lastenemy pointers to manage cohesion between the various components of the lightning column.

Dragon (its dedicated movement functions use the tracer fields as path nodes for its navigation)

TeleportFog uses the actor who teleported as the target.

This list is not exhaustive.

DECORATE & ACS

Several DECORATE and ACS functions support custom retrieval and assignment of pointer values. The data location or method of retrieval is specified using named pointer selectors.

All pointers are automatically supported by all implementing functions, unless the function documentation specifies otherwise.

Selectors

The following set of named values all indicate ways of retrieving an actor pointer from a source actor or a static context. The values are divided into categories, and the first selector applicable to the source actor is used. Selectors from different categories may be combined using BITWISE OR. (SELECTOR_A | SELECTOR_B)

Selector category 1: Player-only selectors

Players will use a pointer in this category if one is specified. Otherwise, a pointer from another applicable category will be used.

AAPTR_PLAYER_GETTARGET: Get the actor in the player's line of sight. Most target-specific functions use this approach to determining the player's target. This only works if the actor has the SHOOTABLE and SOLID flags, and also lacks the NOBLOCKMAP flag, much like **A_JumpIfTargetInLOS**.

AAPTR_PLAYER_GETCONVERSATION: Get the actor currently talking to the player. Best used from a Strife dialogue that gives a custom inventory item, or starts a script with **ACS_ExecuteWithResult** (as it processes immediately).

Selector category 2: Generic context selectors

Any actor (non-null) will use a pointer in this category if one is specified. Otherwise, a pointer from another applicable category will be used.

AAPTR_MASTER: Access the actor's MASTER pointer. (Players normally do not have masters.)

AAPTR_TARGET: Access the actor's TARGET pointer. (Players normally do not use this.)

AAPTR_TRACER: Access the actor's TRACER pointer.

AAPTR_FRIENDPLAYER: Access the actor's FRIENDPLAYER pointer.

AAPTR_GET_LINETARGET: Get the actor in the line of sight. This is similar to **AAPTR_PLAYER_GETTARGET** above, except it is used for non-player actors.

Note that **AAPTR_GET_LINETARGET** is identified as **AAPTR_LINETARGET** in **DECORATE**.

Note on retrieving TARGET information: Most functions use a special approach to find the target of a player; checking what they are aiming/looking at. This corresponds to **AAPTR_PLAYER_GETTARGET**. To make a single function that conforms to this standard, use the selector combination **AAPTR_TARGET|AAPTR_PLAYER_GETTARGET**. The most applicable method will be used (**AAPTR_PLAYER_GETTARGET** for any player).

Selector category 3: Static context selectors

Any specified pointer in this category will be used.

AAPTR_NULL: Return NULL.

AAPTR_PLAYER# (where # is a number in the range 1 - 8):

A static pointer to the player of that number. NULL if the player does not exist.

Scripting tip: **AAPTR_PLAYER1** points to player 1. **AAPTR_PLAYER1<<X** (shift bits X up) points to player (1 + X). This fact can be applied in ACS loops if you need to reference each active player in sequence.

Selector category 4: Default

This selection is always implied, and applies if no other selection was made. To fully disable this, specify one static selection (as they always apply when specified), such as **AAPTR_NULL**.

AAPTR_DEFAULT: Returns the source actor itself (null if there is no source actor).

Assignment

The following selectors expose fields for manipulation: **AAPTR_MASTER**, **AAPTR_TARGET** and

AAPTR_TRACER.

Assignment operations will often, but not necessarily, prevent some assignments from occurring. Examples of such events are:

An actor pointing to itself

An infinite chain of references (two actors referencing eachother as master or target)

Prevention may involve cancelling the operation, or setting the pointer to NULL. Details on this should be included in the documentation of the individual function.

Revision information - assigning to pointers: Significant changes to this functionality are unlikely. Functions that support a set of selectors different from AAPTR_MASTER, AAPTR_TARGET, AAPTR_TRACER should list the supported features, along with any needed revision information.

SetActivator

int SetActivator (int tid[, pointer_selector])

Usage

This changes the activator of the script to the first actor found with the specified tid.

Parameters

tid: TID of the new activator.

pointer_selector: The pointer of the TID to set as the new activator.

Pointer Values

Priority 1, player-only selectors (used if specified with a player origin):

AAPTR_PLAYER_GETTARGET: Gets the actor under the crosshair.

AAPTR_PLAYER_GETCONVERSATION: Gets the actor the player is speaking to.

Priority 2, any-actor selectors (used if specified with any non-null origin):

AAPTR_TARGET

AAPTR_MASTER

AAPTR_TRACER

AAPTR_FRIENDPLAYER

Priority 3, any static selectors (used if specified)

AAPTR_PLAYER1, AAPTR_PLAYER2, AAPTR_PLAYER3, AAPTR_PLAYER4, AAPTR_PLAYER5,

AAPTR_PLAYER6, AAPTR_PLAYER7, AAPTR_PLAYER8

AAPTR_NULL

Minimum-priority

AAPTR_DEFAULT: Returns the origin, be it null or an actor

Selectors from different levels are combined using bitwise OR. Only one selector can be specified for each priority level

Return value

1 (TRUE) if the activator exists. If there were no actors with the supplied tid, or the pointer is NULL, this function returns 0 (FALSE) and the activator is set to the world.

Examples

Sets player 1 as the activator, regardless of who activated the script, and then gives him ammo.

Script 1 (void)

```
{
    SetActivator(0, AAPTR_PLAYER1);
    GiveInventory("Clip", 50);
}
```


SetPointer

bool SetPointer(int assign_slot, int tid[, int pointer_selector[, int flags]])

Usage

Set the value of one of the activator's stored actor pointers.

Note: The function tests for circular reference on master and target fields, setting them to NULL if one is found. That test does not occur when assigning to the tracer field. Any attempt to make the actor point directly to itself will also result in a NULL assignment.

Parameters

int assign_slot: an actor pointer selector. Must refer to an assignable pointer type (target, master, tracer).

int tid: TID of the actor to be stored in the selected slot. 0 selects the activator, but the caller can only be an intermediate selection.

int pointer_selector: if this is specified, the actor specified by TID is used as an intermediate actor. The final value is determined by selecting any available actor pointer from the intermediate actor.

int flags:
PTROP_UNSAFETARGET: don't nullify assignments that result in an infinite chain of missiles referencing each other.
PTROP_UNSAFEMASTER: don't nullify assignments that result in an infinite chain of actors referencing each other.

Return value

TRUE (1) if a non-NULL assignment was made (a suitable actor was found and stored).

FALSE (0) if there is no activator, or if a NULL assignment was made. This function assigns data to the activator, and cannot operate without one.

Examples

This script will cause any monster with provided TID to attack the calling player's current target. It could be used for summoned minions and the script could be activated via a hot-key.

Script "AttackTarget" (int TID) NET

```
{
  int oldTID, target;
  if(SetActivator(0,AAPTR_PLAYER_GETTARGET)) // Get the target of the activator. If it fails because there is none
  nothing will happen.
  {
    oldTID = ActivatorTID(); // Save the TID of the target.
    target = UniqueTID(); // Get a new unique TID for the target actor.
    Thing_ChangeTID(0, target); // Set that TID to the actor.
    SetActivator(TID); // Set the activator to the actor with the given TID.
    SetPointer(AAPTR_TARGET, target); // Give a new target to it.
    Thing_ChangeTID(target,oldTID); // Restore the previous TID, because it might be used already by other scripts.
  }
}
```

Actor properties

The following properties can be set in an actor's DECORATE or ZScript's Default {} block definition. They control various characteristics of your actor, from appearance to behavior or the actor's physical properties. To use, simply specify the property and any associated values on its own line within the actor's definition and outside the state block.

Actor properties

Map editing control

Game gamename (DECORATE only)

Defines the game in which the editor number and spawn ID should take effect. For normal custom WADs this is not necessary but it may have some use in resource WADs targeted at multiple games. Valid game names are:

Chex

Doom

Heretic

Hexen

Strife

Raven (Heretic+Hexen)

Multiple game statements are additive.

SpawnID value (DECORATE only)

Defines the spawn number to be used with Thing_Spawn and its derivatives.

ConversationID value (DECORATE only)

Defines the ID used by the Strife's dialogue system. This property allows to use dialogues in all games, not just Strife. If you use ZSDF instead of the binary format or USDF, then this is unnecessary and instead is set on a per-actor basis on the map.

Note: some standard Strife actors define three numbers here, but for a mod only the first is needed. The latter two are used by the two "teaser" demo versions of Strife, and since ZDoom does not allow to use mods with a demo version, adding support for the Strife teasers in a mod is pointless.

Tag name

Gives the actor a name. The names are used in the key display on Strife's status bar and the default actor name in Strife's dialogs, as well as the name of the currently selected inventory item when the invquery console command is called.

Behavior

Health value

Defines the health a monster or any other shootable item starts with.

Default is 1000.

GibHealth value

Defines the (negative) health below which this actor enters the extreme death sequence. If this is not set the value depends on the game, as defined in the GameInfo definition with the gibfactor property. In Doom it is the negative of the spawn health, in the other games the half of the negative of the spawn health.

WoundHealth value

Defines the health below which the actor enters the Wound state. The default is 6.

ReactionTime value

Time in tics (1/35 seconds) a monster takes to attack after being alerted. Attacking a monster will cause it to return attack immediately, regardless of reaction time. A negative value will cause a monster to never attack unless attacked first. There is normally no need to change this value but it can be used as a counter for A_Countdown.

Default is 8.

PainChance value

PainChance damagetype, value

Probability of entering the pain state (256 = always, 0 = never). You can also specify PainChance per damage type. Sources of damage that have the FORCEPAIN flag ignore this property.

Default is 0.

PainThreshold value

Minimum amount of damage required by the actor to potentially enter its Pain state. Sources of damage that

have the FORCEPAIN flag ignore this property.

Default is 0.

DamageFactor type, value

DamageFactor value

If the actor takes damage of the specified type, that damage is multiplied by the specified value.

A comma is required between type and value. value can be a decimal value; zero means the actor is completely immune to this kind of damage.

Default is 1.0.

The maximum this value can go is 32767. DamageFactor is ignored if the damage is 1,000,000 or more.

See also Custom damage types

If no type is provided, the factor applies to all damage types, not specified for the actor.

SelfDamageFactor value

A multiplier for self-inflicted damage. This is ignored if the damage is 1,000,000 or more.

Default is 1.0.

DamageMultiply value

Generic damage multiplier. This is typically used on actors (e.g. monsters) to strengthen or weaken the damage they deal from their attacks.

Default is 1.0.

Damage value

Damage (expression) (DECORATE only)

For a projectile defines the damage it inflicts upon impact. The formula is $\text{random}(1,8) * \text{damage}$, or $\text{random}(1,4) * \text{damage}$ if the STRIFEDAMAGE flag is set.

This also defines the damage for actors which attack like the Lost Soul. The formula for this is $\text{random}(1,8) * \text{damage}$.

damage is also used to define how many bullets are fired by the generic hitscan attack function

A_BulletAttack.

Custom damage formulas can be specified by enclosing the value completely within parenthesis. For example: $\text{damage}(\text{random}(4,8) * 5 + 6)$. This bypasses the normal calculation and does the exact damage resulting from the custom formula. Also, ACS_ExecuteWithResult and user variables can be used as part of the damage expression; this allows for more sophisticated damage calculations to be performed. The damage formula is evaluated at the moment the actor causes damage (e.g. projectile/hitscan impact).

Default is 0.

DamageFunction expression (ZScript only)

Similar to Damage, it defines the damage of the actor. What is passed is always treated as an expression, be it a single value or a formula, and is not subject to randomization like the case can be with Damage. This property and Damage are considered to be the same property, i.e. an actor can only have one or the other, but not both.

PoisonDamage value [, duration [, period]]

Gives a projectile poison damage. Poison damage is not inflicted all at once, but affects the victim for a certain amount of time. The value specified here defines the time the poisoning effect lasts. Remember, poison damage only affects players if only value is specified (Hexen-style poison). Otherwise, it also affects other actors (ZDoom-style poison).

Poison damage is inflicted every period tics (the default value of 0 means once per second). If a non-zero duration is given, it is used for the total length of the poisoning, otherwise the length depends on the value. See also ADDITIVEPOISONDAMAGE and ADDITIVEPOISONDURATION.

PoisonDamageType type

Changes the damage type of poison, overriding its default type.

RadiusDamageFactor value

Defines the factor radius damage on this actor is multiplied by.

RipperLevel value

Used by projectiles or anything capable of performing with the RIPPER flag. Specifies the 'level' or how high up a projectile can rip through in terms of monsters with different minimum and maximum values.

A projectile that is below a monster's minimum or above maximum means that projectile will not rip through them. All ripper properties are ignored if the DONTRIP flag is present on the monster, though, and will not rip regardless of level through that particular monster.

When used on a monster, however, and no min or max is defined, only that specific level can rip through it. The value can be negative for this, RipLevelMin and RipLevelMax.

RipLevelMin value

Used by monsters. Only projectiles that meet this level or higher can rip through this monster. Default is 0.

RipLevelMax value

Used by monsters. Only projectiles that meet this level or lower can rip through this monster. Default is 0, which means anything above the minimum can rip through it.

DesignatedTeam number

Assigns the actor to the specified team by its number from TEAMINFO. In team play, the actor will not take friendly fire from teammates unless the teamdamage console variable is set. If the monster is friendly, it will not target teammates.

Default is no team.

Speed value

Defines how fast an actor moves. For projectiles this is the distance it moves per tic (1/35 seconds). For monsters it defines the size of one step done in A_Chase. Player's movement speed also depends on the Player.ForwardMove property.

Default is 0.

VSPEED value

Gives an actor an initial vertical momentum. The higher the value, the more momentum. This is mainly used for special effect actors (e.g. weapon puffs) with the NOGRAVITY flag to make them float upwards.

FastSpeed value

Defines the speed for 'fast monsters' and nightmare mode. Please note that if you use this with a monster that monster will have the different speed for the rest of its life. Fast Monster's mode will make monsters and projectiles use their FastSpeed property instead of their defined speed when +fastmonsters or +alwaysfast flag is used. Fast monsters halves the duration of actor states with the Fast keyword. Fast monsters can be set in many ways like using the +ALWAYSFAST flag or +fastmonsters.

FloatSpeed value

Defines the speed the object floats up/down when FLOAT flag is set.

Default is 4.

Species string

Defines the species the monster belongs to. This determines infighting behavior, since a monster's missile do not inflict harm (except through splash damage) to other monsters belonging to the same species by default. When used with keys, this property allows for new keys to open the same locks as the keys they are referencing with the property, without the need to redefine the locks to add the new keys. For example, setting the key's species to RedCard will allow it to open locks as if it were a red keycard.

Default is None that makes the game to look up the actor this actor inherits from and has this property set to something else.

Accuracy value

Defines the object's accuracy. The higher the value, the more accurate the object is said to be.

For players, this value is factored into the accuracy calculation for Strife's native weapon-firing functions.

For monsters and other actors, this value has no predetermined effect, but it can be referenced in other places via the accuracy DECORATE variable and the APROP_Accuracy actor property.

Stamina value

Defines the object's stamina.

For players, each point of stamina will add one point to the player's maximum health pool. The value is also used in the damage calculation for Strife's punch dagger.

For monsters and other actors, this value has no predetermined effect, but it can be referenced in other places via the stamina DECORATE variable and the APROP_Stamina actor property.

Activation flags

Defines how an actor (henceforth called "thing") will execute its special. By default, the special is executed when the thing dies. If the thing is given the BUMPSPECIAL or USESPECIAL flags, the special will also execute upon being bumped into or used by another actor (the "trigger"). Several activation flags can be used, if they are separated by the | character. Available flags are:

THINGSPEC_Default — default behavior. The trigger (bumper/user/killer) is the activator. Only a player may activate it through BUMPSPECIAL or USESPECIAL.

THINGSPEC_ThingActs — the thing is considered the activator.

THINGSPEC_TriggerActs — the trigger is considered the activator. This is the default behavior except for specials activated upon the thing's death in maps with the ActivateOwnDeathSpecials level flag. This flag overrides the MAPINFO flag.

THINGSPEC_ThingTargets — the thing changes its target field to the trigger when triggered.

THINGSPEC_TriggerTargets — the trigger changes its target field to the thing when it triggers it.

THINGSPEC_MonsterTrigger — monsters are allowed to trigger the thing.

THINGSPEC_MissileTrigger — missiles are allowed to trigger the thing.

THINGSPEC_ClearSpecial — the thing's special is cleared after execution.

THINGSPEC_NoDeathSpecial — the thing's special will not execute on death.

THINGSPEC_Activate — the thing is activated when triggered.

THINGSPEC_Deactivate — the thing is deactivated when triggered.

THINGSPEC_Switch — the thing is alternatively activated and deactivated when triggered. If used in conjunction with THINGSPEC_Deactivate, it will be deactivated the first time it is triggered, otherwise it will be activated.

TeleFogSourceType classname

Defines the fog to leave behind when an actor teleports away. Can be changed with A_SetTeleFog.

Default is TeleportFog.

