State Controller Reference

Documentation for version 1.0 (2009)

Updated 13 September 2010

Contents

About Controllers

All state controllers have two optional parameters, "persistent" and "ignorehitpause". These must be set to integer constants. Unless otherwise specified, any other numeric state controller parameter can be specified with an arithmetic expression.

In all cases, if setting a parameter with an expression, you should be careful that the expression does not evaluate to bottom, as in this case the parameter will be set to 0.

Controller Reference

AfterImage

Enables player afterimage effects. The character's frames are stored in a history buffer, and are displayed after a delay as afterimages.

Required parameters:

none

Optional parameters:

time = *duration* (int)

Specifies the number of ticks that the afterimages should be displayed for. Set to -1 to display indefinitely. Defaults to 1.

length = *no_of_frames* (int)

Sets the capacity of the frame history buffer. The history will hold up to *no_of_frames* of the character's most recently saved frames. Assuming constant values for timegap and framegap, increasing the length can increase the number and "age" (for lack of a better term) of afterimages displayed at one time. The maximum length is 60, and the default is 20.

palcolor = *col* (int)

See below.

palinvertall = *invertall* (bool)

See below.

palbright = *add_r*, *add_g*, *add_b* (int)

See below.

palcontrast = *mul_r*, *mul_g*, *mul_b* (int)

See below.

palpostbright = *add2_r*, *add2_g*, *add2_b* (int)

These parameters determine palette effects to be applied to all afterimages. First the color level is adjusted according to the palcolor value, then if invertall is non-zero the colors are inverted. Afterwards, the palbright components are added to the corresponding component of the player's palette, then each component is multiplied by the corresponding palcontrast component divided by 256, then the palpostbright components are added to the result. The value of palcolor ranges from 0 (greyscale) to 256 (normal color). For instance, if the red component of the character's palette is denoted *pal_r*, then the red component of the afterimage palette is given by (*pal_r* + *add_r*) * *mul_r* / 256 + *add2_r*, assuming palcolor and palinvert are left at their default values. Valid values are 0-256 for palcolor, 0-255 for palbright and palpostbright components, and any non-negative integer for palcontrast components. The defaults are:

```
palcolor = 256
palinvertall = 0
palbright = 30,30,30
palcontrast = 120,120,220
palpostbright = 0,0,0
```

paladd = *add_r*, *add_g*, *add_b* (int)

See below.

palmul = *mul_r*, *mul_g*, *mul_b* (float)

These parameters specify palette effects that are applied repeatedly to successive frames in the afterimage. In one application of these palette effects, first the paladd components are added to the afterimage palette, then the components are multiplied by the palmul multipliers. These effects are applied zero times to the most recent afterimage frame, once to the second-newest afterimage frame, twice in succession to the third-newest afterimage frame, etc. Valid values are 0-255 for the paladd components, and any non-negative float value for the palmul multipliers. The defaults are:

```
paladd = 10,10,25
palmul = .65,.65,.75
```

timegap = *value* (int)

This parameter controls how many frames to skip between saving player frames to the history buffer for afterimage display. The default is 1 (skip no frames). To save every third frame (for example), you would use timegap = 3.

framegap = *value* (int)

Every *value*'th frame in the history buffer will be displayed as an afterimage. For instance, if framegap = 4 (the default), then the first, fifth, ninth, ... frames of the history buffer will be displayed as afterimages.

trans = *type* (string)
Specifies the transparency type for the afterimages. Valid values for *type* are "none" for an opaque afterimage, "add", "add1", and "sub". Defaults to "none".

Example:
none

AfterImageTime

Changes the duration of the player's afterimage effects, if currently enabled. If no afterimage effects are being displayed, this controller does nothing. Known bugs: If the timegap parameter in the originating AfterImage controller is not set at 1, using this AfterImageTime will cause the frame positions to be reset.

Required parameters:
time = *new_duration* (int)
Sets the new number of ticks that the afterimages will be displayed before being removed.

Alternate syntax:
value = *new_duration* (int)

Optional parameters:
none

Example:
none

AllPalFX

Same as PalFX, except that this affects the palette of the background and lifebars as well as the palette of all characters and explods (regardless of the ownpal parameter). See the PalFX section for details on the parameters to AllPalFX.

AngleAdd

Adds to the drawing rotation angle used by AngleDraw.

Required arguments:
value = *add_angle* (float)
add_angle should be given in degrees.

Optional arguments:
none

Example:
none

AngleDraw

Draws the player (for 1 frame) rotated about his axis by the angle set by the AngleSet controller. When facing right, a positive angle means a counterclockwise rotation.

Required arguments:
none

Optional arguments:
value = *angle* (float)
Sets the drawing angle in degrees.

scale = *xscale*, *yscale* (float, float)
Scales the player sprite.

Notes:
Rotation/scaling does not affect the player's collision boxes.

Example:
none

AngleMul

Multiplies the drawing rotation angle used by AngleDraw by the specified factor.

Required arguments:
value = *angle_multiplier* (float)
Multiplies the drawing angle by *angle_multiplier*.

Optional arguments:
none

Example:
none

AngleSet

Sets the drawing rotation angle used by AngleDraw. The angle is initialized at 0.

Required arguments:
value = *angle* (float)
the value of *angle* is interpreted to be in degrees.

Optional arguments:
none

Example:
none

AppendToClipboard

This is the same as DisplayToClipboard, except that message text is added on a new line, instead of overwriting whatever text is already on the clipboard. See DisplayToClipboard for a format description.

AssertSpecial

This controller allows you to assert up to three special flags simultaneously. MUGEN will automatically "deassert" each flag at every game tick, so you must assert a flag for each tick that you want it to be active.

Required parameters:
flag = *flag_name*
flag_name is a string specifying the flag to assert.

Optional parameters:
flag2 = *flag2_name*
An optional flag to assert.

flag3 = *flag3_name*
Another optional flag to assert.

Details:
The flag name can be one of the following:
intro

Tells MUGEN that the character is currently performing his intro pose. Must be asserted on every tick while the intro pose is being performed.

invisible
: Turns the character invisible while asserted. Does not affect display of afterimages.

roundnotover
: Tells MUGEN that the character is currently performing his win pose. Should be asserted on every tick while the win pose is being performed.

nobardisplay
: Disables display of life, super bars, etc. while asserted.

noBG
: Turns off the background. The screen is cleared to black.

noFG
: Disables display of layer 1 of the stage (the foreground).

nostandguard
: While asserted, disables standing guard for the character.

nocrouchguard
: While asserted, disables crouching guard for the character.

noairguard
: While asserted, disables air guard for the character.

noautoturn
: While asserted, keeps the character from automatically turning to face the opponent.

nojugglecheck
: While asserted, disables juggle checking. P2 can be juggled regardless of juggle points.

nokosnd
: Suppresses playback of sound 11, 0 (the KO sound) for players who are knocked out. For players whose KO sound echoes, nokosnd must be asserted for 50 or more ticks after the player is KOed in order to suppress all echoes.

nokoslow
: While asserted, keeps MUGEN from showing the end of the round in slow motion.

noshadow
: While asserted, disables display of this player's shadows.

globalnoshadow
: Disables display of all player, helper and explod shadows.

nomusic
: While asserted, pauses playback of background music.

nowalk
: While asserted, the player cannot enter his walk states, even if he has control. Use to prevent run states from canceling into walking.

timerfreeze
: While asserted, keeps the round timer from counting down. Useful to keep the round from timing over in the middle of a splash screen.

unguardable
: While asserted, all the asserting player's HitDefs become unblockable, i.e., their guardflags are ignored.

Example:
: none

AttackDist

Changes the value of the guard.dist parameter for the player's current HitDef. The guard.dist is the x-distance from P1 in which P2 will go into a guard state if P2 is holding the direction away from P1. The effect of guard.dist only takes effect when P1 has movetype = A.

Required parameters:
value = *guard_dist* (int)
New guard distance, in pixels.

Optional parameters:
none

Example:
none

AttackMulSet

Sets the player's attack multiplier. All damage the player gives is scaled by this amount.

Required parameters:
value = *attack_mul* (float)
Specifies the desired multiplier. For instance, an *attack_mul* of 2 deals double damage.

Optional parameters:
none

Example:
none

BGPalFX

Same as PalFX, except that this affects the palette of the background and lifebars instead of the palette of the character. See the PalFX section for details on the parameters to BGPalFX.

BindToParent

If the player is a helper, binds the player to a specified position relative to its parent. If the player is not a helper, this controller does nothing.

Required parameters:
none

Optional parameters:
time = *bind_time* (int)
Specify number of ticks that this binding should be effective. Defaults to 1.

facing = *facing_flag* (int)
If *facing_flag* is -1, makes the player always face the opposite direction from its parent during the binding time. If *facing_flag * is 1, makes the player always face the same direction as its parent during the binding time. If *facing_flag* is 0, the player will not turn regardless of what its parent does. Defaults to 0.

pos = *pos_x* (float), *pos_y* (float)
pos_x and *pos_y* specify the offsets (from the parent's axis) to bind to. Defaults to 0,0.

Notes:
If the player's parent is destroyed (for example, if it is a helper, and executes DestroySelf), then the effect of BindToParent is terminated.

Example:
none

BindToRoot

If the player is a helper, binds the player to a specified position relative to its root. If the player is not a helper, this controller does nothing.

Required parameters:
none

Optional parameters:
time = *bind_time* (int)
Specify number of ticks that this binding should be effective. Defaults to 1.

facing = *facing_flag* (int)
If *facing_flag* is -1, makes the player always face the opposite direction from its root during the binding time. If *facing_flag* is 1, makes the player always face the same direction as its root during the binding time. If *facing_flag* is 0, the player will not turn regardless of what its root does. Defaults to 0.

pos = *pos_x* (float), *pos_y* (float)
pos_x and *pos_y* specify the offsets (from the root's axis) to bind to. Defaults to 0,0.

Notes:
If the player's root is destroyed (for example, if it is a helper, and executes DestroySelf), then the effect of BindToRoot is terminated.

Example:
none

BindToTarget

Binds the player to a specified position relative to the specified target.

Required parameters:
none

Optional parameters:
time = *bind_time* (int)
Specify number of ticks that this binding should be effective. Defaults to 1.

ID = *bind_id* (int)
Specifies ID number of the target to bind to. Defaults to -1 (pick any target).

pos = *pos_x* (float), *pos_y* (float), *postype* (string)
pos_x and *pos_y* specify the offsets (from the bind point) to bind to. The bind point defaults to the target's axis. If *postype* is "Foot", then the bind point is the target's axis. If *postype* is "Mid", then the bind point is the target's midsection. If *postype* is "Head", then the bind point is the target's head. In the latter two cases, the bind point is determined from the values of the head.pos and mid.pos parameters in the target's CNS file. The bind point is not guaranteed to match up with the target's head or midsection.

Example:
none

ChangeAnim

Changes the action number of the player's animation.

Required parameters:
value = *anim_no* (int)
anim_no is the action number to change to.

Optional parameters:
elem = *elem_no* (int)
elem_no is the element number within the specified action to start from.

Example:
none

ChangeAnim2

Like ChangeAnim, except this controller should be used if you have placed P2 in a custom state via a hit and wish to change P2's animation to one specified in P1's air file. For example, when making throws, use this to change P2 to a being-thrown animation.

ChangeState

Changes the state number of the player.

Required parameters:
value = *state_no* (int)
state_no is the number of the state to change to.

Optional parameters:
ctrl = *ctrl_flag* (int)
ctrl_flag is the value to set the player's control flag to. 0 for no control, nonzero for control.

anim = *anim_no* (int)
This is the action number to switch to. If omitted, the player's animation will remain unchanged.

Example:

```
; Change to standing state, and give player control
type = ChangeState
value = 0
ctrl = 1
```

ClearClipboard

Erases any text currently on the player's clipboard.

Required parameters:
none

Optional parameters:
none

Example:
none

CtrlSet

Sets the player's control flag.

Required parameters:
value = *ctrl_flag* (int)
Set to nonzero to have control, or 0 to disable control.

Optional parameters:
none

Example:
none

DefenceMulSet

Sets the player's defense multiplier. All damage the player takes is scaled by this amount.

Required parameters:
value = *defense_mul* (float)
Specifies the defense multiplier.

Optional parameters:
none

Notes:
The LifeAdd controller is not affected by the player's defense multiplier.

Example:

```
; All damage the player takes is reduced to half.
type = DefenceMulSet
value = .5
```

DestroySelf

If called by a helper-type character, DestroySelf causes that character to be removed from the field of play. DestroySelf is not valid for non-helper characters.

Required parameters:
none

Optional parameters:
none

Example:
none

DisplayToClipboard

This controller is only useful for debugging. DisplayToClipboard clears the player clipboard and prints a specified message to it. Display of the player clipboards is enabled in debug mode (press Ctrl+D).

Required parameters:
text = "*format_string*"
format_string must be encased in double-quotes. It is a printf format string, so if you know about printf, you can skip this description. The format string contains any text you wish to display. You can also use \n to generate a line break, and \t to generate a tab character (tab width is equivalent to 4 characters). To display the value of an arithmetic expression, you can put a %d (for ints) or a %f (for floats) in the format string, then specify the expression in the params list. To display a % character, you must put %% in the format string.

Only signed integer and floating-point format specifiers are accepted: %d, %i, %f, %F, %e, %E, %g, or %G. Length-modified format specifiers (e.g., %lld) are not supported. Recognized escape sequences are \n, \t, \\, and \".

Optional parameters:
params = *exp_1*, *exp_2*, *exp_3*, *exp_4*, *exp_5*, *exp_6*
Up to 6 numeric arguments can be specified in the format string. These should be listed under the params item, in order. The type of each parameter must match its format specifier. You cannot specify more or less parameters than are called for in the format string.

If there is a type mismatch between the format specifier and the parameter actually provided, then the actual value of the parameter will be shown in an appropriate form for that type, using default formatting options.

Example:

```
type = DisplayToClipboard
text="The value of var(17) is %d, which is %f%% of 23.\n\t--Kiwi."
params = var(17):=1,var(17)/.230

displays the following to the player's clipboard:
```

```
The value of var(17) is 1, which is 4.347826% of 23.
    --Kiwi.
```

EnvColor

Turns the whole screen a solid color, excepting foreground-layer animations like hit sparks and "ontop" explods. Foreground layers of the stage will not be visible.

Required parameters:

none

Optional parameters:

value = *col_r*, *col_g*, *col_b* (int)

Specifies the R, G, and B components of the color to set the screen to. Each component should be an integer between 0 and 255. The larger a component, the more of that color will appear in the environment color. The default is 255,255,255 (pure white).

time = *effective_time* (int)

Specifies how many ticks the environment color should be displayed. Defaults to 1 tick. Set to -1 to have the EnvColor persist indefinitely.

under = *under_flag* (int)

Set *under_flag* to 1 to have the environment color drawn under characters and projectiles. In other words, characters and projectiles will be visible on top of the colored backdrop. Defaults to 0.

Example:

none

EnvShake

Causes the screen to shake vertically.

Required parameters:

time = *shake_time* (int)

Specifies the number of ticks to shake the screen for.

Optional parameters:

freq = *shake_speed* (float)

shake_speed is a float between 0 (slow shake) to 180 (fast shake). Defaults to 60.

ampl = *shake_amplitude* (int)

The larger the amplitude, the farther the screen shakes up and down. A negative amplitude means that the screen will shake down first. Defaults to -4 in 240p, -8 in 480p, -16 in 720p.

phase = *phase_offset* (float)

Specifies the phase offset for the shaking. The default is 0, unless the frequency multiplier is 90 or greater. In this case, the default phase offset is 90.

Example:

none

Explod

The Explod controller is a flexible tool for displaying animations such as sparks, dust and other visual effects. Its functionality includes that of GameMakeAnim, which is now deprecated.

Required parameters:

anim = *[F]anim_no* (int)

anim_no specifies the number of the animation to play back. The 'F' prefix is optional: if included, then the animation is played back from fight.def.

Optional parameters:

ID = *id_no* (int)

id_no specifies an ID number for this explod. Used to identify particular explods in triggers and controllers that affect explods.

pos = *off_x*, *off_y* (int, int)
off_x and *off_y* specify the offset at which to create the explod. The exact behavior depends on the postype. If these parameters are omitted, they default to 0.

postype = *postype_string* (string)
postype_string specifies how to interpret the pos parameters. In all cases, a positive *off_y* means a downward displacement.

Valid values for *postype_string* are the following:

p1
Interprets pos relative to p1's axis. A positive *off_x* is toward the front of p1. This is the default value for postype. Refer to the note at the end of this controller's description.

p2
Interprets pos relative to p2's axis. A positive *off_x* is toward the front of p2. Refer to the note at the end of this controller's description.

front
Interprets *off_x* relative to the edge of the screen that p1 is facing toward, and *off_y* relative to the top of the screen. A positive *off_x* is to the right of the screen, whereas a negative *off_x* is toward the left.

back
Interprets *off_x* relative to the edge of the screen that p1 is facing away from, and *off_y* relative to the top of the screen. A positive *off_x* is toward the center of the screen, whereas a negative *off_x* is away from the center. For historical reasons, the offset behavior is inconsistent with postype = front.

left
Interprets *off_x* and *off_y* relative to the upper-left corner of the screen. A positive *off_x* is toward the right of the screen.

right
Interprets *off_x* and *off_y* relative to the upper-right corner of the screen. A positive *off_x* is toward the right of the screen.

facing = *facing* (int)
Set *facing* to 1 to have the explod face in the same direction as the positive *off_x* (as determined by postype), and -1 to have the explod face in the opposite direction. Defaults to 1.

vfacing = *vfacing* (int)
Set *vfacing* to -1 to have the explod display vertically flipped, or 1 to have the explod display vertically unflipped. Defaults to 1.

bindtime = *bind_time* (int)
Specifies the number of game ticks to bind the explod to the bind point specified by postype. For instance, if postype = p1, pos = 30, -40, and bindtime = 5, then the explod will be drawn at position 30, -40 relative to p1's axis for 5 ticks, no matter how p1 moves during this time. After the bindtime has expired, the explod will no longer be bound to the bind point, and will maintain its position (unless affected by the vel or accel parameters). If *bind_time* is -1, then the explod will be bound until the explod is removed or another controller affects the bindtime.

vel = *x_vel*, *y_vel* (float, float)
Specifies initial X and Y velocity components for the explod. These are interpreted relative to the explod's "facing" direction. These default to 0 if omitted.

accel = *x_accel*, *y_accel* (float, float)
Specifies X and Y acceleration components for the explod. These default to 0.

random = *rand_x*, *rand_y* (int, int)
Causes the explod's bind point to be displaced by a random amount when created. *rand_x* specifies the displacement range in the x direction, and *rand_y* specifies the displacement range in the y direction. For instance, if pos = 0,0 and random = 40,80, then the explod's x location will be a random number between -20 and 19, and its y location will be a random number between -40 and 39. Both arg1 and arg2 default to 0 if omitted.

removetime = *rem_time* (int)
If *rem_time* is positive, the explod will be removed after having been displayed for that number of game ticks. If *rem_time* is -1, the explod will be displayed indefinitely. If *rem_time* is -2, the explod will be removed when its animtime reaches 0. The default value is -2.

supermove = *bvalue* (boolean)
This parameter is deprecated -- use supermovetime parameter instead

Set supermove = 1 to have the explod persist until the end of a super pause, regardless of the value of removetime. Defaults to 0.

supermovetime = *move_time* (int)
Specifies the number of ticks that the explod will be unfrozen during a SuperPause. Used where you want the explod to be animated during a SuperPause, such as for custom super sparks. Defaults to 0.

pausemovetime = *move_time* (int)
Specifies the number of ticks that the explod should be unfrozen during a Pause. Defaults to 0.

scale = *x_scale*, *y_scale* (float, float)
x_scale and *y_scale* specify the scaling factors to apply to the explod in the horizontal and vertical directions. Both default to 1 (no scaling) if omitted.

sprpriority = *pr* (int)
pr specifies the drawing priority for the explod. Animations with higher priority get drawn over animations with lesser priority. For instances, setting sprpriority = -3 will cause the explod to be drawn under most characters and other explods, which usually have sprpriority >= -2. Defaults to 0 if omitted.

ontop = *bvalue* (boolean)
Set ontop = 1 to have the explod drawn over all other sprites and background layers. This parameter has precedence over sprpriority. Defaults to 0.

shadow = *shad_r*, *shad_g*, *shad_b* (int, int, int)
Specifies the R, G, and B components of the explod's shadow. Valid values for each component are 0-255. The greater a component, the less of that color will be displayed in the shadow. To use the shadow color of the stage, set shad_r to -1. Defaults to 0,0,0 (no shadow).

ownpal = *bvalue* (boolean)
Set ownpal = 1 to give the explod its own copy of its palette. This is desirable if you want to keep temporary changes to the player's palette (e.g. from using the PalFX controller) from affecting the color of the explod. Defaults to 0 if omitted.

removeongethit = *bvalue* (boolean)
Setting this to 1 will have the explod removed if the player gets hit. Defaults to 0.

ignorehitpause = *bvalue* (boolean)
If this is 1, the explod will be animated independently of the player that created it. If set to 0, it will not be updated when the player is in hitpause. Defaults to 1.

trans = *trans_type* (string)
Overrides the explod's animation transparency settings. See the Trans controller for details. An "alpha" parameter must be specified if trans_type is "addalpha". If omitted, does nothing.

Notes:
The position of an explod that is created relative to a player (eg. using "postype = p1") is determined only after all player updates have completed (contrast this to helpers, which are created relative to the player's immediate position when the controller was executed). This behavior is necessary to make explods bind properly to the player's screen position.

For example, assume the player has an x velocity of 5 and a position of (160,0). If an explod is created with an offset of 0,0 relative to p1, then the explod's actual screen position will be 165,0.

ExplodBindTime

Changes the position binding time of the player's explods.

Required parameters:
none

Optional parameters:
ID = *id_no* (int)
Only explods with ID number equal to *id_no* will have their position binding affected. Set ID to -1 to affect the binding of all explods. The default value is -1.

time = *binding_time* (int)
Specifies the number of ticks for which the explods should be bound to their binding points (defined at the time the explods were created.) Defaults to 1 tick. A time of -1 binds the explods indefinitely or until another controller changes the bindtime.