TeleFogDestType classname

Defines the fog the actor will spawn once it arrives. Can be changed with A_SetTeleFog.

Default is TeleportFog.

Threshold value

Defines how long an actor will stick to one target after switching targets caused by infighting. e.g. if an imp changes targets because it was shot by a zombieman, it must call A_Chase this many times before it's ready to switch again. Note that further attacks which cause pain from the same target reset this value, making them stick with that target longer. QUICKTORETALIATE overwrites this property, and the property is adjustable with A_SetChaseThreshold. Note that this property is reset to DefThreshold, allowing monsters to have flexible infighting timers.

DefThreshold value

Sets the default threshold reset value when the actor switches monster infighting. Adjustable with

A_SetChaseThreshold.

FriendlySeeBlocks value

(Warning: setting this property to high values is not recommended, as it could have a negative impact on performance.)

Defines the maximum radius that a friendly monster or unfriendly monster with SEEFRIENDLYMONSTERS on can see enemies within. This is measured in blocks of 128x128 map units.

Default is 10 (1280 map units).

Collisions and physics

Radius value

Defines the radius of this actor.

Default is 20.

Notes:

Using very small values can provoke glitches in collision detection.

The total diameter is double this value, so an actor with a radius of 64 is actually 128 units in length and width, making it impossible for them to navigate through a corridor that's 128 units wide, for example.

Height value

Defines the height of this actor.

Default is 16.

Note: using very small values can provoke glitches in collision detection.

DeathHeight value

Defines the height this actor has after dying.

Default is 1/4 of the original height.

BurnHeight value

Defines the height this actor has after dying by fire.

Default is 1/4 of the original height.

ProjectilePassHeight value

Defines the height of the actor for the purposes of allowing projectiles to pass through. If this is 0, the actor's height is used. If the value is positive, projectiles over this height will pass through the actor. If the value is negative, the absolute value will be used if the associated compatibility option is turned on.

Gravity value

The amount of gravity that affects this actor. Values less than 1 reduce gravity, while values greater than 1

increase it.

Default is 1.0.

Friction value

The amount of friction that affects this actor. Values less than 1 increase friction, while values greater than 1 reduce it. Note that this is multiplied by the sector's own friction.

Default is 1.0.

Mass value

Defines the mass of this actor. The larger the mass the less an actor moves when being thrust by damage. If less than 10, the actor will also make small terrain splashes rather than normal ones.

Default is 100.

Note: Giving an actor an extremely high mass in an effort to make it unthrustable is not recommended. If an actor is stuck in another or in a wall and `A_RadiusThrust` affects the actor, the actor could be killed almost instantly if `RTF_NOIMPACTDAMAGE` isn't used. It's recommended to use the `DONTTHRUST` flag instead of `0x7FFFFFFF` or the likes.

MaxStepHeight value

Defines the maximum height of a step this actor can climb up when moving. Default is 24.

MaxDropOffHeight value

Defines the maximum height of a step this actor can climb down when moving. Normally this only applies to monsters but when you set the `NODROPOFF` flag it applies to all forms of movement.

Default is 24.

MaxSlopeSteepness value

Defines the maximum incline of a slope that can be walked on when moving, based on the Z component of its normal. Thus, a value of 1 would make only perfectly flat floors walkable, while any slope would push down the actor and a value of 0 would make practically any kind of slope walkable, no matter how steep. Default is `STEEPSLOPE` ($46342 / 65536 = 0.707122\dots$), which limits movement to slopes of 45 degrees and below.

BounceType name

Defines the way the actor bounces. See also bounce flags. Valid names include:

None: actor does not bounce at all. This is the default value.

Doom: actor bounces off planes and walls until it loses enough momentum to stop.

Heretic: actor bounces off planes only — and is put in its Death state when bouncing!

Hexen: actor bounces off planes and walls.

Classic: actor bounces off planes only, and is considered an MBF bouncer.

Grenade: actor bounces off planes and walls, and is considered an MBF bouncer.

DoomCompat, HereticCompat, HexenCompat: same as Doom, Heretic or Hexen respectively, but the actor uses its `SeeSound` instead of its `BounceSound` and `WallBounceSound`.

BounceFactor value

Defines the factor to which this actor's momentum will be reduced when bouncing off of ceilings and floors. This only has an effect if `BounceType` is not None. Default is 0.7. The `WallBounceFactor` property can be used to decrease the actor's momentum when it bounces off of walls.

WallBounceFactor value

Defines the factor this actor's momentum will be reduced for when bouncing off walls. This only has an effect if `BounceType` is not None. Default is 0.75.

BounceCount value

Defines the amount of bounces this actor can do before it explodes. This only has an effect if `BounceType` is not "None". The default is infinite but for example Strife's incendiary and explosive grenades only bounce twice before exploding.

ProjectileKickBack value

Similarly to `Weapon.KickBack`, this affects how far the projectile will push its target upon impact. This property overrides a weapon's kickback for projectiles thrown by weapons.

PushFactor value

Defines how much an actor with the `PUSHABLE` flag set can be pushed around. The pusher's momentum is multiplied by this amount and applied to the pushed actor. The default value is 0.25. A value of 0.0 is equivalent to not having the flag set, while a value greater than 1.0 makes it move faster than the pusher.

WeaveIndexXY value

Defines the initial weave index for projectiles using the `A_Weave`, `A_BishopMissileWeave` and `A_CStaffMissileSlither` functions. The value must be in the 0—63 range.

WeaveIndexZ value

Defines the initial weave index for projectiles using the A_Weave, A_BishopMissileWeave and A_CStaffMissileSlither functions. The value must be in the [0,63] range.

ThruBits value

Allows defining collision disabling 'groups' with bit shifted numbers. This only takes effect if ALLOWTHRUBITS is on the actor.

Default is 0, which collides with everything no matter what group is defined for others.

enum EGroups

```
{
    ThruGroup1 = 1 << 0,
    ThruGroup2 = 1 << 1,
    //...
    ThruGroup31 = 1 << 30,
    ThruGroup32 = 1 << 31 // This is the highest it can go, totaling in 32 groups.
};
```

Default

```
{
    +ALLOWTHRUBITS // The flag is needed to activate it.

    // Anyone who is in 'ThruGroup3' or 'ThruGroup4' will pass through as well. Groups can be combined
    with the '|' symbol without the .
    ThruBits (ThruGroup3|ThruGroup4);
}
```

Sound

ActiveSound soundname

Defines the sound the actor makes when active.

AttackSound soundname

Defines the sound the actor makes when attacking.

BounceSound soundname

Defines the sound the actor makes when bouncing. If WallBounceSound is defined, applies only when bouncing off a floor.

CrushPainSound soundname

Defines the sound the actor makes when being crushed.

DeathSound soundname

Defines the sound the actor makes when dying or when a projectile explodes. For non-projectiles the death sound must be explicitly played with A_Scream.

HowlSound soundname

Defines the sound the actor makes when being electrocuted or poisoned.

PainSound soundname

Defines the sound the actor makes when in pain. To hear this sound A_Pain has to be called.

RipSound soundname

Defines the sound the projectile makes when it rips through damage-able actors.

Default is "misc/ripslop".

SeeSound soundname

Defines the sound the actor makes when it sees the player (for monsters) or when a projectile is spawned.

WallBounceSound soundname

Defines the sound the actor makes when bouncing off a wall. If this property is not defined, the BounceSound is played instead.

PushSound soundname (New from 4.10.0)

Defines the sound the actor makes when pushed by another actor. This only works on actors with PUSHABLE.

Rendering

RenderStyle type

Defines how this actor is rendered. Possible values include:

"None" - actor is invisible.

"Normal" - actor is visible and not translucent.

"Fuzzy" - like the Spectre in Doom.

"SoulTrans" - actor is translucent, to an amount determined by the transsouls CVAR.

"OptFuzzy" - actor is fuzzy or translucent, based on user preference.

"Stencil" - actor is drawn as a single color. Use StencilColor property to set the color to use.

"AddStencil" - the same as Stencil, only additive.

"Translucent" - actor is translucent.

"Add" - actor uses additive translucency.

"Subtract" - actor uses reverse-subtractive translucency.

"Shaded" - treats 8-bit indexed images as an alpha map. Index 0 = fully transparent, index 255 = fully opaque. This is how decals are drawn. Use StencilColor property to colorize the resulting sprite.

"AddShaded" - the same as Shaded, only additive.

"Shadow" - equivalent to black translucent stencil with an alpha of 0.3.

Default is Normal.

Alpha value

Defines the opacity/intensity for render styles Translucent, Add, Subtract and Stencil. The range is [0.0, 1.0].

Default is 1.0.

DefaultAlpha

Sets the alpha value of the actor to 0.4 when playing Heretic, and to 0.6 when playing the other games.

StealthAlpha value

Defines the minimum visibility value of a stealth monster.

Default is 0.

XScale value

Defines the X-scaling for this actor. Negative values are valid, and will result in mirroring the sprite on the axis. The range is [-4.0, 4.0].

Default is 1.0.

YScale value

Defines the Y-scaling for this actor. The range is [0.0, 4.0].

Default is 1.0.

Scale value

Combines XScale and YScale. The range is [0.0, 4.0].

Default is 1.0.

LightLevel value

Specifies the absolute light level of the actor, independent of the sectors' light level. This value can range from 0 to 255. The ADDLIGHTLEVEL flag makes this property add or subtract from the actors' current sector light level.

Default is -1, which means that the actor only uses the sectors' light level.

Translation value

Sets one of Doom's standard translations (which translate the color green range) as default. The range is 0-2 and corresponds to the standard Doom translations of gray (0), brown (1), and red (2). Strife defines more, but they can be used only while playing it.

Default is no translation.

Translation translationstring [, translationstring [,....]]

Translation "translationname"

Defines a custom translation for this actor. There can be 65536 custom translations at the same time. All specified translation strings are added to create one final translation table (see Translation for the syntax of translation strings).

The translation can either be set directly, or defined in TRNSLATE and then referenced by its name.

Translation Ice

This special usage of the Translation property will instruct ZDoom to use the built-in ice translation when rendering this actor. The main advantage of this is for GZDoom, where the ice range uses colors outside the normal Doom palette and is thus difficult to recreate with a normal translation.

BloodColor color

Sets the actor's blood color. There can be up to 256 different custom blood colors at the same time. color is either specified as 3 integers, a string "rr gg bb" or a string containing one of the special color names.

BloodType replaceblood [, replacebloodsplatter [, replacebloodaxe]]

Tells ZDoom what class to spawn when the actor is hit with a weapon, replacing the standard blood. If only one type is specified it will be used for all 3 blood actors.

Decal decalname

Allows specifying the decal this actor creates here instead of having to edit the DECALDEF lump as well. Decals can be inherited this way instead of being specified for each actor. Note that the standard Doom projectiles' decals are defined through DECALDEF and custom actors based on them will not inherit their decals.

Decals can be specified within weapon definitions or bullet puff definitions. If both exist, the weapon definition takes precedence unless the FORCEDECAL flag is set on the puff.

Railgun puff decals require the FORCEDECAL flag to be set on the puff.

If a projectile is to leave a decal, it must exist for at least a tic in its Death state — otherwise no decal will be sprayed.

StencilColor color

Sets the color to use when using the Stencil or Shaded RenderStyle.

FloatBobPhase value

Sets the initial bob phase for the actor. Actors with the same phase value bob in-sync with each other. The range of phases is from -1 to 63, with -1 randomly choosing phase from the [0,63] range.

Default is -1.

FloatBobStrength value

Sets the magnitude of the bobbing behavior created by the FLOATBOB flag. A value of 1.0 produces the behavior seen in Heretic and Hexen, while higher and lower numbers produce a more and less extreme bobbing, respectively.

Default is 1.0.

DistanceCheck cvarname

An actor with this property is not rendered if the player's view is at a distance away from the actor which is equal or greater than the value of the specified console variable. Only integral console variables are acceptable.

SpriteAngle angle

Sets the absolute angle for the sprite to draw at that given angle. This requires +SPRITEANGLE to have any effect, and once enabled, overrides all other sprite angle properties. This does not affect an actor's actual angle, rather the sprite a player would see if viewing them from that angle instead. Useful for things like decals, FLATSPRITE, and more. See GetSpriteAngle and A_SetSpriteAngle for manipulation at runtime.

0 is the actor's backside.

90 is the actor facing to the right.

180 is the actor's front. Etc.

SpriteRotation angle

Offsets the actor's sprite angle in degrees. This allows for actors to visibly face one direction while truly looking another way. Use GetSpriteRotation and A_SetSpriteRotation for manipulation at runtime.

VisibleAngles start, end

Defines the range to draw the sprite if the player camera is looking at it from that particular angle range. Viewing it from outside this range will not render it. Can be modified with A_SetVisibleRotation.

VisiblePitch start, end

Same as VisibleAngles, but applies to pitch instead.

NOTE: VisibleAngles and VisiblePitch requires the MASKROTATION flag to work; also they must be in bundle to work correctly (bug?), e. g. "VisibleAngles -35,35nextlineVisiblePitch -180,180".

RenderRadius radius

Defines visual collision box in a similar manner to Radius. The actor is guaranteed to be drawn if any sector within this range is drawn, even if actor's main sector is not visible. This also affects dynamic lights around the actor, which is useful for ensuring distant lights illuminating an actor with a large model properly.

If this value is smaller than Radius, Radius is used instead. A value of 0 restricts the actor to its source sector, matching vanilla Doom engine behavior; while a value of -1 causes the actor to be removed even from the source sector's render list, which can be useful to optimize performances of actors supposed to be invisible.

Default is 0.0.

CameraHeight value

Defines the height to render the view at when the actor is used as a camera.

CameraFOV value

Defines the field of view when the actor is used as a camera.

Default is 90.0.

Obituaries

Obituaries are messages displayed in the log when a player is killed by a monster, projectile or weapon. The string can be either used directly in the DECORATE code, or put in a LANGUAGE lump and referred to by its identifier. Starting a string with the \$ character marks it as an identifier. The following placeholders can be used within the string itself:

%o: victim's name

%k: killer's name

%g: he/she/they/it, depending on the gender of the victim

%h: him/her/them/it

%p: his/her/their/its

%s: his/hers/theirs/its

%r: he's/she's/they're/it's

The actors defined in zdoom.pk3 always use a reference to an identifier in the LANGUAGE lump; however for reference purposes the corresponding English string is provided as a comment. You can override a standard obituary by redefining it in a custom LANGUAGE lump, avoiding the need to replace the entire actor.

Note that in deathmatch, if the cl_bbannounce console variable is true, the Bloodbath announcer replaces weapon obituaries by its own random strings.

HitObituary string

Defines the obituary string for a melee attack by this actor. If not present obituary is used.

Obituary string

Defines the obituary string for this actor. For weapons, this property can be used either directly on the weapon or on the puff for hitscan or rail attacks, but it should be on the projectile instead for missile attacks.

Attacks

MinMissileChance value

This is one part of the missile attack probability calculation. The lower this value is the more aggressive the monster is. This can be combined with the MISSILEMORE and MISSILEEVENMORE flags.

Default value is 200.

DamageType type

The attack uses a specific damage type. A number of predefined damage types are available, but it is possible to define your own custom damage types. Damage types are identified only by their name and do not need to be declared anywhere else. An attack can only use the last defined DamageType assigned to it, so more than one DamageType on an attack is pointless.

DeathType type

The attack uses a specific death type, which overrides its damage type. Death types are identified only by their name and do not need to be declared anywhere else. This can be useful to combine special Death animations with a variety of different DamageFactors.

MeleeThreshold value

The actor with this property set will not attack its target with a missile attack if it is within the specified range, but rather (if chasing) attempt to close in to engage in melee. To be used in place of the old LONGMELEERANGE flag.

MeleeRange value

Specifies the maximum distance to a target where a melee attack inflicts damage. Distance is calculated from the attacker's center to the target's bounding box.

All monster melee attacks use this value for melee range. This include A_BasicAttack, A_ComboAttack, A_CustomComboAttack, A_CustomMeleeAttack, A_MeleeAttack, and so on.

Default is 44.

MaxTargetRange value

A monster with this property set will not attack its target unless it is within the specified range. To be used in place of the deprecated SHORTMISSILERANGE flag. This is used by A_Chase and similar functions.

Seeker missiles have a special use for this property. See A_SeekerMissile for more information.

MeleeDamage value (deprecated)

MeleeSound soundname (deprecated)

MissileHeight value (deprecated) (Except with FastProjectile)

MissileType classname (deprecated) (Except with FastProjectile and a few others)

These properties served as parameters for A_MeleeAttack, A_MissileAttack and A_ComboAttack. However, these three functions have been superseded by parameterized versions called A_CustomMeleeAttack, A_CustomMissile and A_CustomComboAttack, respectively.

Note that FastProjectile, ArtiPoisonBagShooter, ArtiPoisonBagGiver, and classes using A_LightningZap or A_LastZap make special use of the missile properties.

ExplosionRadius value

A_Explode will use this value as the explosion's radius, unless a different value is provided in the function call itself. This property can be useful for inheritance purposes, to override explosion damage in a child actor.

ExplosionDamage value

A_Explode will use this value as the explosion's damage, unless a different value is provided in the function call itself. This property can be useful for inheritance purposes, to override explosion damage in a child actor.