Alternate syntax:
value = *binding_time* may be used instead of time = *binding_time*.

Example:
none

ForceFeedback

Creates force feedback for supported force feedback devices. **This controller is not implemented in MUGEN 1.0.**

Parameters to the ForceFeedback controller may not be specified using arithmetic expressions. It is an exception in this regard.

Required parameters:
none

Optional parameters:
waveform = *wave_type* (string)
Valid waveforms are "sine", "square", "sinesquare", and "off". For the Dual Shock controller, a sine waveform corresponds to the large rumble motor, and a square waveform corresponds to the smaller buzzer motor. sinesquare, of course, corresponds to both motors simultaneously. Use "off" to turn off any force feedback that is currently executing. waveform defaults to sine.

time = *duration* (integer constant)
Specifies how long the force feedback should last, in ticks. Defaults to 60.

freq = *start* (integer constant), *d1*, *d2*, *d3* (float constants)
Force feedback frequency varies between 0 and 255. The formula used to determine force feedback frequency is start + *d1* * t + *d2* * t ** 2 + *d3* * t ** 3 where t represents the number of ticks elapsed since the force feedback was initiated. Defaults to freq = 128,0,0,0. Currently, the frequency parameter is completely ignored.

ampl = *start* (integer constant), *d1*, *d2*, *d3* (float constants)
Force feedback amplitude varies between 0 and 255. The formula used to determine force feedback frequency is start + *d1* * t + *d2* * t ** 2 + *d3* * t ** 3 where t represents the number of ticks elapsed since the force feedback was initiated. Defaults to ampl = 128,0,0,0

self = *self_flag* (boolean constant)
If *self_flag* is 1, then P1's pad will vibrate. If self is 0, then P2's pad will vibrate. Defaults to 1.

Example:
See common1.cns.

FallEnvShake

Shakes the screen using the fall.envshake parameters set by an attack (see HitDef controller). This controller is effective only if GetHitVar(fall.envshake.time) is not zero, and it sets GetHitVar(fall.envshake.time) to zero after being executed. This controller is used in common1.cns to shake the screen when a player falls, and is not normally useful otherwise.

Required parameters:

none

Optional parameters:
none

Example:
See common1.cns.

GameMakeAnim

Creates a game animation, like a hit spark or a super charging effect. This controller has been superseded by Explod and is now considered deprecated. Support for it may be removed in future versions.

Required parameters:
none

Optional parameters:
value = *anim_no* (int)
Specifies the animation number (from fightfx) of the animation to play. Defaults to 0.

under = *under_flag* (int)
If *under_flag* is 1, the animation is drawn behind the character sprites. Defaults to 0 (draw over characters).

pos = *x_pos*, *y_pos* (float)
Specifies the position to display the animation at, relative to the player axis. Defaults to 0,0.

random = *rand_amt* (int)
The position of the animation will be displaced in the x and y directions by (different) random amounts. The displacement can be as large as half of rand_amt. Defaults to 0.

Example:
none

Gravity

Accelerates the player downwards, using the value of the player's "yaccel" constant.

Required parameters:
none

Optional parameters:
none

Example:

```
; Applies constant acceleration throughout state
trigger1 = 1
type = Gravity
```

Helper

Creates another instance of the player as a helper character.

Required parameters:
none

Optional parameters:
helpertype = *type_string* (string)
This parameter is deprecated; player-type helpers are not supported. If helpertype = normal, then the helper will be allowed to move off the edge of the screen. Furthermore, the camera will not move to try to keep the helper on screen. If helpertype = player, then the helper will be constrained to the screen and will be followed by the camera, just like a normal player. Defaults to

normal. If you plan to use a helper for camera manipulation, do not use a player-type helper; instead use the ScreenBound controller in a normal helper with the "movecamera" parameter.

name = "*name_string*" (string)
Specifies a name for this helper, which must be enclosed in double quotes. If omitted, the name defaults to "<parent>'s helper", where <parent> represents the name of the player creating the helper.

ID = *id_no* (int)
Sets an ID number to refer to this helper by. Defaults to 0.

pos = *off_x*, *off_y* (int)
Specifies the x and y offsets to create this helper at. The precise meaning of these parameters is dependent on the postype. Defaults to 0,0.

postype = *postype_string* (string)
postype_string specifies the postype -- how to interpret the pos parameters. In all cases, a positive y offset means a downward displacement. In all cases, *off_y* is relative to the position of the player.

Valid values for *postype_string* are the following:

p1
Interprets offset relative to p1's axis. A positive *off_x* is toward the front of p1. This is the default value for postype.

p2
Interprets offset relative to p2's axis. A positive *off_x* is toward the front of p2. If p2 does not exist, the position is calculated with respect to p1 and a warning is logged.

front
Interprets *off_x* relative to the edge of the screen that p1 is facing toward. A positive *off_x* is away from the center of the screen, whereas a negative *off_x* is toward the center.

back
Interprets *off_x* relative to the edge of the screen that p1 is facing away from. A positive *off_x* is toward the center of the screen, whereas a negative *off_x* is away from the center.

left
Interprets *off_x* relative to the left edge of the screen. A positive *off_x* is toward the right of the screen.

right
Interprets *off_x* relative to the right edge of the screen. A positive *off_x* is toward the right of the screen.

facing = *facing* (int)
If postype is left or right, setting *facing* to 1 will make the helper face the right, and a value of -1 makes the helper face left. For all other values of postype except p2, if *facing* is 1, the helper will face the same direction as the player. If *facing* is -1, the helper will face the opposite direction. In the case of postype = p2, *facing* has the same effect as above, except it is with respect to p2's facing. Defaults to 1.

stateno = *start_state* (int)
Determines the state number that the helper starts off in. Defaults to 0.

keyctrl = *ctrl_flag* (boolean)
If *ctrl_flag* is 1, then the helper is able to read command input from the player (e.g., the keyboard or joystick). Also, the helper will inherit its root's State -1. If *ctrl_flag* is 0, then the helper does not have access to command input, and does not inherit State -1. The default value of *ctrl_flag* is 0.

ownpal = *pal_flag* (boolean)
If *pal_flag* is 0, the helper will inherit its parent's palette. If the parent's palette is temporarily changed (eg. by a PalFX controller), the changes will be reflected in the helper too. If *pal_flag* is 1, the helper will receive its own working palette, that is independent of its parent's. Defaults to 0.

supermovetime = *move_time* (int)
Specifies the number of ticks that the helper should be unfrozen during a SuperPause. Defaults to 0.

pausemovetime = *move_time* (int)

Determines the number of ticks that the helper should be unfrozen during a Pause. Defaults to 0.

size.xscale (float)
See below.

size.yscale (float)
See below.

size.ground.back (int)
See below.

size.ground.front (int)
See below.

size.air.back (int)
See below.

size.air.front (int)
See below.

size.height (int)
See below.

size.proj.doscale (int)
See below.

size.head.pos (int,int)
See below.

size.mid.pos (int,int)
See below.

size.shadowoffset (int)
These parameters have the same meaning as the corresponding parameters in the root's CNS file. You can specify one or more of these parameters to change it to a value suitable for this helper. Otherwise, they default to the values inherited from the parent.

Example:
none

HitAdd

Adds to the current combo counter.

Required parameters:
value = *add_count* (int)
add_count specifies the number of hits to add to the current combo counter.

Optional parameters:
none

Example:
none

HitBy

Temporarily specifies the types of hits that are be allowed hit to the player.

Required parameters:
value = *attr_string* OR value2 = *attr_string*
Only one of the above parameters can be specified. *attr_string* should be a standard hit attribute string. See Details.

Optional parameters:
time = *effective_time* (int)
Specifies the number of game ticks that these HitBy attributes should be effective for. Defaults to 1.

Details:

The player has two hit attribute slots, which can be set using the value or value2 parameters to the HitBy controller. These slots can also be set by the NotHitBy controller. When a slot is set, it gets a timer (the effective time) which counts down toward zero. If the timer has not yet reached zero, the slot is considered to be active. The player can be hit by a HitDef only if that HitDef's attribute appears in all currently active slots. Using the HitBy controller sets the specified slot to contain only those hit attributes which appear in the HitBy attribute string.

Example:

```
; Can be hit only by standing normal attacks
trigger1 = 1
type = HitBy
value = S, NA
```

HitDef

Defines a single hit of the player's attack. If the player's Clsn1 box (red) comes in contact with his opponent's Clsn2 box (blue), and the HitDef was define on or before that particular point in time, then the specified effect will be applied. This is one of the more complex, but most commonly-used controllers. A single HitDef is valid only for a single hit. To make a move hit several times, you must trigger more than one HitDef during the attack.

Required parameters:

attr = *hit_attribute* (string)

This is the attribute of the attack. It is used to determine if the attack can hit P2. It has the format:

attr = *arg1*, *arg2*

Where: *arg1* is either "S", "C" or "A". Similar to "statetype" for the StateDef, this says whether the attack is a standing, crouching, or aerial attack.

arg2 is a 2-character string. The first character is either "N" for "normal", "S" for "special", or "H" for "hyper" (or "super", as it is commonly known). The second character must be either "A" for "attack" (a normal hit attack), "T" for "throw", or "P" for projectile.

Optional parameters:

hitflag = *hit_flags* (string)

This determines what type of state P2 must be in for P1 to hit. *hit_flags* is a string containing a combination of the following characters:

"H" for "high", "L" for "low" or "A" for air. "M" (mid) is equivalent to saying "HL". "F" is for fall, and if included will allow P1 to juggle falling opponents in the air. "D" is for "lying Down", and if included allows P1 to hit opponents lying down on the ground. "H", "L" or "A" (or "M") must be present in the hitflag string.

Two optional characters are "+" and "-". If "+" is added, then the hit only affects people in a gethit state. This may be useful for chain-moves that should not affect opponents who were not hit by the first move in the chain attack. If "-" is added, then the hit only affects players that are NOT in a gethit state. You should use "-" for throws and other moves you do not want P1 to be able to combo into. "+" and "-" are mutually exclusive, ie. cannot be used at the same time.

If omitted, this defaults to "MAF".

guardflag = *hit_flags* (string)

This determines how P2 may guard the attack. hit_flags is a string containing a combination of the following characters:

"H" for "high", "L" for "low" or "A" for air. "M" (mid) is equivalent to saying "HL". If omitted, defaults to an empty string, meaning P2 cannot guard the attack.

affectteam = *team_type* (string)

team_type specifies which team's players can be hit by this HitDef. Use B for both teams (all players), E for enemy team (opponents), or F for friendly team (your own team). The default is E.

animtype = *anim_type* (string)
This refers to the type of animation that P2 will go into when hit by the attack. Choose from "light", "medium", "hard", "back", "up", or "diagup". The first three are self-explanatory. "Back" is the animation where P2 is knocked off her feet. "Up" should be used when the character is knocked straight up in the air (for instance, by an uppercut), and "DiagUp" should be used when the character is knocked up and backwards in the air, eventually landing on his head. The default is "Light".

air.animtype = *anim_type* (string)
Similar to the "animtype" parameter, this is the animtype to set P2 to if P2 is in the air, instead of on the ground. Defaults to the same value as the "animtype" parameter if omitted.

fall.animtype = *anim_type* (string)
Similar to the "animtype" parameter, this is the animtype to set P2 to if P2 is hit while falling. Defaults to Up if air.animtype is Up, or Back otherwise.

priority = *hit_prior* (int), *hit_type* (string)
Specifies the priority for this hit. Hits with higher priorities take precedence over hits with lower priorities. Valid values for *hit_prior* are 1-7. Defaults to 4.

hit_type, if specified, gives the priority class of the hit. Valid priority classes are Dodge, Hit, and Miss. The priority class determines the tiebreaking behavior when P1 and P2 hit each other simultaneously with equal priorities. The behavior for P1 vs. P2 is as follows:

- Hit vs. Hit: both P1 and P2 are hit
- Hit vs. Miss: P1 hits, P2 misses
- Hit vs. Dodge: Both miss
- Dodge vs. Dodge: Both miss
- Dodge vs. Miss: Both miss
- Miss vs. Miss: Both miss

In the case of a no-hit tie, the respective HitDefs stay enabled. "Miss" or "Dodge" are typically used for throws, where P1 and P2 should not be able to simultaneously hit each other. The default for *hit_type* is "Hit".

damage = *hit_damage*, *guard_damage* (int, int)
hit_damage is the damage that P2 takes when hit by P2. The optional *guard_damage* parameter is the damage taken by P2 if the hit is guarded. Both default to zero if omitted.

pausetime = *p1_pausetime*, *p2_shaketime* (int, int)
This is the time that each player will pause on the hit. *p1_pausetime* is the time to freeze P1, measured in game-ticks. *p2_pausetime* is the time to make P2 shake before recoiling from the hit. Defaults to 0,0 if omitted.

guard.pausetime = *p1_pausetime*, *p2_shaketime* (int, int)
Similar to the "pausetime" parameter, these are the times to pause each player if the hit was guarded. Defaults to the same values as the "pausetime" parameter if omitted.

sparkno = *action_no* (int)
This is the action number of the spark to display if the hit is successful. To play a spark out of the player's .AIR file, precede the action number with an S, e.g. "sparkno = S10". Defaults to the value set in the player variables if omitted.

guard.sparkno = *action_no* (int)
This is the action number of the spark to display if the hit was guarded. To play a spark out of the player's .AIR file, precede the action number with an S. Defaults to the value set in the player variables if omitted.

sparkxy = *spark_x*, *spark_y* (int, int)
This is where to make the hit/guard spark. *spark_x* is a coordinate relative to the front of P2. A negative value makes the spark deeper inside P2. "Front" refers to the x- position at P2's axis offset towards P1 by the corresponding width value in the [Size] group in P2's player variables. *spark_y* is relative to P1. A negative value makes a spark higher up. You can use a tool like AirView to determine this value by positioning the cursor at the "attack spot" and reading off the value of the y-position. Defaults to 0,0 if omitted.

hitsound = *snd_grp*, *snd_item* (int, int)
This is the sound to play on hit (from common.snd). The included fight.snd lets you choose from 5,0 (light hit sound) through to 5,4 (painful whack). To play a sound from the player's own SND file, precede the first number with an "S". For example, "hitsound = S1,0". Defaults to the value set in the player variables if omitted.

guardsound = *snd_grp*, *snd_item* (int, int)
This is the sound to play on guard (from common.snd). Only 6,0 is available at this time. To play a sound from the player's own SND file, precede the first number with an "S". There is no facility to play a sound from the opponent's SND file. Defaults to the value set in the player variables if omitted.

ground.type = *attack_type* (string)
This is the kind of attack if P2 is on the ground. Choose from: - "High": for attacks that make P2's head snap backwards. - "Low": for attacks that hit P2 in the stomach. - "Trip": for low sweep attacks. If you use "Trip" type, the ground.velocity parameter should have a non-zero y-velocity, and the fall parameter should be set to 1. A tripped opponent does not bounce upon falling on the ground. - "None": for attacks that do nothing besides pause P1 and P2 for the duration in the pausetime parameter.

If P2 is hit from behind, "High" will be displayed as "Low" and vice-versa. P2's animation for "High" and "Low" types will be superseded if the AnimType parameter is "Back". Defaults to "High" if omitted.

air.type = *attack_type* (string)
This is the kind of attack if P2 is in the air. Defaults to the same value as "ground.type" if omitted.

ground.slidetime = *slide_time* (int)
This is the time in game-ticks that P2 will slide back for after being hit (this time does not include the pausetime for P2). Applicable only to hits that keep P2 on the ground. Defaults to 0 if omitted.

guard.slidetime = *slide_time* (int)
Same as "ground.slidetime", but this is the value if P2 guards the hit. Defaults to same value as "guard.hittime".

ground.hittime = *hit_time* (int)
Time that P2 stays in the hit state after being hit. Adjust this value carefully, to make combos possible. Applicable only to hits that keep P2 on the ground. Defaults to 0 if omitted.

guard.hittime = *hit_time* (int)
Same as "ground.hittime", but this is the value if P2 guards the hit. Defaults to same value as "ground.hittime".

air.hittime = *hit_time* (int)
Time that p2 stays in the hit state after being hit in or into the air, before being able to guard again. This parameter has no effect if the "fall" parameter is set to 1. Defaults to 20 if omitted.

guard.ctrltime = *ctrl_time* (int)
This is the time before p2 regains control in the ground guard state. Defaults to the same value as "guard.slidetime" if omitted.

guard.dist = *x_dist* (int)
This is the x-distance from P1 in which P2 will go into a guard state if P2 is holding the direction away from P1. Defaults to the value in the player variables if omitted. You normally do not need to use this parameter.

yaccel = *accel* (float)
Specifies the y acceleration to impart to P2 if the hit connects. Defaults to .35 in 240p, .7 in 480p, 1.4 in 720p.

ground.velocity = *x_velocity*, *y_velocity* (float, float)
Initial velocity to give P2 after being hit, if P2 is on the ground. If *y_velocity* is not zero, P2 will be knocked into the air. Both values default to 0 if omitted. You can leave out the *y_velocity* if you want P2 to remain on the ground.

guard.velocity = *x_velocity* (float)
Velocity to give P2 if P2 guards the hit on the ground. Defaults to the *x_velocity* value of the "ground.velocity" parameter if omitted.

air.velocity = *x_velocity*, *y_velocity* (float, float)

Initial velocity to give P2 if P2 is hit in the air. Defaults to 0,0 if omitted.

airguard.velocity = *x_velocity*, *y_velocity* (float float)
Velocity to give P2 if P2 guards the hit in the air. Defaults to *x_velocity* * 1.5, *y_velocity* / 2, where *x_velocity* and *y_velocity* are values of the "air.velocity" parameter.

ground.cornerpush.veloff = *x_velocity* (float)
Determines the additional velocity (velocity offset) to impart to the player if he lands a ground hit in the corner. Setting this to a higher value will cause the player to be "pushed back" farther out of the corner. If omitted, default value depends on the attr parameter. If arg1 of attr is "A", default value is 0. Otherwise, defaults to 1.3 * guard.velocity.

air.cornerpush.veloff = *x_velocity* (float)
Determines the additional velocity (velocity offset) to impart to the player if he lands a hit to an aerial opponent in the corner. Setting this to a higher value will cause the player to be "pushed back" farther out of the corner. Defaults to ground.cornerpush.veloff if omitted.

down.cornerpush.veloff = *x_velocity* (float)
Determines the additional velocity (velocity offset) to impart to the player if he lands a hit on a downed opponent in the corner. Setting this to a higher value will cause the player to be "pushed back" farther out of the corner. Defaults to ground.cornerpush.veloff if omitted.

guard.cornerpush.veloff = *x_velocity* (float)
Determines the additional velocity (velocity offset) to impart to the player if his hit is guarded in the corner. Setting this to a higher value will cause the player to be "pushed back" farther out of the corner. Defaults to ground.cornerpush.veloff if omitted.

airguard.cornerpush.veloff = *x_velocity* (float)
Determines the additional velocity (velocity offset) to impart to the player if his hit is guarded in the corner. Setting this to a higher value will cause the player to be "pushed back" farther out of the corner. Defaults to guard.cornerpush.veloff if omitted.

airguard.ctrltime = *ctrl_time* (int)
This is the time before p2 regains control in the air guard state. Defaults to the same value as "guard.ctrltime" if omitted.

air.juggle = *juggle_points* (int)
The amount of additional juggle points the hit requires. Not to be confused with the "juggle" parameter in the StateDef. You typically do not need this parameter, except for HitDefs of projectiles. Defaults to 0 if omitted.

mindist = *x_pos*, *y_pos* (int, int)
See below.

maxdist = *x_pos*, *y_pos* (int, int)
These let you control the minimum and maximum distance of P2 relative to P1, after P2 has been hit. These parameters are not commonly used. Defaults to no change in P2's position if omitted.

snap = *x_pos*, *y_pos* (int, int)
This moves P2 to the specified position relative to P1 if hit. This parameter is not normally used. If you want to snap P2 to a particular position for a throw, it is recommended you use a "TargetBind" controller in P1's throwing state instead. Defaults to no change in P2's position if omitted.

p1sprpriority = *drawing_priority* (int)
This is the drawing priority of P1's sprite if the move hits or is guarded by P2. Together with the p2sprpriority parameter, it controls whether or not P1 is drawn in front of or behind P2. The default value is 1.

p2sprpriority = *drawing_priority* (int)
This is the drawing priority of P2's sprite if the move hits or is guarded by P2. The default value is 0.

p1facing = *facing* (int)
Set to -1 to make P1 turn around if the hit is successful. The default value is no change in where P1 is facing.

p1getp2facing = *facing* (int)
Set to 1 to have P1 face in the same direction as P2 is facing after the hit connects, and -1 to have P1 face the opposite direction from P2. Defaults to 0 (no change). If nonzero, this parameter takes precedence over p1facing.

p2facing = *facing* (int)
Set to 1 to make P2 face the same direction as P1 if the hit is successful, -1 to make P2 face away. The default value is 0, no change in where P2 is facing.

p1stateno = *state_no* (int)
This is the number of the state to set P1 to if the hit is successful. The state must be an attack state (movetype = A) for at least 1 tick. Used mainly for throws. Defaults to -1, no change.

p2stateno = *state_no* (int)
This is the number of the state to set P2 to if the hit is successful. P2 will get P1's state and animation data. P2 will retain P1's states and animation data until P2 is hit, or a SelfState controller is used to return P2 to his own states. The state must be a get-hit state (movetype = H) for at least 1 tick. Used mainly for throws; can also be used for custom hit reactions. Defaults to -1, no change.

p2getp1state = *bvalue* (boolean)
Set to 0 to prevent P2 from getting P1's state and animation data, in case you do not want that default behaviour of the "p2stateno" parameter. Defaults to 1 if the "p2stateno" parameter is used. Ignored otherwise.

forcestand = *bvalue* (boolean)
Set to 1 to force P2 to a standing state-type if the hit is successful, and P2 is in a crouching state. Has no effect if P2 is in an air state. Normally defaults to 0, but if the y_velocity of the "ground.velocity" parameter is non-zero, it defaults to 1.