DontHurtShooter (deprecated)

These properties were used to pass the parameters to A_Explode. Now that A_Explode allows passing the parameters directly there is no need to use these except for compatibility with older definitions.

PainType type

The attack uses a specific pain type, which overrides its damage type. Pain types are identified only by their name and do not need to be declared anywhere else. This can be useful to combine special scripted effect based on custom Pain states with a variety of different DamageFactors.

Flag combos

Monster

Sets all appropriate flags to make this actor act like a regular monster. The following flags are set:

SHOOTABLE

COUNTKILL

SOLID

CANPUSHWALLS

CANUSEWALLS

ACTIVATEMCROSS

CANPASS

ISMONSTER

Projectile

Sets all appropriate flags to make this actor act like a regular projectile. The following flags are set:

NOBLOCKMAP

NOGRAVITY

DROPOFF

MISSILE

ACTIVATEIMPACT

ACTIVATEPCROSS

NOTELEPORT

When playing Heretic or Hexen BLOODSPATTER is also set.

Special

Args arg0[, arg1[, arg2[, arg3[, arg4]]]]

Hardcodes arguments of the actor, they can be any integer value. They will override any argument set in the map editor; using this property is therefore generally not recommended unless if making actors for the Doom map format or actors for the Hexen map format that needs argument values outside the [0,255] range.

ClearFlags

Clears all flags previously set.

DropItem classname [, probability [, amount]]

The item the actor is set to drop. The action of dropping the item is performed by A_NoBlocking, which is typically called during the death sequences of the actor. Optionally the probability can be specified. A probability of 255 or greater means the item is always dropped, while a probability of -1 or less means it is never dropped. When classname is None the drop list will be completely removed.

amount is only applicable for ammunition. It specifies how much single rounds the dropped item contains.

RandomSpawner, BossEye, SpawnShot, Weapon and Skulltag's CustomMonsterInvasionSpot and

CustomPickupInvasionSpot use this property differently.

There can be more than one dropitem definitions for an actor. The amount of definitions per actor is unlimited.

dropitem works differently with inherited actors. An actor can inherit a list of dropitem definitions from its parent but if it defines one of its own the entire parent's list is discarded. This parameter does not work for players.

Spawn (deprecated)

See (deprecated)

Melee (deprecated)

Missile (deprecated)

Pain (deprecated)

Death (deprecated)

XDeath (deprecated)

Burn (deprecated)

Ice (deprecated)

Disintegrate (deprecated)

Raise (deprecated)

Crash (deprecated)

Wound (deprecated)

Crush (deprecated)

Heal (deprecated)

These keywords were originally used to remap states. However, as of 2.1.0 they have been deprecated because now it is possible to place goto statements right behind state labels which gives the same effect in a way that is much cleaner and more flexible.

States

Defines an actor's states.

Skip_Super

Reinitializes the actor as if it has no parent. This can be used to have access to the parent's states without inheriting its attributes.

VisibleToTeam team_number

Renders the actor visible to players of a specific team. The actor's physics and code will act normally, however.

Can be combined with VisibleToPlayerClass. Originally a Skulltag feature.

VisibleToPlayerClass classname[, classname2[, classname3[, ...]]]

Same as VisibleToTeam, but for player classes.

Can be combined with VisibleToTeam. Originally a Skulltag feature.

Instead of just one class, a comma-separated list of classes can be given. Note that this property uses inheritance, so if an actor is visible to class DoomPlayer, it will be visible by all classes derived from DoomPlayer as well.

Additional properties

Subclasses of Actor may have additional properties. Most notably, Inventory and its own subclasses (such as Weapon) do; they are listed with their respective classes. PlayerPawn also defines its own properties.

Inventory

Note that all of Inventory's subclasses (Ammo, Weapons, etc.) can be used with the following inventory properties. The Inventory base class defines the following properties which are available to all inventory subclasses:

Inventory.Amount value

Sets the amount of inventory items given by this item. Mostly used for item types that give larger quantities.

Inventory.DefMaxAmount

Sets the maximum amount the player can carry of this item to the game's default. That is normally 16 for Heretic and 25 for the other games, but can be changed in MAPINFO through the GameInfo section.

Inventory.MaxAmount value

Sets the maximum amount the player can carry of this item.

Inventory.InterHubAmount value

Sets the maximum amount the player can keep of this item when traveling between hubs or non-hub levels. The default value of one replicates the original behavior of Heretic. This property replaces the deprecated INVENTORY.INTERHUBSTRIP.

Inventory.Icon sprite

Defines the icon this item uses when displayed in the HUD or status bar.

Inventory.AltHUDIcon sprite

Defines the icon this item uses when displayed in the alternate HUD. This is of use with keys, weapons, ammunition and artifacts.

Inventory.PickupMessage message

Defines the message that is printed when the player picks up this item. Strings from the LANGUAGE lump can be used by passing the string's identifier prefixed with '\$'. For Health items, this can be overridden by the Health.LowMessage property.

Inventory.PickupSound sound

Defines the sound that is played when a player picks up this item.

Inventory.PickupFlash actor

Defines what actor should appear when the item is picked up; normally this would be used with the PickupFlash actor to make a flash appear as the player picks up the item. If this is not specified there will be no flash when the item is picked up.

Inventory.UseSound sound

Defines the sound that is played when a player uses this item. The sound is played on the item channel (CHAN_ITEM) with normal attenuation.

Inventory.RespawnTics value

Defines the time it takes until this item respawns (if respawn is enabled) in 1/35 of a second.

Inventory.GiveQuest value

Also give a quest item for controlling Strife's dialogs when picking up this one. value can be in the range of [1..31]

Inventory.ForbiddenTo classname[, classname2[, classname3[, ...]]]

Prevents players using one of the listed classes (use actor names, not Player.DisplayName) from picking up this type of inventory item. Note that unless playing in cooperative mode, a forbidden player will still try to pickup a Weapon or WeaponPiece for ammunition, as in Hexen. Use an empty string (e.g.,

Inventory.ForbiddenTo "") to clear an inventory item of inherited interdictions.

Inventory.RestrictedTo classname[, classname2[, classname3[, ...]]]

Prevents players not using one of the listed classes (use actor names, not Player.DisplayName) from picking up this type of inventory item. Note that unless playing in cooperative mode, a forbidden player will still try to pickup a Weapon or WeaponPiece for ammunition, as in Hexen. Use an empty string (e.g.,

Inventory.ForbiddenTo "") to clear an inventory item of inherited restrictions.

FakeInventory

FakeInventory.Respawns

Preserves the item's Thing special and arguments when respawning. This property is specific to FakeInventory, and not shared by other Inventory subclasses, because of differences in the ways FakeInventory handles its special and arguments.

BasicArmorPickup

Armor.SaveAmount amount

The amount of armor that this item gives.

Armor.SavePercent percentage

The percentage of damage that the armor absorbs.

Percentage is specified as a floating point value between 0.0 and 100.0 which is converted internally to a floating point value between 0.0 and 1.0

Armor.MaxFullAbsorb amount

The amount of damage that this armor will fully absorb per tic.

Armor.MaxAbsorb amount

The maximum amount of damage (after adjustment) that this armor will absorb per tic.

BasicArmorBonus

Armor.SavePercent percentage

The percentage of dealt damage that the armor absorbs.

Percentage is specified as a floating point value between 0.0 and 100.0 which is converted internally to a floating point value between 0.0 and 1.0

Armor.MaxSaveAmount amount

Sets the maximum amount of armor you can reach by picking up this item.

Armor.SaveAmount amount

The amount of armor that this item gives.

Armor.MaxBonus amount

The amount of armor you obtain additionally to your save amount value.

Armor.MaxBonusMax amount

The maximum additional save amount you can obtain with the armor.

Weapon

Weapon.AmmoGive amount

Weapon.AmmoGive1 amount

The amount of primary ammo you receive from this weapon.

Weapon.AmmoGive2 amount

The amount of secondary ammo you get from picking up the weapon.

Weapon.AmmoType type

Weapon.AmmoType1 type

The type of primary ammo the weapon uses. This must be a valid ammo type.

Weapon.AmmoType2 type

The type of secondary ammo the weapon uses.

Weapon.AmmoUse amount

Weapon.AmmoUse1 amount

The amount of primary ammo the weapon uses per shot.

Weapon.AmmoUse2 amount

The amount of the secondary ammo that the weapon uses per shot. Normally secondary ammo is used by the secondary attack but if the WEAPON.PRIMARY_USES_BOTH flag is set the primary attack will use both types of ammo for each attack.

Weapon.MinSelectionAmmo1 amount

A weapon with this property will not be auto-switched to if the available amount of ammo for the primary attack is less than the value specified. The weapon can still be switched to manually, however.

Weapon.MinSelectionAmmo2 amount

A weapon with this property will not be auto-switched to if the available amount of ammo for the secondary attack is less than the value specified. The weapon can still be switched to manually, however.

Weapon.BobPivot3D (x, y, z) (ZScript only) (development version 46d36cf only)

(Need more info)

Default is (0.0, 0.0, 0.0).

Weapon.BobRangeX amount

Range multiplier for horizontal bobbing.

Default is 1.0.

Weapon.BobRangeY amount

Range multiplier for vertical bobbing.

Default is 1.0.

Weapon.BobSpeed amount

Bobbing speed multiplier. Affects how quickly the bobbing motion occurs.

Default is 1.0.

Weapon.BobStyle type

The type of bobbing to use. Possible values include Normal, Inverse, Alpha, InverseAlpha, Smooth, and InverseSmooth.

Normal is the default value and corresponds to the bobbing motion used in all old Doom-engine games.

Alpha is the bobbing motion used in the DoomWikiLogoIcon.pngalpha versions of Doom.

Smooth is a smoother version of the normal style.

Inverse* types mirror the motion vertically compared to the non-inverse version. InverseAlpha lowers the weapon sprite while bobbing; the others raise it.

Weapon.KickBack amount

How far attacks by this weapon push the enemy back. Damage is also a factor in kickback.

Weapon.DefaultKickBack

Weapons with this property have a kickback value which is specified by the defkickback MAPINFO property.

Weapon.ReadySound sound

The sound the weapon plays in its ready state.

Weapon.SelectionOrder value

Defines the place in the weapons list when ammo runs out. Lower numbers have higher priority. The

standard Doom II weapons range from 100 for the Plasma Rifle to 3700 for the Fist.

Weapon.SisterWeapon weapontype

Defined weapon is also given when flagged weapon is picked up. Used primarily for Tome Of Power attacks but it can also be used to create weapons with 2 distinct attack modes. Several Strife weapons use this.

Weapon.SlotNumber value

Default slot for this weapon. For mods that want to add new weapons without completely redoing the player's arsenal. For other ways to assign weapons to slots, see weapon slots.

Weapon.SlotPriority value

Selection order within slot for this weapon. This is a floating-point value. Higher numbers mean higher priority.

Weapon.UpSound sound

The sound played when the select state is called.

Weapon.WeaponScaleX value

The horizontal scale multiplier of the weapon's HUD sprite.
Default is 1.0.

Weapon.WeaponScaleY value

The vertical scale multiplier of the weapon's HUD sprite.
Default is 1.2.

Weapon.YAdjust amount

Shifts the HUD sprites vertically when the status bar is off. Note: positive y values shift the weapon down.
The sound played when the select state is called.

Weapon.LookScale amount

Multiplier for the player's look sensitivity. Operates identically to the look sensitivity for FOV scaling, but without changing the FOV.

Ammo

Ammo.BackpackAmount amount

The amount of this ammo type received from a backpack.

Ammo.BackpackMaxAmount amount

The maximum amount of this ammo type that can be carried with a backpack.

Ammo.DropAmount amount

Specifies the amount of ammo a dropped item without any explicit amount given contains. Since you can specify the amount with DropItem this is mainly to customize the amount of items dropped by Strife dialogue scripts.

WeaponPiece

WeaponPiece.Number number

The piece number. This can be a value between 1 and 32. To complete a weapon all available pieces as specified in the weapon must be picked up.

WeaponPiece.Weapon weapontype

The weapon this item helps to assemble.

Health

Health.LowMessage value, message

When picker's health is lower than value, the pickup message is set to message.

PuzzleItem

PuzzleItem.Number value

Defines the number that has to be used with UsePuzzleItem to identify the item.

PuzzleItem.FailMessage string

Message to be displayed when this item is used and not the one requested by UsePuzzleItem.

PuzzleItem.FailSound sound

Sound to be played when the item is used, but is not the one requested by UsePuzzleItem. Default is the player class specific *puzzfail sound.

PlayerPawn

Player.AirCapacity value

Sets the air supply for the player class. This acts as a multiplier for the level's air supply value. A value of 2 will double the air supply, for instance. A value of 0 or less means the air supply is infinite, and thus the player class cannot drown.

Default is 1.0.

Player.AttackZOffset value

The height offset (in Doom map pixels) from the center of the player at which his attacks are fired. Scales according to crouched height.

Player.ClearColorSet number

Removes a color set from the player class. This allows to remove color sets inherited from a parent class.

Player.ColorRange start, end

Defines the color range in which to translate the player's sprite.

Default is no translation (0, 0).

Player.ColorSet number, name, start, end, color [...]

Defines a preset color set for the player class. The number must be unique. The name is used to identify the preset in the player setup menu. The start and end parameters determine the translation for the ColorRange, and the color parameter is used as the visual identifier for the set in the scoreboard.

Up to six additional color ranges can be defined, each with four parameter for range_start, range_end, first_color and last_color, in this order. See StrifePlayer for an example.

Player.ColorSetFile number, name, table, color

Defines a preset color set for the player class. The number must be unique. The name is used to identify the preset in the player setup menu. The translation is indicated by a translation table lump using the format defined by Hexen (TRANTBL0 to TRANTBLD). The color parameter is used as the visual identifier for the set in the scoreboard.

Player.CrouchSprite sprite

Defines the sprites for when your skin is crouching. If you don't define them your sprite will shrink half its vertical size to show it is crouching. For the crouch sprites you have to define all frames apart from the dying and gib frames. Please note that there is no "Crouch" state that is definable for players, as the engine would not support such a feature. Players need to be able to move seamlessly between crouching and standing during any given state (See, Missile, Melee, etc.) without losing their place in that state, hence there can be no generic "Crouch" state to jump to.

Player.DamageScreenColor color[, intensity[, damagetype]]

The color the player's display turns when the player is in pain. This can be specified as three hex values for R, G, B respectively, or as a literal color name (e.g. "blue"). If this isn't defined, it defaults to red. The intensity of the flash can be scaled from 0.0 to 1.0. If a damagetype is specified, the color only applies when the last damage type received is that type. If there is no color specified for a given damage type, it defaults to what is set without one.

Player.DisplayName string

This is the identification string of the player class. It is used in addplayerclass KEYCONF command, in playerclass console variable, in skin definitions, and in menus. Each player class used in game must have a unique display name!

Player.Face string

This is the three-character prefix that will be used by the status bar face display built into DOOM and by custom status bars that define faces; note that with custom status bars in SBARINFO, faces can be used in all games (not just DOOM), provided the necessary graphics are supplied. This is like the face property of S_SKIN but for player classes. To include a status bar face you must put the 3 first letters of the sprite for the face here. For example, if one of the animations was BUGST00 you would put Face "BUG".

When deciding which face to use, the engine checks for STF**** (DOOM games only), the SBARINFO face, the current player class face, the face of the currently loaded skin and finally the morphed animal player class face; whichever it finds last "wins".

The engine also maintains faces across morphing; for example, if you were playing deathmatch in HeXen and have loaded an SBARINFO which defines faces for the fighter and also a DECORATE script containing a custom morph attack weapon that changes your enemy into a dog and includes dog face graphics; if you are turned into a dog, the status bar would change to show your face as a dog and turn back to a human again when you unmorph.

Player.FallingScreamSpeed value_min, value_max

When a player is falling within this range of speeds, they will play *falling sound.

Default is (35.0, 40.0).

Player.FlechetteType string

Type of fléchette used by this player class.

Player.FlyBob value (development version 0d23816 only)

A multiplier for how much the players' view and height bobs up and down when they are flying and the fviewbob console variable is on.

Default is 1.0.

Player.ForwardMove value[, value-run]

Speed modifier for forward/backward movement. The value is for walking and value-run is for running.

Default is 1 for both values. The default run speed is double the walking speed, so Player.ForwardMove 1, 0.5 would disable running.

Player.GruntSpeed value

The minimum speed a player must be falling at the time of landing to play *grunt sound.

Default is 12.0.

Player.HealRadiusType string

The nature of the healing effect the player offers to his allies in cooperative multiplayer when using a Mystic Ambit Incant.

Value can be Health, Armor or Mana. Default is Health.

Player.HexenArmor base value, value armor, value shield, value helm, value amulet

The player's armor class values in Hexen. All values must be a multiple of five (they are divided by five when displayed on the HUD). Base value is the player class's minimum, all other values are the increases gained by picking up the corresponding Hexen armor item.

Player.InvulnerabilityMode string

The secondary effect an invulnerability powerup will have on the player, in addition to the invulnerability and display gimmick.

Ghost will make the player phase in and out of ghost mode rapidly; Reflective will make the player reflect projectile attacks.