fall = *bvalue* (boolean)
Set to 1 if you want P2 to go into a "fall" state (where P2 hits the ground without regaining control in the air). Use if you want a move to "knock down" P2. Defaults to 0.

fall.xvelocity = *x_velocity* (float)
This is the x-velocity that P2 gets when bouncing off the ground in the "fall" state. Defaults to no change if omitted.

fall.yvelocity = *y_velocity* (float)
This is the y-velocity that P2 gets when bouncing off the ground in the "fall" state. Defaults to -4.5 in 240p, -9 in 480p, -18 in 720p.

fall.recover = *bvalue* (boolean)
Set to 0 if you do not want P2 to be able to recover from the "fall" state. Defaults to 1 if omitted (can recover).

fall.recovertime = *recover_time* (int)
This is the time that must pass before P2 is able to recover from the "fall" state by inputting his recovery command. Does not include the time that P2 is paused for while shaking from the hit. Defaults to 4 if omitted.

fall.damage = *damage_amt* (int)
Indicates the amount of damage to deal when P2 hits the ground out of a falling state. Defaults to 0 if omitted.

air.fall = *bvalue* (boolean)
Set to 1 if you want P2 to go into a "fall" state (where P2 hits the ground without regaining control in the air) if hit while P2 is in the air. Defaults to the same value as fall.

forcenofall = *bvalue* (boolean)
Set to 1 to force P2 out of a "fall" state, if he is in one. This parameter has no effect on P2 if he is not in a "fall" state. This parameter is ignored if the "fall" parameter is set to 1. Defaults to 0 if omitted.

down.velocity = *x_velocity*, *y_velocity* (float, float)
This is the velocity to assign P2 if P2 is hit while lying down. If the *y_velocity* is non-zero, P2 will be hit into the air. If it is zero, then P2 will slide back on the ground. Defaults to the same values as the "air.velocity" parameter if omitted.

down.hittime = *hit_time* (int)
This is the time that P2 will slide back for if P2 is hit while lying down. This parameter is ignored if the *y_velocity* is non- zero for the "down.velocity" parameter.

down.bounce = *bvalue* (boolean)

Set to 1 if you want P2 to bounce off the ground one time (using the fall.xvelocity and fall.yvelocity values) after hitting the ground from the hit. This parameter is ignored if the *y_velocity* is zero for the "down.velocity" parameter. Defaults to 0 if omitted (P2 hits the ground and stays there).

id = *id_number* (int)
: Idetifier for the hit. Used for chain moves. You can use this number to later detect if a player was last hit by this particular HitDef. This number is called the targetID. It is used in controllers such as TargetBind, or in the target(ID) redirection keyword. Valid values are all values >= 1. If omitted, defaults to 0 (no ID). TargetID is not to be confused with PlayerID.

chainID = *id_number* (int)
: Main use of this is for chain moves. If P2 was last hit by a move by P1 with this ID, only then can he be hit by the HitDef with this chainID. You can use this in the following parts of a chain move. Note that chain moves are still possible even without the use of the "id" and "chainid" parameters. Valid values are all values >= 1. If omitted, defaults to -1 (chain from any hit).

nochainID = *nochain_1*, *nochain_2* (int)
: nochainID specifies up to 2 ID numbers of hits which cannot be chained into this hit. If these are -1 (the default), then chaining is not explicitly disabled for any hit ID numbers. nochain_2 can be omitted. Except for -1, the values specified must not coincide with the value for chainID. This parameter has no effect if P2 is hit by a third party between P1's previous HitDef and the current HitDef.

hitonce = *hitonce_flag* (boolean)
: If set to 1, the HitDef only affects one opponent. If the hit is successful, all other targets will be dropped. Normally defaults to 0. The exception is if the "attr" parameter is a throw type, which makes it default to 1.

kill = *bvalue* (boolean)
: Set to 0 if this hit should not be able to KO the opponent when the hit is successful. Defaults to 1.

guard.kill = *bvalue* (boolean)
: Set to 0 if this hit should not be able to KO the opponent when the hit is guarded. Defaults to 1.

fall.kill = *bvalue* (boolean)
: Set to 0 to prevent this attack from KO'ing the opponent when he falls on the ground (see fall.damage). Defaults to 1.

numhits = *hit_count* (int)
: *hit_count* indicates how many hits this hitdef should add to the combo counter. Must be 0 or greater. Defaults to 1.

getpower = *p1power*, *p1gpower* (int, int)
: p1power specifies the amount of power to give P1 if this HitDef connects successfully. p1gpower specifies the amount of power to give P1 if this HitDef is guarded. If omitted, *p1power* defaults to *hit_damage* (from "damage" parameter) multiplied by the value of Default.Attack.LifeToPowerMul specified in data/mugen.cfg. If *p1gpower* is omitted, it defaults to the value specified for *p1power* divided by 2.

givepower = *p2power*, *p2gpower* (int, int)
: p2power specifies the amount of power to give P2 if this HitDef connects successfully. p2gpower specifies the amount of power to give P2 if this HitDef is guarded. If omitted, p1power defaults to *hit_damage* (from "damage" parameter) multiplied by the value of Default.GetHit.LifeToPowerMul specified in data/mugen.cfg. If *p1gpower* is omitted, it defaults to the value specified for *p1power* divided by 2.

palfx.time = *palfx_time* (int)
: See below.

palfx.mul = *r1*, *g1*, *b1* (int, int, int)
: See below.

palfx.add = *r2*, *g2*, *b2* (int, int, int)
: If included, this allows for palette effects on P2 if the hit is successful. *palfx_time* is the time in game-ticks to apply palette effects on P2. *palfx_time* is 0 by default (no effect). The rest of the parameters are the same as in the PalFX controller.

envshake.time = *envshake_time* (int)
: See below.

envshake.freq = *envshake_freq* (float)
See below.

envshake.ampl = *envshake_ampl* (int)
See below.

envshake.phase = *envshake_phase* (float)
If included, this shakes the screen if the hit is successful. *envshake_time* is the time in game-ticks to shake the screen. The rest of the parameters are the same as in the EnvShake controller.

fall.envshake.time = *envshake_time* (int)
See below.

fall.envshake.freq = *envshake_freq* (float)
See below.

fall.envshake.ampl = *envshake_ampl* (int)
See below.

fall.envshake.phase = *envshake_phase* (float)
Similar to the envshake.* parameters, except the effects are applied only when P2 hits the ground.

Notes:
The behavior of HitDef is undefined when executed from a [Statedef -2] block while the player has another player's state and animation data.

Example:
none

HitFallDamage

When the player has been hit and is in a falling state, apply damage from the fall (specified in the hitdef) to the player.

Required parameters:
none

Optional parameters:
none

Example:
none

HitFallSet

When the player has been hit, sets the player's fall variables.

Required parameters:
none

Optional parameters:
value = *fallset_flag* (int)
If *fallset_flag* is -1, then this controller does not change whether the player will fall or not. A *fallset_flag* of 0 means that the player should not fall, and a 1 means that he should. Defaults to -1.

xvel = *x_velocity* (float)
See below.

yvel = *y_velocity* (float)
If specified, sets the player's fall.xvel and fall.yvel parameters, respectively. See HitDef for a description of these parameters.

Example:
none

HitFallVel

If the player has been hit and is in a falling state, sets the player's velocities to the fall velocities (fall.xvel and fall.yvel) specified in the HitDef.

Required parameters:
none

Optional parameters:
none

Example:
none

HitOverride

Defines a hit override. If the player is hit by an attack of the specified type, he will go to the specified state number instead of his default gethit behavior. Up to 8 hit overrides can be active at one time.

Required parameters:
attr = *attr_string* (string)
Standard hit attribute string specifying what types of hits to override. See HitDef's description for the "attr" parameter.

stateno = *value* (int)
Specifies which state to go into if hit by a HitDef with the specified attributes.

Optional parameters:
slot = *slot_no* (int)
Specifies a slot number (0 to 7) to place this hit override in. Defaults to 0 if omitted.

time = *effective_time* (int)
Specifies how long this hit override should be active. Defaults to 1 (one tick). Set this to -1 to have this override last until overwritten by another one.

forceair = *value* (boolean)
If set to 1, the player's gethit variables will be set as if he was in an aerial state when hit. Useful if you want to force the player to fall down from any hit. Defaults to 0 if omitted.

Notes:
If P1 has one or more active HitOverrides, P1 will not be affected by any of P2's matching HitDefs that have any of the following characteristics:

- p1stateno parameter value is not -1
- p2getp1state parameter value is 1

Example:
none

HitVelSet

This controller is deprecated.

When the player has been hit, sets the desired components of the player's velocity to the appropriate gethit velocities.

Required parameters:
none

Optional parameters:
x = *x_flag* (int)
A nonzero flag means to change that x-component of the player's velocity to the gethit velocity.

y = *y_flag* (int)
A nonzero flag means to change that y-component of the player's velocity to the gethit velocity.

Example:
none

Notes:
Obsolete.

LifeAdd

Adds the specified amount to the player's life, scaled by the player's defense multiplier if necessary.

Required parameters:
value = *add_amt* (int)
Specifies amount of life to add to the player's life bar.

Optional parameters:
kill = *kill_flag* (int)
If *kill_flag* is 0, then the addition will not take the player below 1 life point. Defaults to 1.

absolute = *abs_flag* (int)
If *abs_flag* is 1, then exactly *add_amt* is added to the player's life (the defense multiplier is ignored). Defaults to 0.

Example:
none

LifeSet

Sets the player's life to the specified value.

Required parameters:
value = *life_amt* (int)
Specifies amount of life that the player will have after execution.

Optional parameters:
none

Example:
none

MakeDust

This controller is deprecated; use the Explod controller.

Creates dust effects.

Required parameters:
none

Optional parameters:
pos = *x_pos*, *y_pos* (int)
Specifies the position that the dust should be drawn at, relative to the player's axis. Defaults to 0,0.

pos2 = *x_pos*, *y_pos* (float)
Specifies the position to simultaneously draw a second dust cloud at. If omitted, the second dust cloud is not drawn.

spacing = *value* (int)
Determines the number of frames to wait between drawing dust clouds. For instance, spacing = 3 (the default) will draw a new cloud of dust every third frame. spacing should be 1 or greater.

Example:
none

ModifyExplod

Modifies the parameters of an existing Explod. Syntax is basically the same as Explod. However, this controller is subject to future change. Any code relying on this controller is not guaranteed to work in the future.

Notes:
The pos and postype parameters must be specified together.

MoveHitReset

Resets the movehit flag to 0. That is, after executing MoveHitReset, the triggers MoveContact, MoveGuarded, and MoveHit will all return 0.

Required parameters:
none

Optional parameters:
none

Example:
none

NotHitBy

Temporarily specifies types of hits that are not allowed to hit the player.

Required parameters:
value = *attr_string* OR value2 = *attr_string*
Only one of the above parameters can be specified. *attr_string* should be a standard hit attribute string. See details.

Optional parameters:
time = *effective_time* (int)
Specifies the number of game ticks that these NotHitBy attributes should be effective for. Defaults to 1.