Default is unnamed. To obtain the default effect, just make sure the line is not present.

Player.JumpZ value (fixed point)

The player's jump speed. The player's jump height in normal gravity is $(\text{value} \times (\text{value} + 1)) / 2$. A player can still climb their MaxStepHeight while jumping, allowing to reach heights higher than indicated purely by the jump height. For example with the default values, a player can climb a height of up to 60 map units: 36 from jump, 24 from step.

Default is 8.0.

Player.MaxHealth value

Default is 100.

The maximum health reachable by using normal health items.

Player.MorphWeapon weapon

Sets the weapon the player will use when morphed to this class.

Player.MugShotMaxHealth value

Defines the value that is considered to be max health for the status bar mugshot. Negative values use the player's max health as the mug shot max health, zero (default value) uses 100 as the mug shot max health, and positive values are used directly as the mug shot max health.

Player.Portrait string

Replaces the player portrait in the class selection menu to show a customized image, much like Hexen's player classes.

Player.RunHealth value

The minimum health the player can have and still be able to run. (This is normally 16% for Strife)

Default is 0.

Player.ScoreIcon sprite

The icon visible next to the player's name on multiplayer scoreboard.

Player.SideMove value [value-run]

Speed modifier for left/right movement. The value is for walking and value-run is for running.

Default is 1 for both values. The default run speed is double the walking speed, so Player.SideMove 1, 0.5 would disable side-running.

NOTE: Side movement running speed is actually 4/5ths forward running speed, and the side walk speed is 24/25ths forward walk.

Player.SoundClass string

The sound class used by SNDINFO's \$playersound command.

Default is "player".

Player.SpawnClass spawnclass

This is the filter for spawning things in a map. The spawnclass can be one of Any, Fighter, Cleric, Mage or a number between 1 and 16. 1 is synonymous with Fighter, 2 with Cleric and 3 with Mage.

Player.StartItem classname [amount]

Adds an item to player's start inventory. First weapon added is the weapon selected at start.

Note: The initial startitem list is never inherited and must be specified in full for each player class.

Player.TeleportFreezeTime value

The time in tics the player remains immobile after teleportation. If this is 0 or less, or the player has at least one active powerup which has the INVENTORY.NOTELEPORTFREEZE flag set, this behavior is overridden, and as a result, the player does not become immobile after teleportation.

Default is 18.

Player.UseRange range

Determines how far away a player class can activate lines or objects with the +use command.

Default is 64.0.

Player.ViewBob value

A multiplier for move and still bob.

Default is 1.0.

Player.ViewHeight value

Specifies how high above the floor the player's eyes are.

Default is 41.0.

Player.WaterClimbSpeed value (development version 73159da only)

The speed at which the player can climb walls when swimming.

Default is 3.5.

Player.WeaponSlot slot, weapon1[, weapon2, weapon3, ...]

Assigns the specified weapons to the given slot number for this player class. slot can be any single digit from 0 - 9. This is followed by a comma-separated list of each weapon's class name that should be assigned to that slot. If there is more than one weapon assigned to a given slot, the player can cycle between them by repeatedly pressing the slot button. For other ways to assign weapons to slots, see weapon slots.

Powerup

Powerup.Color color [, alpha]

Powerup.Color BlueMap

Powerup.Color GoldMap

Powerup.Color GreenMap

Powerup.Color InverseMap

Powerup.Color RedMap

Powerup.Color None

Specifies the color and the opacity of the screen blend that is used when the powerup is active. Color is specified either as three integers (DECORATE only), an "RR GG BB" string or a color name from the X11R6RGB lump. Alpha is optional, and is a value between 0.0 and 1.0, defaulting to 0.33333 if not specified. Instead of specifying a color you can also use the predefined values BlueMap, GoldMap, GreenMap, InverseMap and RedMap. These do not set a blending value but instead use a predefined blend. Using a color value of None for a PowerupGiver, results in having no screen blend. Setting an arbitrary color with zero alpha, instead, to override the powerup's own does not achieve the same task.

Powerup.Colormap [sourcecolor,]destcolor

A generalization of the colormaps (which, counter-intuitively, are used with Powerup.Color), this creates a color ramp from the source color to the destination color. This uses decimal numbers for the red, green, blue components, ranging from 0.0 to 1.0, inclusive. If only one color is provided, then black (0.0, 0.0, 0.0) is used as the source color. For example, Powerup.Colormap 0.0, 0.0, 0.0, 1.0, 1.0, 1.0 and Powerup.Colormap 1.0, 1.0, 1.0 both create a grayscale, Powerup.Colormap 1.0, 1.0, 1.0, 0.0, 0.0, 0.0 creates the well known inverse map and Powerup.Colormap 1.0, 0.0, 0.0, 0.0, 1.0, 0.0 creates a red to green effect.

Powerup.Duration time

The length of time the power up will last in tics (1/35 of a second). If a negative duration is given, it will be taken as the duration in seconds (e.g., -30 is half a minute, while 30 is less than a second). The maximum duration is 0x7FFFFFFF, which corresponds to about two years of real time and can therefore safely be used for permanent powerups. Note, however, that due to the design of the TimeFreezer powerup, using such a duration value for this powerup is not recommended, as the engine will add two tics to the powerup's duration if the duration's value was even and the powerup was activated on an odd game tic, resulting in a signed overflow error. A value of 0x7FFFFFFD is safe to use in such a case.

Powerup.Mode mode

For some powerups which may behave in different ways, specify the mode. Currently affects Invulnerability, Invisibility and Environment protection.

Invulnerability modes: None and Reflective. "None" is invulnerability without any added effect. "Reflective" also makes all projectiles bounce off of the player while the powerup is in effect.

Invisibility modes: Cumulative, Fuzzy, Opaque, Stencil and Translucent. "Cumulative" allow multiple invisibility powerups of the same type to stack when active at the same time, adding their strength value together. "Fuzzy" uses the fuzz effect renderstyle, like that of the spectre. "Opaque" results in an opaque renderstyle, making the actor not invisible graphically speaking. "Stencil" uses the stencil renderstyle, showing just the shape. "Translucent" is merely a way of explicitly specifying the default invisibility mode.

Environment protection modes: Normal and Full. "Normal" is environment protection's default mode. "Full" is total-protection mode. It is similar to "Normal", but it also protects against the types of damaging floors that the "Normal" mode does not protect against, whether it is partial protection or not.

Powerup.Strength strength

The magnitude of a powerup's effect. Currently applies to Invisibility and Regeneration, as other powerups have unquantifiable effects or use predetermined fields such as DamageFactor.

Invisibility strength: determines how translucent the affected actor becomes. 0 is for opaque, and 100 is for fully invisible. Does not apply to "Fuzzy" mode.

Regeneration strength: determines the amount of health regenerated.

PowerSpeed

The PowerSpeed base class defines the following properties which are available to all subclasses:

PowerSpeed.NoTrail value

If set to 1, it disables the speed trail effect that is created while the player is moving.

Default is 0.

PowerupGiver

All the Powerup properties, which they override, and in addition the following ones:

Powerup.Type power_name

The type of powerup given when item is used. This must be the name of an existing powerup. In DECORATE only, when passing the powerup's name, the "Power" prefix is optional and may be left out.

HealthPickup

HealthPickup.AutoUse value

Determines whether (and how) the item is set to be automatically used when the player's health drops below a certain point.

The following values are accepted:

0 - Item will never be auto-used.

1 - Item will be automatically used when the player would die from damage taken, but only if the currently-selected skill level has the "autousehealth" flag set. (Like the Quartz Flask from Heretic/Hexen)

2 - Item will be automatically used when the player would die from damage taken, but only if the currently-selected skill level has the "autousehealth" flag set or during a deathmatch game. (Like the Mystic Urn from Heretic/Hexen)

3 - Item is auto-used when the player drops below 50% health; if the player has more than 1 such item, they will continue to be consumed until the player is over 50% health. However, if damage takes the player below 0% health, this will not save them from death. (Like Strife's small health items).

MorphProjectile

MorphProjectile.PlayerClass classname

Defines what class to morph players into.

MorphProjectile.MonsterClass classname

Defines what class to morph monsters into.

MorphProjectile.Duration tics

Defines the time that the victim stays morphed.

MorphProjectile.MorphStyle flags

Defines the behaviour of the morphing effects. Use | to use multiple flags at once. Omitting this property or specifying it as zero causes the default Heretic/Hexen behaviour to be applied; see the notes below for more information. These flags only affect morphed players (not morphed monsters) except where explicitly stated in a flag's description.

MRF_ADDSTAMINA — The current stamina is added to the maximum health that is effect (see

MRF_FULLHEALTH) whenever the player receives health via normal means. This is normal under-the-hood behaviour that is by default suppressed when morphed. You would normally leave it off for a morph curse and turn it on for a morph powerup. Note that by default, only Strife games define a stamina mechanism. MRF_FULLHEALTH — The morphed player's maximum health is the MaxHealth of the morphed player class (instead of hardcoded to 30).

MRF_UNDOBYTOMEOPPOWER — A morphed player unmorphs when they activate a Tome of Power.

MRF_UNDOBYCHAOSDEVICE — A morphed player unmorphs when they activate a Chaos Device.

MRF_FAILNOTELEFRAG — A morphed player stays morphed if unmorph by Tome of Power fails, instead of being telefragged.

MRF_FAILNOLAUGH — A morphed player doesn't laugh if unmorph by Chaos Device fails.

MRF_WHENINVULNERABLE — A player can morph when invulnerable but ONLY if morphing themselves via a morph powerup.

MRF_LOSEACTUALWEAPON — When the player unmorphs, the actual weapon given by the Player.MorphWeapon property is taken away.

MRF_NEWTIDBEHAVIOUR — A morphed actor's TID is by default transferred from the old actor to the new actor. (Applies to players and monsters)

MRF_UNDOBYDEATH — A morphed actor unmorphs when killed and (unless MORPH_UNDOBYDEATHSAVES is also used) stays dead. (Applies to players and monsters)

MRF_UNDOBYDEATHFORCED — A morphed actor having the MRF_UNDOBYDEATH style flag, is guaranteed to unmorph when killed. (Applies to players and monsters)

MRF_UNDOBYDEATHSAVES — A morphed actor having the MRF_UNDOBYDEATH style flag unmorphs normally when killed, instead of dying. (Applies to players and monsters)

MRF_UNDOALWAYS — Certain conditions would prevent an unmorph from occurring when the morph powerup expires, such as not having enough room, so it would continue to attempt unmorphing every second. This disables these conditions entirely and forces them to unmorph no matter what.

MRF_TRANSFERTRANSLATION — Transfers the actor's translation to the morphed actor. (Applies to players and monsters)

MorphProjectile.MorphFlash classname

Defines the effect flash actor to spawn when the victim morphs. If omitted, the game's default teleport fog is used.

MorphProjectile.UnMorphFlash classname

Defines the effect flash actor to spawn when the victim unmorphs. If omitted, the game's default teleport fog is used.

Actor states

An actor's states can be defined within its DECORATE definition. State sequences describe all the behavior of the actor as well as its animation.

A state definition is started with the states keyword and enclosed by braces '{', '}'.

It consists of the following:

State labels

A state label is an identifier followed by a colon (:). State labels give names to state sequences which can then be initiated or checked for using those names.

A state label can be any alphanumeric string (within reason) and is not case sensitive. Some labels (Spawn, See, Death, Ready, Select, Deselect, Fire, etc.) are assumed to exist by the engine for certain actors.

A single state can have several labels, each on a different line. However, most states do not have a label, instead they merely follow other states in sequences.

State definitions

The main elements of any given state are the following:

Its sprite name

Its frame letter

Its duration in tics

Its associated action function

Its successor (the next state in sequence)

It also might have additional properties which are expressed through special keywords detailed below.

These consist of a sprite name, a frame letter, the duration in tics and optionally additional keywords and an action function name (code pointer). For example:

```
STUF C 5 Bright A_Look
```

Here, STUF is the sprite name, C is the frame letter, 5 the duration, and A_Look the action function.

The successor is defined implicitly as the next defined state, unless a goto, loop, wait, or stop keyword is used to explicitly change it. For instance:

```
STUF C 5 Bright A_Look
```

```
STUF D 5 Bright
```

Here, the successor for the first state is the second state. The second state's successor is not defined in this example.

If several states share the same sprite name, duration, keywords, and action functions, and they follow in sequence, the states can be "collapsed" together by stringing the frame letters in a word.

```
STUF ABCD 5 Bright A_Look
```

Here, four different states are defined on a single line. Each of A, B, C, and D are different states. And likewise:

```
STUF VVVVVV 5 Bright A_Look
```

The six V states are all entirely identical (except for their successor), but they are nonetheless separate states.

When the duration runs out, the actor moves to the next state in the sequence and runs the new state's action function immediately. Note that setting -1 as a duration means infinite duration. The actor, once it enters this state, will never leave it on its own; though it can still be moved to a different state by external actions (e.g., suffering damage might put

it in the Pain state).

Random durations are possible with the random(min, max) function. Alternatively, you can use A_SetTics; this allows to use full-fledged DECORATE expressions to set any kind of dynamic duration; but prevents the state from having another action function.

The next state is automatically implied to be the following letter on a frame sequence, or if there aren't any more states on a line, the states defined in the next line. Alternatively, flow control keywords (loop, wait, goto, stop) listed after a state can change it. Jump functions such as A_Jump will ignore normal sequence logic and immediately move to their designated state, without waiting for the duration to run out first.

Variable duration

A state can have a random duration. Instead of defining a frame like this:

```
POSS A 10 A_Look
```

You can define it as:

```
POSS A random(10,20) A_Look
```

and the state will last a random duration between 10 and 20 tics, inclusive. More control can be obtained by using the A_SetTics function and DECORATE expressions.

```
POSS A 0 A_Look
```

```
POSS A 10 A_SetTics((waterlevel + 10) - (accuracy / 10))
```

State keywords

The existing keywords can be used in a state between the duration and the action function call.

Bright

The sprite will be displayed as fullbright while the actor is in this state. Note that this is ignored in fog.

CanRaise

Mark the state as allowing A_VileChase to target the actor provided the other conditions are met. By default, only states with a -1 (infinite) duration are eligible. Also, if a state with this keyword is reached, a monster is eligible for respawning if respawning is enabled.

Fast

The state has its duration halved in fast mode (if using a skill with the FastMonsters property, or the -fast command-line parameter) and for actors with the ALWAYSFAST flag. This has no effect on actors with the NEVERFAST flag.

Light("lightname")

Adds a dynamic light to the state. See below for further information.

NoDelay

Forces the action function associated to the state to be executed during the actor's first tic. This is only useful for the first state in an actor's Spawn sequence.

Offset(x, y)

Gives the state a sprite offset, only used for HUD sprites (most relevant for weapons. Note that Offset(0, 0) is interpreted as "keep previous offset", not as "reset offset to 0, 0" for compatibility with Hexen, which is the game from which this feature originates. More control over offsets can be gained by using A_WeaponOffset and A_OverlayOffset functions instead.

Slow

The state has its duration doubled in slow mode (if using a skill with the SlowMonsters property).

Example

```
POSS AABBCDD 4 A_Chase
```

This defines 8 states. Each one of them uses the sprite POSS, has a duration of 4 and uses the code pointer A_Chase which is the standard walk function for monsters. Of these 8 states the first 2 will use the sprite frame 'A', the next 2 the frame 'B' and so on. The length of the frame sequence can be up to 256 characters. Valid frames are 'A'-'Z', '[', '\ and ']'. Different sprites can be freely mixed in an actor definition; however, each separate line of a state definition is limited to one sprite only.

Notes

If the frames '[', '\ or ']' are used the frame sequence has to be enclosed in quotation marks ("").

Sprite name and frame TNT1 A means no sprite, making the actor invisible for the duration of the state.

It is possible for a state to keep the actor's current sprite and/or frame, using special sprite names such as "----" or "#####". See the sprite page for more information.

Anonymous functions

You can declare an anonymous function by using braces in place of an action function at the end of a state. This allows you to call multiple action functions from a single state. A semicolon is required after each statement.

POSS A 4

```
{
  A_Chase; //<-- Semicolon is REQUIRED when inside of these!
  A_SpawnItemEx("BloodyTrail",0,0,0,0,0,0,0,SXF_NOCHECKPOSITION);
}
```

Return

To terminate an anonymous function early, use return. It can either return a value or not. The type of value returned is used to infer the return type of the anonymous function, so if you have more than one return statement, they must all return the same type. The types are state (including jumping functions), int, bool and float. To jump to a new state, you can return a state, either by specifying it directly or by calling a jump function.

```
{ return state("Null"); } // Destroys the actor. All actors have a Null state by default unless overridden.
{ return state("JumpState"); } // Guarantees a jump.
{ return A_Jump(256, "JumpState"); } // So does this.
{ return state(""); } //Aborts an anonymous function without jumping and plays the rest of the tics.
{ return state(0); } //Same here.
{ return; }
```

In ZScript returns require a pointer acquired via FindState() or ResolveState(). The primary difference is that ResolveState() is context-aware and will work properly when called from PSprite, while FindState() won't. As a result, ResolveState() can be used from Weapon states, where self is interpreted as the weapon's owner. For example:

Fire:

```
TNT1 A 0
{
  if (invoker.ammol.amount < invoker.ammouse1) //check ammo
  {
    return ResolveState("Reload"); //go to Reload sequence if not enough
  }
  return ResolveState(null); //otherwise continue to the next state
}
```

Note that any function that contains a conditional return in it must end with a null return (such as return ResolveState(null)), so that it knows where to go if the condition isn't met.