Details:
The player has two hit attribute slots, which can be set using the "value" or "value2" parameters to the NotHitBy controller. These slots can also be set by the HitBy controller. When a slot is set, it gets a timer (the effective time) which counts down toward zero. If the timer has not yet reached zero, the slot is considered to be active. The player can be hit by a HitDef only if that HitDef's attribute appears in all currently active slots. Using the NotHitBy controller sets the specified slot to contain all hit attributes except those specified in the NotHitBy attribute string.

Example:

```
; Not hit by anything
trigger1 = 1
type = NotHitBy
value = SCA

; Not hit by normal attacks, and all projectiles
trigger1 = 1
type = NotHitBy
value = , NA, AP
```

Null

Does nothing. May be used for temporarily disabling other state controllers by simply changing their type to Null.

Required parameters:

none

Optional parameters:
none

Offset

Changes the player's display offset. The player is drawn shifted from his axis by this amount.

Required parameters:
none

Optional parameters:
x = *x_val* (float)
See below.

y = *y_val* (float)
Specifies the x and y offsets, respectively. You can specify one or both of the optional parameters.

Example:
none

PalFX

Applies temporary effects the player's palette. These will also affect the palette of any explods and helpers the player owns, unless they have specified ownpal = 1.

Required parameters:
none

Optional parameters:
time = *duration* (int)
Specifies the number of ticks that the palette effects should last. Specify -1 to have the palette effects last indefinitely. Specify 0 to stop any ongoing palette effects.

add = *add_r*, *add_g*, *add_b* (int, int, int)
See below.

mul = *mul_r*, *mul_g*, *mul_b* (int, int, int)
Each add component is added to the appropriate component of the player's palette, and the result is multiplied by the appropriate mul component divided by 256. For instance, if *pal_r* is the red component of the character's original palette, then the new red component is (*pal_r* + *add_r*) * *mul_r* / 256. The values for mul must be >= 0. The defaults for these parameters are for no change, i.e. add = 0,0,0 and mul = 256,256,256.

sinadd = ampl_r, ampl_g, ampl_b, period (int, int, int, int)
Creates an additional sine-wave palette addition effect. Period specifies the period of the sine wave in game ticks, and the amplitude parameters control the amplitude of the sine wave for the respective components. For instance, if t represents the number of ticks elapsed since the activation of the PalFX controller, and *pal_r* is the red component of the character's original palette, then the red component of the character's palette at time t is (*pal_r* + *add_r* + *ampl_r* * sin(2 * pi * t / *period*)) * *mul_r* / 256.

invertall = *bvalue* (bool)
If *bvalue* is non-zero, then the colors in the palette will be inverted, creating a "film negative" effect. Color inversion is applied before effects of add and mul. bvalue defaults to 0.

color = *value* (int)
This affects the color level of the palette. If value is 0, the palette will be greyscale. If *value* is 256, there is no change in palette. Values in between will have an intermediate effect. This parameter's effects are applied before invertall, add and mul. Values must be in range 0 to 256. Default value is 256.

Example:
none

ParentVarAdd

If the player is a helper, adds to one of the player's parent's working variables. Either a float variable or an int variable can be added to by this controller. If the player is not a helper, this controller does nothing.

Required parameters (int version):

v = *var_no* (int)

var_no should evaluate to an integer between 0 and 59.

value = *int_expr* (int)

int_expr is the value to add to the int variable indicated by *var_no*.

Required parameters (float version):

fv = *var_no* (int)

var_no should evaluate to an integer between 0 and 39.

value = *float_expr* (float)

float_expr is the value to add to the float variable indexed by *var_no*.

Optional parameters:

none in both cases

Alternate syntax:

var(*var_no*) = *int_expr* (int version)

fvar(*var_no*) = *float_expr* (float version)

Notes:

Due to historical reasons, note that the alternate VarAdd syntax listed above matches neither the syntax for variable assignment within an expression, nor the syntax for variable addition within an expression.

If you have placed P2 in a custom state through a successful hit, do not use variable assignment within the custom states. Otherwise, you will overwrite P2's parent's variables, which can cause unintended malfunction of the opponent player.

Example:

none

ParentVarSet

If the player is a helper, sets one of the parent's working variables. Either a float variable or an int variable can be set by this controller. Does nothing if the player is not a helper.

Required parameters (int version):

v = *var_no* (int)

var_no should evaluate to an integer between 0 and 59.

value = *int_expr* (int)

int_expr is the value to assign to the int variable indicated by *var_no*.

Required parameters (float version):

fv = *var_no* (int)

var_no should evaluate to an integer between 0 and 39.

value = *float_expr* (float)

float_expr is the value to assign to the float variable indexed by *var_no*.

Optional parameters:

none in both cases

Alternate syntax:

var(*var_no*) = *int_expr* (int version)

fvar(*var_no*) = *float_expr* (float version)

Notes:

Due to historical reasons, note that the alternate variable assignment syntax listed above does not exactly match the syntax for variable assignment within an expression.

If you have placed P2 in a custom state through a successful hit, do not use variable assignment within the custom states. Otherwise, you will overwrite P2's parent's variables, which can cause unintended malfunction of the opponent player.

Example:
none

Pause

Pauses the game for the specified amount of time. Player and background updates are not carried out during this time.

Required parameters:
time = t (int)
This is the number of game ticks to pause for. Valid values for t are all positive numbers, starting from 0.

Optional parameters:
endcmdbuftime = bt (int)
This is the number of ticks during the end of the pause in which the player's move commands will be buffered. Buffered commands will be detected by the "command" trigger immediately after the pause ends. The buffering applies only to players who are unable to move during the pause (see movetime parameter). Valid values for endcmdbuftime are from 0 to t, where t is the value of the time parameter. Defaults to 0.

movetime = mt (int)
This is the number of ticks during the start of the pause in which the player is allowed to move. Collision detection is carried out during this time, so it is possible to hit other players. Valid values for mt are from 0 to t, where t is the value of the time parameter. Defaults to 0.

pausebg = p (boolean)
If set to 1, the background is stopped during the pause. If 0, the background continues updating during the pause. Defaults to 1.

Notes:
Executing a Pause controller during the pausetime of another will cancel out the effect of the previous Pause controller. Executing a Pause during a superpause will delay the effects of the pause until after the superpause has ended.

Example:
none

PlayerPush

Disables the player's push checking for one tick. Push checking keeps players from overlapping one another. By temporarily disabling push checking, dodge-type moves in which the player passes through another (but can still be hit) can be implemented.

Required parameters:
value = $push\_flag$ (boolean)
If $push\_flag$ is nonzero, then push checking is enabled. If $push\_flag$ is zero, then push checking is disabled.

Optional parameters:
none

Example:
none

PlaySnd

Plays back a sound.

Required parameters:

value = *group_no*, *sound_no* (int, int)

group_no and *sound_no* correspond to the identifying pair that you assigned each sound in the player's snd file. To play back a sound from "common.snd", precede group_no with an "F".

Optional parameters:

volumescale = *volume_scale* (float)

volume_scale controls the volume of the sound. A value of 100 specifies 100% volume, 50 for 50%, and so on. Valid values are from 0 to 100. Defaults to 100.

channel = *channel_no* (int)

channel_no specifies which of the player's sound channels the sound should play on. Only one voice may play on a particular channel at a time. For example, if you play a sound on channel 2, then play any sound on the same channel before the first sound is done, then by default the first sound is stopped as the second one plays. 0 is a special channel reserved for player voices. Channel 0 voices are stopped when the player is hit. It's recommended you play your character's voice sounds on channel 0. If omitted, *channel_no* defaults to -1, meaning the sound will play on any free channel.

lowpriority = *pr* (int)

This is only valid if the channel is not -1. If *pr* is nonzero, then a sound currently playing on this sound's channel (from a previous PlaySnd call) cannot be interrupted by this sound.

freqmul = *f* (float)

The sound frequency will be multiplied by *f*. For example. Setting *f* to 1.1 will result in a higher-pitched sound. Defaults to 1.0 (no change in frequency).

loop = *loop_flag* (int)

Set *loop_flag* to a nonzero value to have the sound sample loop over and over. Defaults to 0.

pan = *p* (int)

This is the positional offset of the sound, measured in pixels. If $p > 0$, then the sound is offset to the front of the player. If $p < 0$, then sound is offset to the back. Defaults to 0. This parameter is mutually exclusive with abspan.

abspan = *p* (int)

Like pan, except the sound is panned from the center of the screen, not from the player's position. This parameter is mutually exclusive with pan.

Example:

```
; Plays back sound 2,0 from the player's SND file
type = PlaySnd
value = 2,0

; Plays back sound 5,2 from fight.snd
type = PlaySnd
value = F5,2
```

Notes:

Prior to version 1.0 RC8, a volume parameter was used instead of volumescale. The volume parameter is no longer supported and is now ignored.

PosAdd

Offsets the player's position by the specified amounts. The X coordinate is relative to the player's axis, with positive values moving in the direction that the player is facing. The Y coordinate is relative to the player's axis, with negative values moving up.

Required parameters:

none

Optional parameters:

x = *x_value* (float)

Moves the player *x_value* pixels forward. Defaults to 0.

y = *y_value* (float)

Moves the player *y_value* pixels downwards. Defaults to 0.

Example:

none

PosFreeze

Temporarily freezes the player's position.

Required parameters:

none

Optional parameters:

value = *freeze_flag* (boolean)

If *freeze_flag* is non-zero, the player's position will be frozen, else it will not be. Defaults to 1.

Example:

none

PosSet

Sets the player's position to the specified coordinates. The X coordinate is relative to the center of the screen, with positive values moving right. The Y coordinate is relative to the floor, with negative values moving up.

Required parameters:

none

Optional parameters:

x = *x_value* (float)

Specifies the new x-position of the player.

y = *y_value* (float)

Specifies the new y-position of the player.

Example:

none

PowerAdd

Adds the specified amount to the player's power.

Required parameters:

value = *add_amt* (int)

add_amt is the number to add to the player's power.

Optional parameters:

none

Example:

none

PowerSet

Sets the amount of power that the player has.

Required parameters:

value = *pow_amt* (int)

pow_amt is the new value to set the player's power to.