Note, returning states manually is the only way to create a complex conditional jump, whereas A_Jump* functions won't work for this, since they always jump either to the specified sequence, or to the next state. For example, this will not work:

Fire:

```
TNT1 A 0
{
  if (invoker.ammol.amount < invoker.ammouse1) //check ammo
  {
    return A_Jump(256, "Reload") // jumps to Reload or to next state
  }
}
```

```

        return ResolveState(null); // This will never be reached!
    }

```

As a result, A_Jump* functions should not be used in anonymous functions.

Break

Used in for, do-while and while loops. When encountered, a loop is stopped entirely and the code after the loop will continue executing.

Continue

Used in for, do-while and while loops. When encountered, a loop will restart and skip execution of all commands after the statement.

If and else keywords

Anonymous function blocks support if/else statements to do different things based a condition without the need to use an A_Jump* function to jump to another state.

It is possible to use an A_Jump-type function inside the condition part of the if statement. The jump will not happen, but if it would have, the condition is considered as true.

NOTE: All jump functions used as the condition check for an if statement need to be passed a valid state label, or the function will always return false for that condition. "Null" is a valid state that can be used instead of needing to define one, as the jump is never performed.

POSS A 4

```

{
    if (CheckClass("PlayerPawn",AAPTR_TARGET,true))
    {
        //A_Warp performs what it's meant to do, but will never jump.
        //The same applies to all functionality that can jump -- they do nothing here.
    }
}

```

```

A_Warp(AAPTR_TARGET,x,y,z,0,WARPF_NOCHECKPOSITION|WARPF_USECALLERANGLE|WARPF_ABSO
LUTEPOSITION|WARPF_COPYVELOCITY,"Enrage");

```

```

    A_Jump(256,"Enrage"); //<-- This does nothing
    user_enrage = 1;
}

```

```

else
{

```

```

    user_enrage = 0;
    if (user_cooldown != 0)
    { // This returns from the function, skipping the rest of it.
        return;
    }
    user_cooldown = 100;
}

```

// Using a jump function to check a condition instead of jumping

```

if (CountInv("Cell") > 0)
{
    // We still have cell ammo!
}
else if (CountInv("RocketAmmo") > 0)
{
    // We still have rocket ammo!
}

```



```

    }
    else
    {
        // We're out of ammo!
    }
}

```

For keyword

For loops will repeatedly execute code within until a user variable reaches its destined amount, or it encounters a return/continue/break keyword. The syntax of a for loop is very much like C++.

TNT1 A 1

```

{
    for (user_i = 0; user_i < 15; user_i++)
    {
        //...stuff to execute in here. This is all done in an instant.
    }
}

```

While keyword

While loops will repeatedly execute code until the condition is met. Unlike for loops, the while syntax only has a condition and does not auto increment. The condition is checked before the execution of the code. The syntax of a while loop is very much like C++.

TNT1 A 1

```

{
    while (user_i < 30)
    {
        //...stuff.
        user_i++;
    }
}

```

Do/While keyword

Do-while loops will repeatedly execute code until the condition is met. Unlike for loops, the do-while syntax only has a condition and does not auto increment. The condition is checked after the execution of the code. The syntax of a do-while loop is very much like C++.

TNT1 A 1

```

{
    do
    {
        //...stuff.
        user_i++;
    } while (user_i < 30); //Semicolon here is important.
}

```

Flow control

There are 5 different instructions that control the execution order of an actor's frames directly:

loop

Jumps to the most recently defined state label. This is used for a looping animation. Do not put a loop on a state with a duration of -1, this is unnecessary and can cause problems.

stop

Stops animating this actor. Normally this is used at the end of the death sequences. If the last state has a duration > -1 the actor will be removed. Note that if a state contains only the stop instruction, the actor will behave as if it doesn't have that state. This can be useful, for example, to remove a state that an actor has inherited from its parent.

wait

Loops the last defined state. This is useful if a code pointer is used that waits a given time or for a certain event. Currently useful code pointers include A_WeaponReady, A_Raise, A_FreezeDeathChunks, and similar code pointer functionality.

fail

Used with custom inventory items, means that the state sequence failed to succeed.

goto label [+offset]

Jumps to an arbitrary state in the current actor. With this, you can also jump to a base class state, i.e. one that was inherited by a parent. The statement goto See jumps to the walking animation that has been overridden in the current actor class, or if such does not exist, to the inherited class's state. goto is however static, i.e. will not do virtual jumps â€” for that, see A_Jump.

Offset specifies the number of frames to skip after the specified label. That is, using "goto Spawn+2" will jump to this frame:

Spawn:

```
TNT1 AAAAAAAA 0
```

^

In addition, if an actor is using inheritance, you can use goto with the scope operator (::) to go to a parent class' state. The keyword "super" always refers to the immediate parent, but any parent class can be referred to by name as well, for example "goto Actor::GenericFreezeDeath" is a valid instruction.

Important note

This format has been designed for maximum flexibility. As a result no assumptions are made about what the designer wants. States are never implicitly created.

Also, if no flow control is used ZDoom will continue to the state provided directly after. Consecutive state labels can be used to assign the same frames to more than one state.

States

These are the predefined states each actor has access to:

Spawn

Defines the state that is displayed when an actor is spawned. For monsters this is normally also the idle loop.

Also entered by hitscan puffs when hitting a non-bleeding actor, provided the puff has +PUFFONACTORS.

Note: An actor that has just been spawned does not run the action function attach to the first frame of its Spawn sequence. For example, if a monster calls A_Look in its first frame, it won't look for enemies until the Spawn sequence is looped once. If you need to force an actor to call the function instantly upon entering the Spawn sequence, add NoDelay before the function call.

Idle

Defines an alternate idle state for a monster to return to when it has run out of targets. If this state is not present, the monster will return to the Spawn state instead.

See

Defines the walking animation for a monster. Note that this state must be present for actors which are able to chase and attack other actors.

Melee

Defines the melee (close-range) attack.

Also entered by puffs when hitting an enemy with a melee attack, such as A_CustomPunch, provided the puff has +PUFFONACTORS.

Missile

Defines the missile (ranged) attack.

Pain

Defines the pain action. Multiple Pain states can be used depending on the type of damage inflicted. See custom damage types.

Death

Defines the normal death sequence. Multiple Death states can be used depending on the type of damage that kills the actor. See custom damage types. Also entered by projectiles when hitting a wall (or an actor as well if the Crash and/or

XDeath states are not defined).

Death.Sky

Defines an alternate death sequence for projectiles. This is entered when hitting a sky plane while having the SKYEXPLODE flag set. Does not work on fast projectiles.

Death.Extreme

XDeath (internally, Death.Extreme)

Defines the extreme (splatter) death sequence (if the actor's health is lower than its GibHealth). Multiple XDeath states can be used depending on the type of damage that kills the actor.

Also entered by projectiles when hitting a bleeding actor (if no XDeath state is defined, they enter their Death state instead).

Also entered by puffs when hitting a bleeding actor, provided the puff has +PUFFONACTORS. (If the actor is non-bleeding or this state isn't defined, Spawn is used instead).

Error.gif

Warning: For monsters that disappear with their death states, always ensure there is at least a 1 tic delay before the stop of the actor. A VM abort can potentially happen otherwise.

Death.Fire

Burn (internally, Death.Fire)

Defines the burn (Fire) death sequence.

Death.Ice

Ice (internally, Death.Ice)

Defines the freeze (Ice) death sequence.

Death.Disintegrate

Disintegrate (internally, Death.Disintegrate)

Defines the disintegration death sequence.

Raise

Defines the resurrection sequence.

note: This is when a monster is being resurrected (ie: by an Arch-Vile), not when its resurrecting another monster.

Heal

Define the healing sequence. This is entered when this monster is resurrecting another one. Note that by the time this monster enters this state, the resurrection process has already started. The process is usually started either by A_Chase, with the CHF_RESURRECT flag passed to it, A_VileChase, or A_CheckForResurrection.

Crash

Defines the crash sequence. Multiple Crash states can be used depending on the type of damage that kills the actor. This is entered when the actor is a corpse and hits the floor.

Also entered by projectiles when hitting a non-bleeding actor (if no Crash state is defined, they enter their Death state instead).

Also entered by puffs when the attack hits a wall/plane (but for non-bleeding actors puffs use Spawn, as opposed to projectiles).

Crash.Extreme

Defines the extreme (splatter) crash sequence. Multiple Crash.Extreme states can be used depending on the type of damage that kills the actor. This is entered when the actor is a corpse and hits the floor after being gibbed.

Crush

Defines the crush sequence. This is entered when the actor is crushed by ceiling/door/etc.

Wound

This state is entered when the actor is damaged and its health is lower than its WoundHealth but greater than 0. Multiple Wound states can be used depending on what type of damage is inflicted upon the actor. See custom damage types.

Slam (New from 4.10.0)

This state is entered if an actor with SKULLFLY hits another actor. This takes precedence over entering the Spawn or Idle states, or entering the See state if RETARGETAFTERSLAM is on.

Slide (New from 4.10.0)

This state is entered if an actor with PUSHABLE is successfully pushed by another actor.

Greetings

This is used by the Strife dialog system. It is entered when a conversation is about to start.

Yes

This is used by the Strife dialog system. It is entered when the actor reacts positively to the player's choice.

No

This is used by the Strife dialog system. It is entered when the actor reacts negatively to the player's choice.

Active

This is used by Hexen-style switchable decorations. It is entered when the actor is activated.

Inactive

This is used by Hexen-style switchable decorations. It is entered when the actor is deactivated.

Bounce

This is used by bouncers with the USEBOUNCESTATE flag. It is entered when the actor bounces. Multiple bounce states can be used depending on what the actor bounced off:

Bounce

Bounce.Floor

Bounce.Ceiling

Bounce.Wall

Bounce.Actor

Bounce.Actor.Creature

Partial matches work just like Pain states, so if an actor bounces off a floor and you don't have a Bounce.Floor state, but you do have a Bounce state, it will use the Bounce state. Conversely, if you only have a Bounce.Floor state but no Bounce state, then the actor will only enter the Bounce.Floor state when it bounces on a floor; bouncing off anything else will not cause it to change state. The Bounce.Actor.Creature state is used for bouncing over a shootable actor without the NOBLOOD flag.

Weapons and custom inventory items define a few more states to define their animations.

Note that you can also define your own states that can be referred to using A_Jump or other jump instructions.

Examples

This is an example of a state sequence. The rest of this actor has been removed for readability:

actor ZombieMan 3004

```
{
  ...
  states
  {
    Spawn:
      POSS AB 10 A_Look
      loop
    See:
      POSS AABBCDD 4 A_Chase
      loop
    Missile:
      POSS E 10 A_FaceTarget
      POSS F 8 A_PosAttack
      POSS E 8
      goto See
    Pain:
      POSS G 3
      POSS G 3 A_Pain
      goto See
    Death:
```

```

    POSS H 5
    POSS I 5 A_Scream
    POSS J 5 A_Fall
    POSS K 5
    POSS L -1
    stop
XDeath:
    POSS M 5
    POSS N 5 A_XScream
    POSS O 5 A_Fall
    POSS PQRST 5
    POSS U -1
    stop
Raise:
    POSS KJIH 5
    goto See
}
}

```

Note: The first frame of the “Spawn” state, “POSS A 10”, calls A_Look. This function is not called the very first time the zombie is spawned in the map, so it has to wait 10 tics to get into its second frame, “POSS B 10”. From then on, it will call A_Look every 10 tics. If it runs out of targets, and since it has no “Idle” state, it will return to its Spawn state where it will call A_Look immediately, even in the A frame.

This example demonstrates using the stop keyword to remove a state. This definition uses inheritance to define a tougher version of the imp that cannot be resurrected by the Arch-Vile:

```

actor SuperImp : DoomImp
{
    health 1500
    mass 200
    painchance 10

    States {
        Raise:
            stop
    }
}

```

DECORATE expressions

DECORATE supports complex mathematical expressions as parameters for codepointers. (Unfortunately, expressions are not supported for the values of properties.) The expression may include standard operators (+, -, *, /, <<, >>, |, ? and : (ternary) etc.), math functions, and certain actor properties (occasionally called "keywords" in the forums) to compare values with. A_JumpIf, in particular, is meant to be used with expressions, but they can be used for any numeric (integer or floating point) parameter. For example, using velx, vely, and velz in A_SpawnItemEx to preserve velocity, or using args[] as arguments to a call of ACS_Execute.

Mathematical functions

abs(x)

Returns the absolute value of x.

exp(x)

Returns the base-e exponential function of x, which is e raised to the power x: e^x .

log(x)

Returns the natural logarithm of x -- the opposite of exp.

log10(x)

Returns the common (base-10) logarithm of x.

ceil(x)

Rounds x upward to the next closest integral value. Result is still floating point.

floor(x)

Rounds x down to the next closest integral value. Result is still floating point.

round(x)

Rounds x to the closest integral value. Result is still floating point.

sqrt(x)

Returns the square root of x.

min(float or int, ...)

Gets the smallest value of all values listed. Can take any amount of numbers, and can solve both ints and floats.

max(float or int, ...)

Gets the largest value of all values listed. Can take any amount of numbers, and can solve both ints and floats.

clamp(src, min, max)

Returns src within the range of min and max inclusively. All parameters can be ints or floats.

Trigonometry functions

cos(x)

sin(x)

tan(x)

Returns cosine, sine, or tangent of angle x. x must be in degrees.

acos(x)

asin(x)

atan(x)

Returns the inverse cosine, sine, or tangent of x. Returns an angle in degrees.

cosh(x)

sinh(x)

tanh(x)

Returns the hyperbolic cosine, sine, or tangent of x.

atan2(y, x)

Similar to atan, but used to get the quadrants the angles are in. Returns in degrees.

VectorAngle(x, y)

Like atan2, only with x first instead of y. Can be used to calculate an angle, for example, by passing coordinates to it.

VectorAngle(user_x[0] - user_x[1], user_y[0] - user_y[1])

Random number functions

random[identifier](min, max)

Returns a random integer value between min and max.

random2[identifier](mask)

Returns a random integer value between -mask and +mask. It is equivalent to (random() & mask) - (random() & mask). If no mask is provided, i.e, random2(), the maximum value of 255 is used instead. Mask is used as a binary mask, e.g. if 9 is used, the random results can be [0, 1, 8, 9] - [0, 1, 8, 9], so it is advised to use as a mask values one less than a power of two, such as 1, 3, 7, 15, 31, 63, or 127.

frandom[identifier](min, max)

Returns a random floating point value between min and max.

randompick[identifier](int, ...)

Picks a number from the numbers placed in it. This can take an unlimited amount of parameters, i.e.

randompick(1, 4, 12, 16) will choose one of the four numbers.

frandompick[identifier](float, ...)

Similar to randompick but for float-point values.

SetRandomSeed[identifier](seedvalue)

Used if a guaranteed sequence of random numbers is desired. When a specific number is set as seed, the random-number generator (RNG) will always produce the same numbers. Useful for cases where an RNG may be used for purposes where total randomness is not wanted.

identifier is optional. Calls to a random function with an identifier do not interfere with the RNG for calls with a different identifier, which can be used to make the outcome of some events unaffected by others.

ACS functions

ACS_ExecuteWithResult

Runs a script and uses its return value.

ACS_NamedExecuteWithResult

Runs a named script and uses its return value.

CallACS

Shorter alias for ACS_NamedExecuteWithResult.

DECORATE functions

These functions are not like regular DECORATE action functions. They can be used just like expressions (i.e. randompick) inside of action functions directly such as A_SetUserVar.

CheckClass (deprecated)

Checks an actor's class name, returning true if it matches the passed parameter.

CountInv

Gets the number of inventory items an actor is holding from a pointer and returns it to the expression.

CountProximity

Similar to A_CheckProximity but returns how many of the actors are found instead of jumping.

GetAngle

Gets the angle which an actor pointer can be found at. Can toggle the relative functionality

GetCvar

Gets a server cvar, internal or user.

GetCrouchFactor

Gets the current crouch factor of a player if the player is a pointer.

GetDistance

Gets the distance from the calling actor to a pointer. Can toggle the Z checking functionality.

GetGibHealth

Gets the gib health property of the calling actor.

GetMissileDamage

Gets the damage of the actor.

GetPlayerInput

Performs the same functionality as the ACS GetPlayerInput.

GetSpawnHealth

Gets the spawn health property of the calling actor.

GetSpriteAngle

Gets an Actor pointer's sprite angle.

GetSpriteRotation

Gets an Actor pointer's sprite rotation.

GetZAt

Gets the Z coordinate of a floor or ceiling at a specific set of coordinates.

IsPointerEqual

Compares two actor pointers, returning true if they reference the same actor.

OverlayID()

When used in an overlay, it retrieves the overlay's layer index.

OverlayX

Returns the x position of the specified overlay.

OverlayY

Returns the y position of the specified overlay.

SetDamage

Sets the actor's damage.

Variables

As of GZDoom 2.2.0, it is possible to directly modify some of the variables with assignment operators.

Bold - Read and write variables. Allows direct assignment operators.

Italics - Read-only variables.

Read-only variables must still rely upon the action function used to manipulate them (e.g. *A_Warp* for x/y/z, *A_ChangeVelocity* for velx/vely/velz). As of GZDoom 2.2.0, unary increment/decrement operators (user_test++, --user_test, etc.) may be used, as well.

NOTE: Direct variable modifications will not be interpolated or have any other effects performed upon them. If these effects are desired, use the action function associated with the variable(s) instead (such as *A_SetAngle* for interpolating angle changes).

The following is dynamic data for use in DECORATE definitions and expressions:

Actor position and movement

x — The actor's X position in the world.

y — Same, but for Y.

z — Same, but for Z.

angle — Actor's angle, in degrees

ceilingz — returns Z height of the ceiling above the calling actor as an absolute double value (portal- and 3D-floor-aware)

floorz — returns Z height of the floor underneath the calling actor as an absolute double value (portal- and 3D-floor-aware)

pitch — The actor's pitch in degrees.

roll - The actor's roll. Currently, this only really affects players such as the rotation of view.

velx or momx — Actor's velocity along the absolute X axis. The "mom" names are (deprecated).

vely or momy — Same, but for Y.

velz or momz — Same, but for Z.

Actor properties

accuracy — The accuracy of the Actor.

alpha — The Alpha value of the Actor.

args[] — Arguments passed to the thing special; args[0] through args[4] are valid.

damage — The actor's Damage. (deprecated)

health — How much health the Actor has left.

height — The actor's Height.

mass — The actor's Mass.

meleerange — The actor's MeleeRange.

radius — The actor's Radius.

reactiontime — The actor's ReactionTime.

scaleX — The actor's horizontal scale. See *A_SetScale*.

scaleY — The actor's vertical scale. See *A_SetScale*.

score — The actor's score.

special — ID of the special currently assigned to this actor.

speed — The actor's Speed.

stamina — The stamina of the Actor.

tid — The actor's TID.

tidtohate — TID of the current assigned target. (see Thing_Hate.)

threshold — The actor's Threshold.

defthreshold — The actor's DefThreshold.

VisibleStartAngle/VisibleEndAngle - The actor's angle range which the camera must see it to draw the sprite.

Requires MASKROTATION flag to have any effect.

VisibleStartPitch/VisibleEndPitch - Same as VisibleStartAngle and VisibleEndAngle but for pitch instead.

Constant Variables — Constant variables can be defined both inside and outside of an actor.

User Variables — user variables are defined as "var int user_(name);" in actor properties.

User Arrays — user arrays are defined as "var int user_(name)[(size)];" in actor properties.

waterlevel — How "submerged" the actor is in a Transfer_Heights or 3D floor water pool:

0: Not submerged at all (e.g. standing on solid ground or on shallow TERRAIN-based water)

1: Less than half submerged ("ankle deep")

2: At least half submerged ("waist deep")

3: Entirely submerged (completely underwater)

waterdepth — The exact depth that the actor is at inside of a Transfer_Heights or 3D floor water volume.

Returns an integer. Can be used for more detailed depth checks than waterlevel.

Advanced DECORATE functions guide

Overview

This guide will go into some potentially confusing details about how to use some of the newer and/or advanced functions. It will also demonstrate some complex uses for the functions as well.

It is recommended to read more about Actor pointers first to help understand more about the vast majority of functions covered in this guide.

Be sure to also read up on the functions themselves before continuing with this guide.

A_Warp

A_TransferPointer

A_RearrangePointers

A_CheckFlag

A_SetUserVar

A_SetUserArray

A_Warp

The function A_Warp acts similarly to A_Fire. The difference is extreme to the point where, with a little smart thinking, anyone can utilize it to perform other capabilities as well.

Uses:

Laser-like effects (Real instant beams, NOT fast projectiles!)

Attachment-like actors capable of sticking to other actors

Satellite rotating abilities (can be combined effectively with A_Weave if done correctly)

Can attach to just about anything

Checking for empty space to see if it can perform a certain ability or not

Self-moving ability

A_TransferPointer

This is direct manipulation of pointers down to the core. This, along with A_RearrangePointers, it is possible to change relationships of any actor with other actors inside of it's "scope".

This function in particular can allow transferring code pointers from itself or from other actors to itself or to other actors. This can get the target of a target, or even set the target of another target. It may seem confusing at first, but once you understand just how Actor pointers work, it will usually open up a wide load of possibilities. With this function, for example, one could make a missile which drains health from a monster, and then returns to the shooter, giving it the health in return.

Uses:

Behavior modification

Alignments

Master/Target/Tracer manipulations

A_RearrangePointers

This function is limited to only modifying the calling actor's pointers, but allows for copying. Say a monster wants to remember another monster later, it could use

A_RearrangePointers(AAPTR_DEFAULT,AAPTR_DEFAULT,AAPTR_TARGET). This stores the target in the tracer field, which is not used in monsters. This allows for easy retrieval later on, just by performing

A_RearrangePointers(AAPTR_TRACER,AAPTR_DEFAULT,AAPTR_DEFAULT).

NOTE: It is important to keep in mind that when rearranging a pointer through copying such as the first function above, that target is now also the tracer. If the original monster changes course, the AAPTR_TRACER must be specified

because that's where the monster is for retrieval. This is because the target field will change, so putting `A_RearrangePointers(AAPTR_TARGET,AAPTR_DEFAULT,AAPTR_DEFAULT)` will have absolutely no effect.

Bear in mind, `AAPTR_DEFAULT` means no change, and it will keep the same pointer, which allows for copying.

`A_CheckFlag`

With this function it is now possible to check flags on other actors. This means we can use it to check if the target/master/tracer has a certain flag enabled on them. It performs a jump if they have the flag, and doesn't jump if they don't. By default, it checks the calling actor.

`A_SetUserVar/Array`

Since its introduction, user variables and arrays have been an extreme benefit to the community. This alone allows for reduction of bloat code (copy/pasted lines with many angle changes) except for weapons and players. Those two are the only things which cannot hold user variables or arrays due to their complex nature.

User variables and arrays are even capable of working side along with ACS scripts such as to retrieve a variable of a console setting.

Ultimately, they are capable of modifying and/or retrieving other user variables through CustomInventory actors. Bear in mind, the actor that is being manipulated also must have the same `user_var`, otherwise you will get a warning about an unknown variable attempting to be manipulated by the actor who receives the special CustomInventory actor.

Uses:

Storage and Calculation

Tracking and Determination

Converting Sprites

Converting Sprites into other formats is a great help.

Say you want to have Hexen's Korax Or Doom's Imp but you want to have them in Heretic you can use a tool known as DoomPicDump to convert sprites quite easily.

To use this tool first you must get sprites from a wad (e.g Doom.wad) when you have exported them using your wad or lump management tool (preferably SLumpEd) but have them exported into two formats .imp and .png.

Then boot up DoomPicDump.

Picture Of The Program

Navigate to your sprite folder select and modify the input to match the extensions e.g if you saved the files as .png change it to .PNG.

Then modify any settings you want and click convert all the newly created png files are ready to be imported to your wad.

Creating decorations that can be (de)activated

You can also define objects that change their state when being activated or deactivated.

For this purpose, two base classes exist: SwitchableDecoration and SwitchingDecoration.

Switchable decoration can be activated and deactivated; whereas switching decorations can only be activated but not deactivated again.

These classes use two special state labels: Active and Inactive. You must of course include a spawn state in addition to these two special states to have the actor work properly when spawned.

Activation and deactivation can be controlled by the action specials Thing_Activate (130) and Thing_Deactivate (131). In addition, the Activation property and the BUMPSPECIAL and USESPECIAL flags can be used to make it possible to control how the actor is activated even without ACS.

An example: this torch starts unlit (TRCHA0), but if the player presses "use" in front of it, it is lighted (TRCHB0). With a switchable decoration and the THINGSPEC_Switch activation flag as well, then it would be possible to make a torch that can be lit and unlit several times.

Actor Torch : SwitchingDecoration 10242

```
{
  Height 40
  Radius 20
  Activation THINGSPEC_Activate
  +SOLID
  +USESPECIAL
  States
  {
    Spawn:
      TRCH A 10
      Loop
    Active:
      TRCH B 10 A_PlaySound("torch/start")
      TRCH B 10
      Wait
  }
}
```

Creating new inventory items

Inventory items are special actors that can be picked up and/or placed into a player's or monster's inventory.

To define an inventory item you have to inherit from one of the predefined inventory classes:

Inventory - used for items without special function.

PowerupGiver - used for items that give a special powerup to the player

Health - used for items that immediately increase the player's health

HealthPickup - used for items that are placed in the inventory and give health when being used.

BasicArmorPickup - used for armor items

BasicArmorBonus - used for items that increase the amount of armor

Ammo - used for ammunition items

Weapon - used for weapons

Key - used for keys

PuzzleItem - used for Puzzle items.

WeaponPiece - used for weapon pieces. These are used to create weapons that have to be put together from more than one item.

MapRevealer - used to give the player the automap

Backpack - used to increase the player's ammo carrying capacity.

CustomInventory - used to define items with simple DECORATE based behavior.

FakeInventory - used to replicate simple pickups like the ones defined with the old DECORATE definitions.

Creating new monsters or other complex items

Monsters are just regular actors that call some specific monster AI action functions:

A_Look
A_Look2
A_Chase
A_FastChase
A_VileChase
A_ExtChase
A_Wander

It's these functions that make an actor act like a monster but there's a few more things to observe:

Use the Monster property. This sets up all the necessary flags to make an actor act like a monster.

A monster's height and radius should be set properly. Unlike decorations it is essential that these values are correct because they are used by the collision detection code which also checks whether a monster has been hit by a weapon.

A monster should have proper definitions for all standard sounds or it might remain silent in certain situations.

A monster requires a minimum set of states:

Spawn: This should define a looping 'idle' sequence. This sequence has to call A_Look or A_Look2 repeatedly so that the monster can react to players.

See: This defines a looping walking sequence. This sequence has to call A_Chase or one of its variants repeatedly so that the monster can walk around and do things.

Melee/Missile: This defines a near or far attack sequence. At least one of them is needed so that the monster is able to attack players or other monsters

Death: This is called when the monster's health goes below zero.

You can also define one of the optional special death sequences, a Pain state which makes the monster react to being attacked or the Raise state which makes it resurrectable by Arch-Viles or other monsters capable of doing that.

This is an example of a properly defined monster. It is just an exact replica of Doom's pistol shooting Zombie:

```
actor ZombieClone 3004
{
    SpawnID 4
    Obituary "%o was killed by a zombieman."
    Health 20
    Radius 20
    Height 56
    Mass 100
    Speed 8
    PainChance 200
    SeeSound "grunt/sight"
    AttackSound "grunt/attack"
    PainSound "grunt/pain"
    DeathSound "grunt/death"
    ActiveSound "grunt/active"
    DropItem "Clip" 256
    Monster
    +FLOORCLIP
    States
    {
    Spawn:
        POSS AB 10 A_Look
        loop
```

```

See:
    POSS AABBCDD 4 A_Chase
    loop
Missile:
    POSS E 10 A_FaceTarget
    POSS F 8 A_PosAttack
    POSS E 8
    goto See
Pain:
    POSS G 3
    POSS G 3 A_Pain
    goto See
Death:
    POSS H 5
    POSS I 5 A_Scream
    POSS J 5 A_NoBlocking
    POSS K 5
    POSS L -1
    stop
XDeath:
    POSS M 5
    POSS N 5 A_XScream
    POSS O 5 A_NoBlocking
    POSS PQRST 5
    POSS U -1
    stop
Raise:
    POSS KJIH 5
    goto See
}
}

```

If you don't need a complete monster but only a subset of its functions you can do so. For example to create a shootable item that doesn't act as a monster all you have to do is to remove the See state, the call to A_Look in the spawn state and replace the Monster property with the appropriate flags. The one flag you need is SHOOTABLE but normally you might want to set a few others as well. This is an example of a shootable decoration:

```

actor FloorLamp 10247
{
    Health 1
    Radius 16
    Height 51
    DeathSound "misc/glass"
    +SHOOTABLE
    +SOLID
    +NOBLOOD
    states
    {
    Spawn:
        HAWA A -1 Bright
        Loop
    Death:
        HAWA B 11 A_Scream
        HAWA C 9
    }
}

```



```
    HAWA D -1 A_Fall
  Stop
}
}
```

Other things are possible as well, for example an actor that reacts to seeing a player but instead of waking up and attacking the player it is doing something else, for example playing a sound:

```
actor TriggerSound1 10601
{
  SeeSound "ts1"
  +NOBLOCKMAP
  states
  {
    Spawn:
      TNT1 A 2 A_Look
      Loop
    See:
      TNT1 A 1
      Stop
  }
}
```

Creating new player classes

Choosing player classes in-game

Choosing a player class through the game's GUI can occur in two instances:

- When starting a new game (a class menu will popup inbetween choosing 'new game' and 'skill level')
- When changing the (Multi-)Player Setup in the Options menu

Defining player classes in your PWADs

First thing you have to do is to create new actor in DECORATE inheriting from PlayerPawn or any of its subclasses. It is recommended to inherit from the subclass the closest to what you want to do, for example a replacement of the Doom marine should inherit from the Doom player.

```
actor MyPlayer : DoomPlayer
{
    ...
}
```

Contrarily to simple actor replacement, you cannot use the "replaces" keyword here, because player pawns are not spawned like other actors. Instead, you must define access to this class in MAPINFO, with a GameInfo section:

```
GameInfo
{
    PlayerClasses = "MyPlayer"
}
```

If you have several classes, you can list them all, separated by commas. You can also keep the original classes as well. If there are at least two classes, the player will have a class selection screen when starting a new game.

```
GameInfo
{
    PlayerClasses = "MyPlayer", "DoomPlayer"
}
```

Creating new projectiles

Projectiles are rather simple items. There's only a few things to observe:

Use the Projectile property to set all the necessary flags.

Set a proper height and radius.

Define a looping Spawn state sequence and a terminating Death sequence. The Spawn sequence is shown while the projectile is moving and the Death sequence is the explosion animation.

Projectiles can also use different death states depending on what they hit:

If the projectile hits an actor that has both the SHOOTABLE and NOBLOOD flags set, it will use its Crash state, if present. If there is no Crash state, the XDeath state will be tried next, and finally if that doesn't exist the normal Death state will be used.

If the projectile hits an actor without the NOBLOOD flag (but with the SHOOTABLE flag), it will use the XDeath state. If there is no XDeath state, the normal Death state will be used.

If the projectile hits a wall, floor, or a non-SHOOTABLE actor, the Death state will be used as normal.

This shows a simple projectile:

```
actor BlueShot02
{
    height 8
    radius 6
    damage 5
    speed 10
    renderstyle Add
    alpha 1
    seesound "misc/shot"
    deathsound "misc/shotx"
    PROJECTILE
    +RANDOMIZE
    states
    {
        Spawn:
            WS16 AB 4 bright
            loop
        Death:
            WS16 CDE 6 bright
            stop
    }
}
```

To create a homing missile you have to add the SEEKERMISSILE flag and call one of the seeking code pointers in the Spawn sequence:

```
actor BlueShotSeeking
{
    height 8
    radius 6
    damage 5
    speed 10
    renderstyle Add
    alpha 1
    seesound "misc/shot"
```

```

deathsound "misc/shotx"
PROJECTILE
+SEEKERMISSILE
states
{
Spawn:
    WS16 A 4 bright A_Tracer2
    loop
Death:
    WS16 CDE 6 bright
    stop
}
}

```

You can also do more complex things with Action functions, for example spawning a trail behind the projectile or doing some special effect while exploding. This is a projectile that is doing some BFG-like effect with a different sprite. This projectile also has a limited life span. It automatically explodes after a few seconds because the Spawn state is not looped:

```

actor DevastatorBall
{
    radius 13
    height 8
    speed 25
    damage 200
    renderstyle Add
    alpha 0.9
    deathsound "weapons/bfgx"
    PROJECTILE
    states
    {
    Spawn:
        DVS1 A 100 bright
    Death:
        DVE1 AB 8 bright
        DVE1 C 8 bright A_BFGSpray ("DevastatorExtra")
        DVE1 DEF 8 bright
        stop
    }
}
}

```

Creating new sprite graphics

Jump to navigationJump to search

When creating new sprite graphics, remember the lumpname format for the graphics. First comes the unique 4-letter sprite identifier:

XXXX

Next comes the frame letter, which can range from A-Z. Think of a frame as a single animation step for this creature. Which frame is displayed at any given time depends on the current state of the monster and how you've setup its frames in the DECORATE lump. It is also possible to use the characters [, \, and]; note that if you use them then the frame string referencing them in DECORATE code must be placed within quote marks (").