Optional parameters:
none

Example:
none

Projectile

Creates a projectile for the player. The Projectile controller takes all the parameters of the HitDef controller, which control the HitDef for the projectile. In addition, Projectile has the following additional parameters:

Required parameters:
none

Optional parameters:

ProjID = *id_no* (int)
Specifies an ID number to refer to this projectile by. Should be positive, if specified.

projanim = *anim_no* (int)
Specifies the animation action number to use for the projectile's animation. Defaults to 0 if omitted.

projhitanim = *anim_no* (int)
Specifies the animation action number to play when the projectile hits the opponent. Defaults to -1 (no change in animation) if omitted.

projremanim = *anim_no* (int)
Specifies the animation action number to play when the projectile is removed (due to its time expiring or hitting the its removal boundaries, etc.) If omitted, projhitanim is used instead.

projcancelanim = *anim_no* (int)
Specifies the animation action number to play when the projectile is cancelled by hitting another projectile. If omitted, projremanim is used instead.

projscale = *x_scale*, *y_scale* (float, float)
Specifies the scale factor of the projectile. The final scale of the projectile is affected by both this parameter and the "proj.doscale" parameter in the [Size] group of p1's constants file. Defaults to 1,1 (normal size) if omitted.

projremove = *remove_flag* (boolean)
Set to a non-zero value to have the projectile be removed after it hits, or to 0 to disable this behavior. Defaults to 1.

projremovetime = *remove_time* (int)
Specifies the number of ticks after which the projectile should be removed from the screen. If -1, the projectile will not be removed. Defaults to -1.

velocity = *x_vel*, *y_vel* (float, float)
Specifies the initial x and y velocities for the projectile to travel at. Defaults to 0,0 if omitted.

remvelocity = *x_vel*, *y_vel* (float, float)
Specifies the x and y velocities at which the projectile should travel while being removed. Defaults to 0,0 if omitted.

accel = *x_accel*, *y_accel* (float, float)
Specifies the acceleration to apply to the projectile in the x and y directions. Defaults to 0,0 if omitted.

velmul = *x_mul*, *y_mul* (float, float)
Specifies x and y velocity multipliers. The projectile's velocity is multiplied by these multipliers on every tick. The multipliers default to 1 if omitted.

projhits = *num_hits* (int)
Specifies the number of hits that the projectile can impart on an opponent before it is removed. Defaults to 1.

projmisstime = *miss_time* (int)

If the projectile is configured for multiple hits, *miss_time* specifies the number of ticks after each hit before the projectile can hit again. Defaults to 0.

projpriority = *proj_priority* (int)
Specifies the projectile priority. If the projectile collides with another projectile of equal priority, they will cancel. If it collides with another of lower priority, it will cancel the lower- priority projectile, and the higher-priority one will have its priority decreased by 1. Defaults to 1.

projsprpriority = *priority* (int)
Specifies the sprite priority of the projectile. Higher-priority sprites are drawn on top of lower-priority sprites. Defaults to 3.

projedgebound = *value* (int)
Specifies the distance off the edge of the screen before the projectile is automatically removed. Units are in pixels. Defaults to 40 in 240p, 80 in 480p, 160 in 720p.

projstagebound = *value* (int)
Specifies the greatest distance the projectile can travel off the edge of the stage before being it is automatically removed. Defaults to 40 in 240p, 80 in 480p, 160 in 720p.

projheightbound = *lowbound*, *highbound* (int, int)
Specifies the least and greatest y values the projectile is allowed to reach. If the projectile leaves these boundaries, it is automatically removed. Note: since y values decrease with increasing height on the screen, lowbound actually specifies the greatest height the projectile can attain. *lowbound* defaults to -240 in 240p, -480 in 480p, -960 in 720p. *highbound* defaults to 1 in 240p, 2 in 480p, 4 in 720p.

offset = *off_x*, *off_y* (int, int)
Specifies the x and y offsets at which the projectile should be created. Both parameters default to 0 if omitted. Projectiles are always created facing the same direction as the player. *off_x* is in relation to the direction the projectile is facing. The exact behavior of the offset parameters is dependent on the postype.

postype = *postype_string* (string)
postype_string specifies the postype -- how to interpret the pos parameters. In all cases, a positive y offset means a downward displacement. In all cases, *off_y* is relative to the position of the player.

Valid values for *postype_string* are the following:

p1
Interprets offset relative to p1's axis. A positive *off_x* is toward the front of p1. This is the default value for postype.

p2
Interprets offset relative to p2's axis. A positive *off_x* is toward the front of p1. If p2 does not exist, the position is calculated with respect to p1 and a warning is logged.

front
Interprets *off_x* relative to the edge of the screen that p1 is facing toward. A positive *off_x* is toward the front of p1.

back
Interprets *off_x* relative to the edge of the screen that p1 is facing away from. A positive *off_x* is toward the front of p1.

left
Interprets *off_x* relative to the left edge of the screen. A positive *off_x* is toward the front of p1.

right
Interprets *off_x* relative to the right edge of the screen. A positive *off_x* is toward the front of p1.

projshadow = *shad_r*, *shad_g*, *shad_b* (int, int, int)
Specifies the R, G, and B components of the projectile's shadow. These components should be integers between 0 and 255, inclusive. If *shad_r* evaluates to -1, then the stage's shadow color will be used. The higher a component value, the less of that color is displayed in the shadow. So a perfectly black shadow is 255,255,255. Defaults to 0,0,0 (no shadow).

supermovetime = *move_time* (int)

Specifies the number of ticks that the projectile will be unfrozen during a SuperPause. Defaults to 0.

pausemovetime = *move_time* (int)
Specifies the number of ticks that the projectile will be unfrozen during a Pause. Defaults to 0.

afterimage.time = *aftimg_time* (int)
See below.

afterimage.length
See below.

afterimage....
If included, these parameters add afterimage effects to the projectile. The parameters are the same as in the AfterImage controller, except these are all prepended with "afterimage."

Notes:
All projectiles created by helpers immediately become owned by the root.

The behavior of a projectile's HitDef is undefined when executed from a [Statedef -2] block while the player has another player's state and animation data.

Example:
none

RemapPal

Changes one of the player's palettes to another.

Required parameters:
source = *src_pal_grp*, *src_pal_item*
See below.

dest = *dst_pal_grp*, *dst_pal_item*
All of the player sprites that use the source palette will be drawn using the dest palette instead. The source and dest palettes must exist within the player's sprites, and both must be of the same color depth. Note that the dest palette number refers to an unmapped palette; e.g. if you map (1,1) -> (1,6), and then you map (2,2) -> (1,1), the result is that (2,2) will be mapped to palette (1,1) from the original sprite file, not (1,6).

Optional parameters:
none

Example:

```
; All sprites using palette (1,1) will be drawn using palette (1,3)
; instead.
type = RemapPal
source = 1,1
dest = 1,3
```

RemoveExplod

Removes all of a player's explods, or just the explods with a specified ID number.

Required parameters:
none

Optional parameters:
ID = *remove_id* (int)
remove_id is the ID number of the explods to remove. If omitted, removes all explods owned by the player.

ReversalDef

Defines a reversal. If one of P2's Clns1 boxes comes in contact with one of P1's Clsn1 boxes and a ReversalDef is active, then P1 will reverse P2's attack. Use with p1stateno (and optionally p2stateno) for creating reversal attacks.

ReversalDefs take the HitDef parameters pausetime, sparkno, hitsound, p1stateno, and p2stateno, plus:

Required parameters:

reversal.attr = *attr_string*

attr_string specifies the list of attack attributes that can be reversed by this ReversalDef. It is a standard hit attribute string. For instance, `reversal.attr = SA,NA,SA` means stand+air, normal attack, special attack.

Optional parameters:

none

Notes:

The sparkxy parameter is treated as an offset to P2's hitdef's sparkxy. The MoveHit trigger can be used to detect if P1 successfully reversed P2.

Example:

none

ScreenBound

Specifies whether or not the player's movement should be constrained to the screen or not. Also determines whether the camera should move to follow the player or not. The results of this controller are valid for 1 tick.

Required parameters:

none

Optional parameters:

value = *bound_flag* (boolean)

If *bound_flag* is 0, the player will be allowed to move off the screen. If 1, the player is constraing to the screen. Defaults to 0 if omitted.

movecamera = *move_x_flag*, *move_y_flag* (boolean, boolean)

If 1, specifies that camera should pan to follow the player in the x direction and in the y direction, respectively. Defaults to 0 in both instances if omitted.

Examples:

none

SelfState

Like ChangeState, except that this changes a player back to a state in his own state data. Use this when you have placed an opponent player in a custom state via an attack, and wish to restore the opponent to his own states.

SprPriority

Changes the player's sprite priority. Higher-priority sprites are drawn on top of lower-priority sprites.

Required arguments:

value = *priority_level* (int)

Valid values are -5 to 5.

Optional arguments:

none

Example:

none

StateTypeSet

Changes the current state type and move type. Useful for states that go from the ground into the air, etc.

Required parameters:
none

Optional parameters:
statetype = *state_type* (string)
Set *state_type* to A for air, C for crouch, S for stand, or L for liedown. Defaults to no change.

movetype = *move_type* (string)
Set *move_type* to I for idle, A for attack, or H for gethit. Defaults to no change.

physics = *physics* (string)
Set *physics* to A for air, C for crouch, S for stand, or N for none. Defaults to no change.

Example:
none

SndPan

Changes the panning of a currently playing sound. This controller may be continually triggered to smoothly move a sound across the sound field or to have a sound follow the player.

Required parameters:
channel = *chan_no* (int)
Specifies the channel number of the sound to pan.

pan = *p* OR abspan = *p* (int)
These parameters cannot both be specified at the same time. p determines the sound offset in pixels from the player (in the case of pan) or from the center of the screen (in the case of abspan). See PlaySnd for a description of the panning parameters.

Optional parameters:
none

Example:
none

StopSnd

Stops any sound which is playing on the specified channel.

Required parameters:
channel = *chan_no* (int)
Stops playback of any sound on *chan_no*. If *chan_no* is -1, then all sounds are stopped, including those belonging to other players.

Optional parameters:
none

Example:
none

SuperPause

Freezes the gameplay and darkens the screen. While each player is frozen, no time passes for them. Use for a dramatic pause during the start of hyper attacks.

Required parameters:
none

Optional parameters:

SuperPause accepts all optional parameters that the Pause controller does. In addition, SuperPause also takes the following parameters:

time = *pause_time* (int)
: Specifies the number of ticks that the pause should last. Default is 30 ticks (half a second at default speed).

anim = *anim_no* (int)
: Specifies the animation number (from fightfx.air) to play during the SuperPause. The default is 30, which is a charging effect. If anim is -1, no animation will be played. If you prepend "S" to *anim_no*, the animation used will be from the player's AIR file. For example, `anim = S10`.

sound = *snd_grp*, *snd_no* (int, int)
: Specifies a sound to play (from common.snd) during SuperPause. The default is -1, which means no sound is played. If you prepend "S" to *snd_grp*, then the sound used will be from the player's own SND file. For example, `sound = S10,0`.

pos = *x_pos*, *y_pos* (float)
: Specifies the offset (from the player axis) at which the super anim is to be displayed. Defaults to 0,0.

darken = *bvalue* (boolean)
: If this is 1, the screen will darken during the SuperPause. Set this to 0 to disable this effect. The default value is 1.

p2defmul = *def_mul* (float)
: This is the amount in which to temporarily multiply the defence of any targets the player has. This is used to make chaining into supers less damaging. Setting this at 1 will make no changes to the targets' defence. 0 is a special value that will set the defence to the number set in Super.TargetDefenceMul in the [Rules] section of mugen.cfg. The default value is 0. Valid values are all positive numbers, including zero.

poweradd = *value* (int)
: This is the amount to add to the player's power. Defaults to 0.

unhittable = *bvalue* (boolean)
: If set to 1, the player cannot be hit during the SuperPause. Set to 0 to disable this. Defaults to 1.

Notes:

If the Pause controller was previously executed, and the action is still paused, executing a SuperPause will preempt the Pause controller's effects. During the SuperPause, the time left until the Pause controller's effects expires will not count down.

Example:

none

TargetBind

Binds the player's specified targets to a specified location relative to the player's axis.