XXXXA

Each frame must have 1, 8, or 16 rotations. The number following the frame letter does two things: sets this graphic as the specified rotation for the frame, and tells ZDoom how many rotations the frame will have. If you have only 1 (which will look the same from all sides and will seem to "turn" with you as you move around it -- barrels, corpses, etc), you use the number 0 in the single rotation for this frame. If it's going to have 8, you use the numbers 1-8. If it's going to have 16, you still use 1-8, and then use 9, A, B, C, D, E, F, and G, which will display between the other 8 frames as you rotate around the enemy.

XXXXA2

Another feature of Doom (and therefore ZDoom) is automatic mirroring. Using this, a single graphic can be reversed and used for two rotations, saving space in the WAD. To do this, you add an optional extra frame letter and rotation number to the end of the sprite name:

XXXXA2A8

The graphic we just created will display as the XXXX sprite during the A frame when the actor is being seen from its front-left or front-right. For one of those, ZDoom will automatically reverse the graphic.

Mirroring

The optional mirroring allowed by the format is used to reduce the number of graphics needed to complete a full sprite set. This can be seen in all the original IWADs for monsters. The technique should also work fine with 16 rotations, though the following explanations are for the classic 8 rotations.

Take the classic 4-frame (ABCD) walk sequence. In A and B, the monster steps forward with its left foot and comes to rest. In C and D, the monster steps forward with its right foot and comes to rest.

To do this perfectly, you'd need 8 angles each for all four frames, so a total of 32 sprite angles:

A1, A2, A3, A4, A5, A6, A7, A8

B1, B2, B3, B4, B5, B6, B7, B8

C1, C2, C3, C4, C5, C6, C7, C8

D1, D2, D3, D4, D5, D6, D7, D8

By using mirroring on each frame, you can shorten each frame from 8 sprite angles to 5 sprite angles.

A1, A2A8, A3A7, A4A6, A5

B1, B2B8, B3B7, B4B6, B5

C1, C2C8, C3C7, C4C6, C5

D1, D2D8, D3D7, D4D6, D5

This totals 20 sprite frames, saving 12. When the monster becomes mirrored it may shift from a right-foot-first to a left-

foot-first in mid-stride.

The A frames and C are essentially mirrored copies of the monster's movement. If we rearrange the frames to reflect this, we can mirror the correct positions by crossing over frame boundaries.

A1, A2C8, A3C7, A4C6, A5

B1, B2D8, B3D7, B4D6, B5

C1, C2A8, C3A7, C4A6, C5

D1, D2B8, D3B7, D4B6, D5

This also totals 20 sprite frames, saving 12. This method will eliminate the mirrored effect present in the first layout.

Consider mirroring the entire 8-angle sprite set to its counter:

A1C1, A2C8, A3C7, A4C6, A5C5, A6C4, A7C3, A8C2

B1D1, B2D8, B3D7, B4D6, B5D5, B6D4, B7D3, B8D2

C1A1, C2A8, C3A7, C4A6, C5A5, C6A4, C7A3, C8A2

D1B1, D2B8, D3B7, D4B6, D5B5, D6B4, D7B3, D8B2

We now have 32 angles for the 4 frames, which mirror properly to the opposite sequence. Doom will read any sprite angle declaration in the second position as a mirror of the first. Knowing this, we can remove the entire second set of sprite frames and get the same effect:

A1C1, A2C8, A3C7, A4C6, A5C5, A6C4, A7C3, A8C2

B1D1, B2D8, B3D7, B4D6, B5D5, B6D4, B7D3, B8D2

Totaling 16 frames, saving 16.

If you come away with nothing else from that description, let it be this: Using the last method means less work for the sprite artist!

Creating new weapons

While making a custom weapon in ZDoom is a bit tricky, this guide should help you get on your way.

Note: This tutorial uses custom sprites, which are not provided. You should provide sprites with these names or adapt the code to use standard sprite names if you want to actually test this code in ZDoom.

DECORATE Code

A basic weapon in the decorate lump would look like this:

```
ACTOR UberShotgun : SuperShotgun 20024
{
    Weapon.SelectionOrder 350
    Inventory.PickupSound "misc/usgpickup"
    Weapon.AmmoGive 12
    Weapon.AmmoUse 4
    Weapon.SlotNumber 3
    AttackSound "weapons/ubersgf"
    States
    {
        Spawn: // This state is entered when you drop the weapon.
            USGP A -1
            Stop
        Ready: // This state is entered when you have this weapon selected.
            USHG A 1 A_WeaponReady
            Loop
        Deselect: // This state is entered when you deselect the current selected weapon.
            USHG A 1 A_Lower
            Loop
        Select: // This state is entered when you select this weapon.
            USHG A 1 A_Raise
            Loop
        Fire: // The firing state.
            USHG A 3
            USHG A 0 A_FireBullets (13.4, 9.2, 50, 10)
            USHG A 7 A_GunFlash // While on fire state it spawns the weapon's flash.
            USHG B 7
            USHG C 7 A_CheckReload
            USHG D 7 A_OpenShotgun2
            USHG E 7
            USHG F 7 A_LoadShotgun2
            USHG G 6
            USHG H 6 A_CloseShotgun2
            USHG A 5 A_ReFire
            Goto Ready
        Flash: // The weapons flash.
            USGF A 6 bright A_Light1
            Goto LightDone
    }
}
```

Now that big chunk of code describes the functions and states of a shotgun. Let's analyse this code.

Defining an actor

See DECORATE format specifications for more information

(ACTOR UberShotgun : SuperShotgun 20024) - Describes the actor that is going to be defined within the following code block.

The format for this line is: ACTOR <ActorName> [: <inheriting actor>] [<DoomEd Number>] [replaces <replaces>] So our actor is going to be named "UberShotgun", it will inherit all the properties of the super shotgun ("SuperShotgun"), and its DoomEd Number will be 20024.

Note: In order to make a weapon, the actor always has to inherit from Weapon or another actor that does inherit from Weapon, such as SuperShotgun.

The brackets (" { }") define the start and end of the code block. All the information pertaining to this actor must be within these brackets.

Actor properties

See Actor properties for a full list

Next there are various weapon properties, such as the ammo type the weapon uses, and how much it uses with each shot.

(Inventory.PickupSound "misc/usgpickup") - Tells ZDoom what sound to play when the item is picked up, as be defined in the SNDINFO lump.

(Weapon.AmmoGive 12) - Tells ZDoom how much ammo to give the player when this weapon is picked up. This is NOT the maximum amount of ammo it can hold.

(Weapon.AmmoUse 4) - Tells ZDoom how much ammo to use when firing the weapon, provided the use argument in the relevant codepointers is set.

(Weapon.SlotNumber 3) - Tells ZDoom that the weapon will occupy slot 3, like the other shotguns. There are other ways to set weapon slots, but this is the simplest by far.

(AttackSound "weapons/ubersgf") - Tells ZDoom what sound to play when A_FireBullets or A_CustomPunch are called [Has to be defined in the SNDINFO lump]. If the weapon shoots a custom projectile, such as a rocket, the sound played will be the projectile's spawn sound.

(Weapon.SelectionOrder 350) - Tells ZDoom how much priority this weapon has. So if you run out of ammo, you might switch to the Super Shotgun, rather than the Pistol, because its priority is higher. The lower this value, the higher the priority.

There are other properties you can set here, but they are not all required for every weapon. This weapon inherits all of its missing properties from the original shotgun. Like the pickup message, if we wanted to define a custom pickup message, then we could put in a line of code that looks like this;

(Inventory.Pickupmessage "You got the UBER SHOTGUN!")

Also, notice how this did not define an ammo type. Since this weapon was meant to use the same shells as the other shotguns, inheriting makes it a somewhat shorter process by taking out the unneeded pieces of code that need to be written.

Read all about these various properties you can set for weapons on the Weapon and Inventory pages.

Actor states

The next chunk in the code defines the Actor states of the weapon. These tell ZDoom what is going on and what to do at certain frames. This removes the previous limitations of DeHacked or Whacked where you couldn't add in more frames, but could only replace the current ones. With this method, you are able to make brand new sequences, to brand new weapons.

Within a state there is a certain form to be observed.

To define a state: <State Name>: (Note the colon.)

To define a frame: <Sprite> <Frame(s)> <Duration> [<Action>]

<Sprite> is the first four letters of the sprite name. The sprite names are usually representative of what they are, in this

case, the three are: (Uber Shotgun Pickup), (Uber Shotgun), and (Uber Shotgun Flash).

<Frame> is the letter that follows. If you specify more than one letter, the <Action> and <Duration> will be repeated for every frame (one for each letter).

<Duration> is in tics (1/35 seconds).

<Action> is an optional function that will be performed by the actor.

There are also several special commands you can execute:

Loop - Loops the entire state. Not needed if you've already defined -1 as a duration, because it may cause problems.

Wait - Loops the last frame until otherwise instructed by a function.

Stop - Stops animating the actor. If the last frame defined has a duration of -1, it will be displayed forever. Otherwise, the actor will be removed.

Goto <State> [+<Offset>] - Jumps to the specified state. If <Offset> is specified it will skip that many frames from the beginning of the state.

Note: These functions are not frames, and will not be counted as part of any jump function or a Goto command with an offset.

Let's analyse the states of our UberShotgun.

Spawn state

First off is the spawn state: Code:

Spawn:

USGP A -1

Stop

Spawn is the state the weapon will be in when it is spawned as a pickup. This state definition is very simple. The shotgun lying on the ground will display the sprite USGPA until it's picked up by the player.

Ready state

Next in these series of states is the READY state: Code:

Ready:

USHG A 1 A_WeaponReady

Loop

This state will be entered when the weapon has been selected. The action A_WeaponReady tells ZDoom that the weapon is ready, ie. it can fire, be deselected, and freely bob. The sprite, USHGA, is the image of the shotgun when it's not doing anything.

Select/Deselect states

The next two states are entered when the player selects, or deselects the weapon. Look at the format of the SELECT and DESELECT states: Code:

Deselect:

USHG A 1 A_Lower

Loop

Select:

USHG A 1 A_Raise

Loop

By default, when a weapon is selected, its sprite is drawn off the screen. The action A_Raise moves the sprite up, and by looping the state, the weapon will be moved until its sprite is at the correct vertical position, at which point the game will go to the Ready state. You can also speed up both actions by adding another line before the current line you have defined for deselect/select, with 0 tic duration. Your code should look like this;

Code:

Deselect:

TNT1 A 0 A_Lower

```
USHG A 1 A_Lower
Loop
Select:
  TNT1 A 0 A_Raise
  USHG A 1 A_Raise
Loop
```

When the weapon is now selected/deselected, the animation will be faster.

Fire state

The Fire state is the state that's entered when the player presses the fire button. The firing sequence looks like this:

Code:

Fire:

```
USHG A 3
USHG A 0 A_FireBullets (13.4, 9.2, 50, 10)
USHG A 7 A_GunFlash
USHG B 7
USHG C 7 A_CheckReload
USHG D 7 A_OpenShotgun2
USHG E 7
USHG F 7 A_LoadShotgun2
USHG G 6
USHG H 6 A_CloseShotgun2
USHG A 5 A_ReFire
Goto Ready
```

The first frame shows us the same sprite we saw in the ready state, this will mean that firing is not instant (although it's pretty close). The next frame last 0 tics, this means that frame and the one after it will be executed at the same time. If you ever need two or more actions to happen at the same time, you can use a frame with 0 duration.

The first of these frames uses A_FireBullets, this is what actually shoots the gun. The second frame uses A_GunFlash. What this means is that the Flash state will be executed and run in tandem with the Fire state from that point onward.

A_CheckReload will check if the player still has enough ammo to fire another shot (4 shells in this case), and if they don't, it will switch their weapon to the highest priority available (remember Weapon.SelectionOrder). This makes sense, because if the player had no ammo left, we wouldn't want them to see the 'reloading' sprites that are coming up next.

The rest of the frames would show the 'reloading' sequence of the shotgun. The 3 Open/Load/Close Shotgun2 actions play the sounds of the shotgun reloading.

The last frame of the sequence has A_ReFire. This will check if the player still has the fire button held down, and if they do it will jump to the Hold state. If no Hold state exists, it will jump to the Fire state instead.

The hold state may not seem useful at first, but consider if the weapon you were making was a laser cannon. When you first press the fire button, the cannon needs to charge up, and then it shoots the actual laser beam. But you wouldn't want it to shoot the laser for 6 tics and then go back to the charging sequence. Instead you would create a hold state that is exactly like the Fire state but without the charging sequence.

Finally, at the end of the sequence, we go back to the Ready state so the player can fire again, or switch to another weapon.

Flash state

The last state in our weapon is the Flash state. Code:

Flash:

USGF A 6 bright A_Light1

Goto LightDone

As stated above, this state runs at the same time as the Fire state, because of when we called A_GunFlash. The first and only frame of this state has the sprite USGFA, which is an image of the muzzle flash. It also has the 'bright' keyword, this means the sprite will be drawn at full brightness, regardless of the light level of the sector. It also calls A_Light1 to illuminate the level slightly. There's also A_Light2 for stronger illumination. Alternatively, you may want to use A_AttachLight or A_AttachlightDef to attach a dynamic light instead of using vanilla-style global illumination. Having a flash state is not mandatory. At the end of the sequence we have goto LightDone. This sends the state to an internal state label LightDone that simply removes the lighting by calling A_Light0 and then stops the animation. We don't need to go back to Ready, because the Fire sequence is running underneath the Flash one, and when the Fire sequence is completed, it will return to the Ready state.

Conclusion

Hopefully you now have a better idea of how to use DECORATE to create a custom weapon. Remember, the best way to learn is by doing, so go try it for yourself!

Creating non-interactive decorations

Non-interactive decorations are the simplest form of actor that can be defined. They usually consist of a single sprite with one frame or a looping animation. Therefore they only define a Spawn state which defines the entire visual appearance.

You may set any property and flag as needed.

This is an example of a simple decoration:

```
actor CEye 10242
{
    Height 40
    Radius 20
    +SOLID
    States
    {
        Spawn:
            HAW6 A 10
            HAW6 B 10 Bright
            HAW6 C 10
            Loop
    }
}
```

Simple decorations can use code pointers just as any other actor. This is an example of a decoration that uses actions:

```
actor Pulsar 10239
{
    PainSound "pulsar/pulse"
    +NODAMAGETHRUST
    States
    {
        Spawn:
            HAX7 A 15 bright
            HAX7 B 15 bright A_Pain
            HAX7 C 20 bright A_Explode
            Loop
    }
}
```

DECORATE format specifications

This page describes the format to create a new actor in the DECORATE lump:

```
actor classname [ : parentclassname] [replaces replaceclassname] [doomednum]
```

```
{  
  properties  
  flags  
  ...  
}
```

classname

The name this new actor is referenced by in the game. While ZDoom will accept a much larger range of values for the name, this should for best compatibility be a valid identifier (alphanumeric plus underscores, but not starting with a digit).

parentclassname

The name of a parent class this new actor inherits its attributes from (optional). If none is specified, the parent class is Actor.

replaceclassname

The name of the class this class replaces (optional). This works at a higher level than using duplicate doomednums and will affect all attempts to spawn the replaced actor in the map. However, this does not affect effect that create the actor through other means, such as giving an inventory item directly to a player. Player actors are handled differently from other actors, so this method also does not work for custom player classes. Skill definitions may specify other replacement for an actor, and occurs prior to Decorate replacement.

doomednum

Editor number for this actor (optional). This is the number used to distinguish the actor from other things in map. If the actor is intended to be placed in a map editor, it should have an editor number. The actual number value is generally arbitrary but should avoid conflicting with already used numbers.

An actor definition consists of properties, flags and state definitions. In the state definitions you can call Action functions.

Actor properties and flags define the general behavior of an actor.

States define the various sprite animations of an actor.

Action functions (also known as "code pointers") cause the actor to perform some particular action when the frame that calls them is shown. They form the basis of almost all enemy and weapon behavior in the game. Instead of using one of the special action functions you can also use almost any action special that is available in ACS.

Comments are supported. Both types of C-style comments (`//` to end of line, and `/*` to `*/`) are allowed. While not part of the specification, certain editing tools, such as Doom Builder and SLADE 3, make use of specific comments for special purposes, these usually start with `//$`.

Including Decorate scripts into others

Using the `#include "<full_path_and_lump_name>"` directive, you can include other Decorate scripts, for example, if you wanted to organize your data into subfolders; a common practice is to place Decorate lumps into actors/ subfolder of a PK3-file, leaving only the main `decorate.txt` in the root directory, that includes them. That also allows to precisely set the loading order of lumps, in case of some actors used in several places, in order to avoid loading errors. The directive may appear in any place other than actor definition.

DECORATE usage

Actors are also capable of storing individual constant integers. Unlike user variables, these cannot be changed.

These may be used in parameters of almost any kind, except for strings or actor names. Constants may also be declared outside of an actor for global usage.

```
const int MyGlobalInt = 2;
Actor TestDummy
{
    const int MyConst = 1;
    const int FLAGS = CLOFF_JUMPOBJECT|CLOFF_SKIPTARGET; //Saves on space for defining flags!
    const float Floaty = 1.234;
}
```

Enums can also be used for integer values only. All others (i.e. floats) must be defined as above outside an enum. Assigning a number to specific values means they will always be that number, rather than being auto-assigned numbers. This may be vital to ensuring compatibility with saved games or demos, since adding anything to the enum list may cause unintended side effects or prevent proper function. A semicolon is required on the closing bracket of an enum. Like constants, they can be inside of an actor.

Examples:

```
Actor Test
{
    enum //Non-initialized numbers
    {
        StateNum_Punch, //Autoassigned 0
        StateNum_Kick, //Autoassigned 1
    };
    //...States and stuff go here.
}
enum //Initialized numbers
{
    Mons_MaxCount = 128,
    Mons_MinCount = 4,
};
```

Notes

DECORATE:

Constants can only be used with decorate functions.

Constants cannot be used for defining actor parameters. For example Health must have an actual number, not a constant. Inheritance from parents only overwrites constants if the state itself is overwritten. It is recommended to have a state dedicated to the setup of user variables instead.

DECORATE

DECORATE is a text-format lump which allows one to define actors that can be placed in a level. DECORATE was originally intended to aid in the creation of decorative objects such as the lamps and torches in Doom and other games without using up extra frames in DeHacked. However, it has been expanded to create virtually anything you want, not just decorations.

DECORATE is currently deprecated in GZDoom: while it's still supported, it has been completely replaced by ZScript. All DECORATE features, such as actor properties, flags and functions are available in ZScript. Using DECORATE is still possible but is not recommended since it doesn't offer any advantages over ZScript, and all DECORATE methods can still be applied in ZScript.

For historical reasons there are two distinct ways to define actors here. However, the old method has been completely superseded by the new one and it is strongly advised not to use it anymore.

DECORATE lumps are cumulative, meaning several may be loaded at once without overwriting each other in memory. Therefore, adding new actors to ZDoom does not require editing the pre-existing actor definitions. Instead, authors should simply create a new text-format lump (WAD format) or file (PK3 format) called DECORATE and define only their new actors there. DECORATE lumps support including other lumps with the `#include` directive.

For examples of valid DECORATE code, see the list of ZDoom classes. Many DECORATE actors are also available on Realm 667.

List of DECORATE topics:

- DECORATE format specifications
- The old DECORATE format (deprecated)
- Creating non-interactive decorations
- Creating new monsters or other complex items
- Creating new projectiles
- Creating new inventory items
- Creating new weapons
- Creating decorations that can be (de)activated
- Creating new player classes
- Using inheritance
- Improving original monsters with DECORATE
- Using Constants in DECORATE
- Making weapon pieces
- Projectile Trap
- Custom CVARs with DECORATE
- Advanced DECORATE functions guide
- Creating new sprite graphics
- Converting Sprites
- List of actor references:

- DECORATE expressions
- Action functions
- Actor flags
- Actor properties
- Actor states
- Actor pointers

Improving original monsters with DECORATE

DECORATE can do more than creating new stuff, you can also use it to improve the original monsters. Sometimes they act strangely, and DECORATE can help to remove these little quirks.

Improvement examples

Take a close look at the Cacodemon. When it dies, it collapses in the air or slowly collapses after hitting the floor, but we all would expect it to smash on the ground, splashing its guts out instead. So let's fix this quirk.

```
actor Cacodemon2 : Cacodemon replaces Cacodemon
```

```
{
  BloodColor "blue"
  states
  {
    Death:
      HEAD G 4 A_SetSolid
      HEAD G 4 A_SetShootable
      HEAD H 8 A_Scream
      HEAD H -1
      stop
    Crash:
      HEAD I 4 A_PlaySound ("*fist") //I had no better sound, so feel free to use another.
      HEAD J 4
      HEAD K 2 A_NoBlocking
      HEAD K 1 A_UnsetSolid
      HEAD K 1 A_UnSetShootable
      HEAD L -1 A_SetFloorClip
      stop
    Raise:
      HEAD L 8 A_UnSetFloorClip
      HEAD KJIHG 8
      goto See
  }
}
```

This DECORATE code, for example, improves the Cacodemon a little bit now. You can shoot it in the air, and it blocks any shots while falling. When it hits the ground, the state "Crash" is called (this state is called every time an actor is dead and hits the floor); it contains the code for collapsing and resetting the old corpse behavior (unshootable etc.) It's obviously not the best solution for it, but still a nice improvement when they drop after shooting them on maps with a high ceiling.

It also changes the Cacodemon's blood color to blue rather than the default red color, as the Cacodemon's insides appear blue when he dies.

Some other monsters can be improved just by simply changing their blood color. For example, the Baron of Hell and Hell Knight's blood color can be changed to green because they bleed out a green pool of blood upon dying.

```
actor BaronofHell2 : BaronOfHell replaces BaronOfHell
```

```
{
  BloodColor "green"
}
```

However, the Hell Knight will not be affected by the change since it's still going to inherit from the original Baron of Hell. As such, you'll need to modify it as well to reflect the change.


```
actor HellKnight2 : HellKnight replaces HellKnight
{
  BloodColor "green"
}
```

Making weapon pieces

There is now a way to create pieces of a weapon that assemble into one weapon, just like the three strongest weapon in Hexen, being Quietus, Wraithverge, and BloodScourge. Players would have to hunt for these pieces in order to get that weapon. This is a great way to add a powerful weapon into your mod without it being too easy to acquire.

To do this, you would need your complete weapon first. Create your DECORATE entry for your complete weapon, just like anything else, except you give the weapon a HEALTH property. The HEALTH you just defined has nothing to do with the weapon's life; this value determines the number of WeaponPieces required to complete the weapon. The health of 5 would mean that Weapon Pieces number 1 through 5 are required to fully complete the weapon.

As a side note, it would be a good idea not to define its SPAWN state so that the weapon is never found in its complete state.

Now, you would need to define your Weapon Piece's entry. This entry should have its parent as "WeaponPiece". Now, it needs some WeaponPiece specific properties:

WeaponPiece.Weapon "name"

Name of the complete weapon.

WeaponPiece.Number value

Specific ID for the particular Weapon Piece being defined. Value of (1-32) are acceptable.

Inventory.Pickupmessage "string" (optional)

Message to display when you pick up that piece.

Now, define its SPAWN state. Do this for your other Weapon Pieces. You may have up to 32 weapon pieces.

Here is an example code of a weapon that requires 3 pieces to complete:

ACTOR Nightmare : Weapon

```
{
  Health 3
  Inventory.Pickupmessage "Time for the Nightmare...!"
  Obituary "%k deserves to kill %o because %g worked hard to assemble this weapon."
  States
  {
    Ready:
      NITE A 1 A_WeaponReady
      LOOP
    ...
  }
}
```

ACTOR NMPiece1 : WeaponPiece

```
{
  WeaponPiece.Number 1
  WeaponPiece.Weapon "Nightmare"
  Inventory.Pickupmessage "Picked up a fragment of something."
  States
  {
    Spawn:
      PIEC A -1
      Loop
  }
}
```

```

ACTOR NMPiece2 : WeaponPiece
{
    WeaponPiece.Number 2
    WeaponPiece.Weapon "Nightmare"
    Inventory.Pickupmessage "Picked up a fragment of something."
    States
    {
        Spawn:
            PIEC B -1
        Loop
    }
}

```

```

ACTOR NMPiece3 : WeaponPiece
{
    WeaponPiece.Number 3
    WeaponPiece.Weapon "Nightmare"
    Inventory.Pickupmessage "Picked up a fragment of something."
    States
    {
        Spawn:
            PIEC C -1
        Loop
    }
}

```

The strongfaced parts are what makes the Weapon Pieces work. Although this is not really a working weapon, this "Nightmare" weapon requires 3 pieces of itself, NMPiece1, NMPiece2, and NMPiece3. When these are all collected, you will have completed the Nightmare weapon and will immediately be switched to your fresh weapon. The names of the pieces do not really matter. As long as the weapon has its health defined the number of the pieces, and your weapon pieces inherit "WeaponPiece" and define its complete weapon and number, there should be no problem.

Of course, the weapon pieces must be obtainable somehow. You can give the pieces a DoomEdNumber and spawn them in your levels, or you can make the monsters drop them in a random occasion. Also, you should keep the number of your weapon pieces into something like three to five, unless your weapon pieces are ridiculously easily obtained.

Projectile Trap

Description

Add this to your DECORATE lump, and this will add a new thing that fires projectiles in the direction of the thing. It has 3 modes. It works using Thing Arguments.

Arg0(Mode) - Determines the mode this thing is in. There are 3.

1 - Fires a continuous stream of projectiles, non-stop.

2 - Does the same, but will only fire when it can see a hostile target

3 - Same as 2, but unlike the other modes, this one turns and fires directly at its target.

Anything else defaults to 1. This includes 0.

Arg1(Type) - The SpawnID of the projectile. Note that it doesn't actually need to be a projectile. Could be useful as an ammo generator. Might spawn too much though.

Arg2(Speed) - The speed at which the projectile is thrown. This is in units per 8 tics, so eight times the value of a projectile's Speed. For example, a PlasmaBall has a speed property of 25, so for the projectile trap to throw it at its normal speed, you would need to use a value of 200 (25*8).

Arg3(Pitch) - The vertical speed for the projectile to fire at. 0 is horizontal, negative is down and positive is up.

Arg4(FOV) - Used for determining whether to fire in modes 2 and 3. It's the trap's field of view.

To use it in your map:

Add the definition to the DECORATE lump

Add a thing of type 20005 to the map. You can change this if it gets in the way.

Give it the correct facing and ZHeight, and set the arguments.

Other tips:

You can change how fast it fires by editing the first State definition duration. On the line TNT1 A 5 A_LookEx... change the number. Higher numbers result in slower fire. DO NOT SET TO ZERO. This is the number of frames the delay lasts. 35 frames = 1 second. By default it fires at a rate of 5, or 7 rounds per second.

You can disable it by setting the DORMANT flag and unsetting with a script.

Set the friendly flag for modes 2 and 3 to make it only react to enemies.

The Definition

ACTOR TrapProjectileLauncher 20005

```
{
    Monster
    +STANDSTILL
    +NOBLOCKMAP
    +NOGRAVITY
    -SHOOTABLE
    States
    {
    Spawn:
        TNT1 A 5 A_LookEx (LOF_NOSOUNDCHECK, 0, 0, 0, 0, "SpawnSkip")
        Loop
    SpawnSkip:
        TNT1 A 5
        TNT1 A 0 A_JumpIf (args[0]==1, "Constant")
        TNT1 A 0 A_JumpIf (args[0]==2, "Automatic")
        TNT1 A 0 A_JumpIf (args[0]==3, "Smart")

    Constant: //First Function: Always Shoot
        TNT1 A 0 Thing_Projectile (0, args[1], (angle * 256 / 360)%256, args[2], args[3])
        TNT1 A 0
        Goto Spawn
    }
```

Automatic: //Second Function: Shoot when in sight

TNT1 A 0 A_JumpIfTargetInLOS (2,args[4]*256)

TNT1 A 0 A_Jump (256,2)

TNT1 A 0 Thing_Projectile (0, args[1], (angle * 256 / 360)%256, args[2], args[3])

TNT1 A 0

Goto Spawn

Smart: //Third Function: Same as second, but can turn.

TNT1 A 0 A_FaceTarget

TNT1 A 0 A_JumpIfTargetInLOS (2,args[4]*256)

TNT1 A 0 A_Jump (256,2)

TNT1 A 0 Thing_Projectile (0, args[1], (angle * 256 / 360)%256, args[2], args[3])

TNT1 A 0

Goto Spawn

}

}

Using inheritance

Inheritance is a mechanism that lets you take all properties defined by a previous actor and only change a few of its properties, flags or states.

Inheritance has special meaning when used with monsters. Any related monsters belong to one 'species'. Monsters within the same species cannot hurt each other with projectiles. A species is defined as all monsters that have one monster as a common ancestor. For example, if you create various variations of imps which are all derived from DoomImp they will form one species. This automatic mechanism can be altered with the Species property.

Inheritance is an essential means to create new inventory items. Inventory items are derived from predefined inventory classes.

Examples

This is a Zombie with a different attack and more health. But it inherits everything else from the already existing Doom zombie:

DECORATE:

```
actor PlasmaZombie : ZombieMan 9600
{
    health 40
    dropitem Cell
    missiletype PlasmaBall
    states
    {
        Missile:
            POSS E 10 A_FaceTarget
            POSS F 5 A_MissileAttack
            POSS E 5 A_FaceTarget
            POSS F 5 A_MissileAttack
            POSS E 5 A_FaceTarget
            POSS F 5 A_MissileAttack
            goto See
    }
}
```

This is a dead Zombieman from Doom. It uses SKIP_SUPER to reset the actor to default values. It only uses inheritance to get access to the parent's states:

DECORATE:

```
actor DeadZombieMan : ZombieMan 18
{
    SKIP_SUPER
    DropItem "None"
    States
    {
        Spawn:
            Goto Super::Death +4
    }
}
```

Note that using inheritance retains all states unless redefined. As an example, the ScriptedZombie below has an incomplete death animation because calling the Death state removes any previous inheritance from that state. This is a zombie which uses a script to perform its attack and opens a door when dying:

DECORATE:

```
actor ScriptedZombie : ZombieMan 9604
{
    health 40
    states
    {
        Missile:
            POSS E 10 A_FaceTarget
            POSS F 5 ACS_ExecuteAlways (999,0,0)
            goto See
        Death:
            POSS A 1 Door_Open (1337, 16)
            stop
    }
}
```

Another possibility is to inherit from actors that have programmed capabilities. For example, Strife's LoreShot works like a grappling hook. You can easily get access to this behavior by inheriting from it and define your new actor around it:

DECORATE:

```
actor GrapplingHook : LoreShot
{
    seesound "hook/shoot"
    deathsound "hook/hit"
    states
    {
        Spawn:
            WS12 AB 2 bright
            loop
        Death:
            WS12 CDEF 6
            stop
    }
}
```

States in parent classes which have been redefined in a subclass can still be accessed using the 'Super' scope. In this example 'Super' allows us to do stuff at the beginning of a pain state without rewriting the entire state.

DECORATE:

```
actor WimpyZombieMan : ZombieMan replaces ZombieMan
{
    States
    {
        Pain:
            TNT1 A 0 A_ChangeFlag("FRIGHTENED", 1)
            goto Super::Pain
    }
}
```

```
}  
}
```

As an alternative to using 'Super,' you can use the name of the superclass whose state you want to jump to. This example has former humans calling ACS scripts when they die. It is especially useful to go farther up in the hierarchy than the direct parent:

DECORATE:

```
actor MyZombieMan : ZombieMan replaces ZombieMan  
{  
    States  
    {  
        Death:  
            TNT1 A 0 ACS_ExecuteAlways(85)  
            goto Super::Death  
    }  
}  
actor MyOtherZombieMan : MyZombieMan replaces ShotgunGuy  
{  
    States  
    {  
        Death:  
            TNT1 A 0 ACS_ExecuteAlways(86)  
            goto ZombieMan::Death  
        XDeath:  
            TNT1 A 0 ACS_ExecuteAlways(86)  
            goto ZombieMan::XDeath  
    }  
}
```

Dynamic and static jumps

Jumps can be dynamic and static. Goto is the only example of a static jump. Other jumps, such as A_JumpIf, A_JumpIfNoAmmo, A_JumpIfInventory and others, as well as returning a state directly, are all dynamic jump methods.

It's important to remember that goto performs the jump within the actor, it's unaffected by inheritance. Goto <Label> in the parent actor will always jump to the "Label" sequence inside that actor, even if the child actor defines the same state label:

```
class ParentActor : Actor  
{  
    States  
    {  
        Spawn:  
            FRAM ABC 1;  
            goto Death; //this will only go to Death within this actor  
        Death:  
            FRAM DE 1;  
            stop;  
    }  
}  
  
class ChildActor : ParentActor  
{  
    States  
    {
```



```
Death: //this will never be entered, because parent Death will be used instead
```

```
    FRAM AB 1;
```

```
    loop;
```

```
  }
```

```
}
```

The only way to create a "goto" that work properly with inheritance is to use dynamic jumps insteadâ€”i.e. the A_Jump* functions or return ResolveState("<state label>"). For example:

```
class ParentActor : Actor
```

```
{
```

```
    States
```

```
    {
```

```
    Spawn:
```

```
        FRAM ABC 1;
```

```
        TNT1 A 0 A_Jump(256,"Death");
```

```
        wait; //it's OK to loop a 0-tic state here, since it contains a 100% chance jump
```

```
    Death:
```

```
        FRAM DE 1;
```

```
        stop;
```

```
    }
```

```
}
```

```
class ChildActor : ParentActor
```

```
{
```

```
    States
```

```
    {
```

```
    Death: //this will be entered properly
```

```
        FRAM AB 1;
```

```
        loop;
```

```
    }
```

```
}
```

This will also work:

```
    Spawn:
```

```
        FRAM ABC 1;
```

```
        TNT1 A 0
```

```
        {
```

```
            return ResolveState("Death");
```

```
        }
```

```
        stop; //this will never be reached, so it doesn't matter what you use here
```

CREDIT v1.0

All content taken from: https://zdoom.org/wiki/Main_Page

NLP Engineering: Kontra Kommando

April 18, 2023