Required parameters:

none

Optional parameters:

time = *bind_time* (int)
: Specifies number of ticks that this binding should be in effect. Defaults to 1.

ID = *bind_id* (int)
: Specifies the desired target ID to bind. Only targets with this target ID will be bound. Defaults to -1 (bind all targets.)

pos = *x_pos*, *y_pos* (float)
: Specifies the offset from the player's axis to bind the target to. Defaults to 0,0 if omitted.

TargetDrop

Drops all targets from the player's target list, except possibly for those having a specified target ID number. Useful for applying effects to only certain targets.

Required parameters:
none

Optional parameters:
excludeID = *id_no* (int)
Any targets with target ID number not equal to id_no will be dropped from the player's target list. Defaults to -1 (drop all targets).

keepone = *keep_flag* (boolean)
If *keep_flag* is non-zero, then at most one target is kept on the player's target list. If there are multiple targets whose target ID number is the same as id_no, one will be picked at random and the rest will be dropped. This behavior is useful in throws, to keep from throwing multiple opponents simultaneously. If *keep_flag* is 0, then all targets with the appropriate ID number will be kept. *keep_flag* defaults to 1.

Example:
none

TargetFacing

Turns all targets to face a specified direction relative to the player.

Required parameters:
value = *facing_val* (int)
If *facing_val* is positive, all targets will turn to face the same direction as the player. If *facing_val* is negative, all targets will turn to face the opposite direction as the player.

Optional parameters:
ID = *target_id* (int)
Specifies the desired target ID to affect. Only targets with this target ID will be affected. Defaults to -1 (affects all targets.)

Example:
none

TargetLifeAdd

Adds the specified amount to all targets' life, scaled by the targets' defense multipliers if necessary.

Required parameters:
value = *add_amt* (int)
add_amt is added toe ach target's life.

Optional parameters:
ID = *target_id* (int)
Specifies the desired target ID to affect. Only targets with this target ID will be affected. Defaults to -1 (affects all targets.)

kill = *kill_flag* (boolean)
If kill_flag is 0, then the addition will not take any player below 1 life point. Defaults to 1.

absolute = *abs_flag* (boolean)
If *abs_flag* is 1, then *add_amt* will not be scaled (i.e. attack and defense multipliers will be ignored). Defaults to 0.

Example:
none

TargetPowerAdd

Adds the specified amount to all targets' power.

Required parameters:
value = *add_amt* (int)
add_amt is added to each target's power.

Optional parameters:
ID = *target_id* (int)
Specifies the desired target ID to affect. Only targets with this target ID will be affected. Defaults to -1 (affects all targets.)

Example:
none

TargetState

Makes all targets change to the specified state number.

Required parameters:
value = *state_no* (int)
Specifies the number of the state to change the targets to.

Optional parameters:
ID = *target_id* (int)
Specifies the desired target ID to affect. Only targets with this target ID will be affected. Defaults to -1 (affects all targets.)

Examples:
none

TargetVelAdd

Adds the specified amounts to all targets' velocities. A positive x velocity is in the direction that the target is facing, while a positive y velocity is downward on the screen.

Required parameters:
none

Optional parameters:
x = *x_value* (float)
Specifies the value to add to the x-velocity of the target.

y = *y_value* (float)
Specifies the value to add to the y-velocity of the target.

ID = *target_id* (int)
Specifies the desired target ID to affect. Only targets with this target ID will be affected. Defaults to -1 (affects all targets.)

Example:

```
; Applies constant gravity to all targets
type = TargetVelAdd
trigger1 = 1
y = 0.45
```

TargetVelSet

Sets all targets' velocities to the specified values. A positive x velocity is in the direction that the player is facing, while a positive y velocity is downward on the screen.

Required parameters:

none

Optional parameters:

x = *x_value* (float)

Specifies the value to set the x-velocity of the target to

y = *y_value* (float)

Specifies the value to set the y-velocity of the target to.

ID = *target_id* (int)

Specifies the desired target ID to affect. Only targets with this target ID will be affected. Defaults to -1 (affects all targets.)

Example:

none

Trans

Overrides the player's animation transparency parameters for current game tick. Useful for special effects.

Required parameters:

trans = *trans_type* (string)

trans_type must be one of the following:

- default - does nothing
- none - disables transparency
- add - draws with full additive transparency
- addalpha - draws with additive transparency (alpha must be specified)
- add1 - draws with additive transparency, with alpha at 256,128
- sub - draws with full subtractive transparency

Optional parameters:

alpha = *source_alpha*, *dest_alpha* (int, int)

These are the source and destination alpha values for the addalpha trans type. Valid values are from 0 (low) to 256 (high). If omitted, defaults to 256,0.

Example:

```
; Fades the character in, over 256 ticks.
type = Trans
trigger1 = time < 256
trans = add_alpha
alpha = time, 256-time
```

Turn

Instantly turns the player to face the opposite direction. Does not play a turning animation.

Required parameters:

none

Optional parameters:

none

Example:

none

VarAdd

Adds to one of the player's working variables. Either a float variable or an int variable can be added to by this controller.

Required parameters (int version):

v = *var_no* (int)

var_no specifies the number of the variable to affect. It must evaluate to an integer between 0 and 59.

value = *int_expr* (int)

int_expr specifies the value to add to the int variable indexed by *var_no*.

Required parameters (float version):

fv = *var_no* (int)

var_no specifies the number of the variable to affect. It must evaluate to an integer between 0 and 39.

value = *float_expr* (float)

float_expr is the value to add to the float variable indexed by *var_no*.

Optional parameters:

none in both cases

Alternate syntax:

var(*var_no*) = *int_expr* (int version)

fvar(*var_no*) = *float_expr* (float version)

Notes:

Due to historical reasons, note that the alternate VarAdd syntax listed above matches neither the syntax for variable assignment within an expression, nor the syntax for variable addition within an expression.

If you have placed P2 in a custom state through a successful hit, do not use variable assignment within the custom states. Otherwise, you will overwrite P2's variables, which can cause unintended malfunction of the opponent player.

Example:

none

VarRandom

Sets the specified int variable to a random value. Float variables cannot be set by this controller.

Required parameters:

v = *var_no* (int)

var_no is the index of the int variable to affect. It must evaluate to an integer between 0 and 59.

Optional parameters:

range = *least_val*, *greatest_val* (int)

least_val and *greatest_val* specify the least and greatest values which can be assigned by this controller, respectively. The value assigned to the variable will be a randomly chosen integer from this range. range defaults to 0,1000. If only one argument is specified, that is considered to specify the range 0,(argument).

Notes:

If you have placed P2 in a custom state through a successful hit, do not use variable assignment within the custom states. Otherwise, you will overwrite P2's variables, which can cause unintended malfunction of the opponent player.

Example:

```
;Assign a random number between 0 and 500 to var(5).
type = VarRandom
v = 5
range = 500
```

VarRangeSet

Sets a contiguous range of the player's working variables to the same value. Either float variables or int variables can be set by this controller, but not both at the same time.

Required parameters (int version):

value = *int_expr* (int)

int_expr is evaluated once to give the value that is assigned to all int variables in the range.

Required parameters (float version):

fvalue = *float_expr* (float)

float_expr is evaluated once to give the value that is assigned to all float variables in the range.

Optional parameters (both versions):

first = *first_idx* (int)

Specifies the lower end of the range of variables to set. Defaults to 0 (first variable).

last = *last_idx* (int)

Specifies the higher end of the range of variables to set. Defaults to 59 for int variables, or 39 for float variables (this is the last available variable in both cases).

Notes:

If you have placed P2 in a custom state through a successful hit, do not use variable assignment within the custom states. Otherwise, you will overwrite P2's variables, which can cause unintended malfunction of the opponent player.

Example:

none

VarSet

Sets one of the player's working variables. Either a float variable or an int variable can be set by this controller, but not both at the same time.

Required parameters (int version):

v = *var_no* (int)

var_no specifies the number of the variable to affect. It must evaluate to an integer between 0 and 59.

value = *int_expr* (int)

int_expr specifies the value to assign to the int variable indexed by *var_no*.

Required parameters (float version):

fv = *var_no* (int)

var_no specifies the number of the variable to affect. It must evaluate to an integer between 0 and 39.

value = *float_expr* (float)

float_expr is the value to assign to the float variable indexed by *var_no*.

Optional parameters:

none in both cases

Alternate syntax:

var(*var_no*) = *int_expr* (int version)

fvar(*var_no*) = *float_expr* (float version)

Notes:

Due to historical reasons, note that the alternate variable assignment syntax listed above does not exactly match the syntax for variable assignment within an expression.

If you have placed P2 in a custom state through a successful hit, do not use variable assignment within the custom states. Otherwise, you will overwrite P2's variables, which can cause unintended malfunction of the opponent player.

Example:

none

VelAdd

Adds the specified amounts to the player's velocity. A positive x velocity is in the direction that the player is facing, while a positive y velocity is downward on the screen.

Required parameters:
none

Optional parameters:
x = *x_value* (float)
Specifies the value to add to the player's x-velocity.

y = *y_value* (float)
Specifies the value to add to the player's y-velocity.

Example:

```
; Applies constant gravity to the player
trigger1 = 1
type = VelAdd
y = 0.45
```

VelMul

Multiplies the player's velocity by the specified amounts. A positive x velocity is in the direction that the player is facing, while a positive y velocity is downward on the screen.

Required parameters:
none

Optional parameters:
x = *x_value* (float)
Specifies the value to multiply the player's x-velocity with.

y = *y_value* (float)
Specifies the value to multiply the player's y-velocity with.

Example:

```
; Applies constant friction to the player
trigger1 = 1
type = VelMul
x = 0.8
```

VelSet

Sets the player's velocity to the specified values. A positive x velocity is in the direction that the player is facing, while a positive y velocity is downward on the screen.

Required parameters:
none

Optional parameters:
x = *x_value* (float)
Specifies the value to assign to the player's x-velocity.

y = *y_value* (float)
Specifies the value to assign to the player's y-velocity.

Example:
none

VictoryQuote

Selects a victory quote from the player to display in the next victory screen.

Required parameters:
none

Optional parameters:
value = *quote_index* (int)
Specifies the index of the quote to use. Valid index values are from 0 to 99. If *quote_index* evaluates to an invalid index, a random quote will be selected. Defaults to -1.

Notes:
This controller can be called by any player at any time during a match; however only the winning player will affect the quote that is shown. This controller only affects the victory screen immediately following the current match. This controller has no effect if executed by a helper. The actual victory quotes are specified in the [Quotes] group of the player's constants file.

Example:
none

Width

Temporarily changes the size of the player's width bar for 1 tick. Useful for controlling the "pushing" behavior when the player makes contact with another or with the sides of the screen.

Required parameters:
none

Optional parameters:
edge = *edgewidth_front*, *edgewidth_back* (int, int)
Sets the player's edge width in front and behind. Edge width determines how close the player can get to the edge of the screen. These parameters default to 0,0 if omitted.

player = *playwidth_front*, *playwidth_back* (int, int)
Sets the player width in front and behind. Player width determines how close the player can get to other players. These parameters default to 0,0 if omitted.

Alternate syntax:
value = *width_front*, *width_back* (int, int)
This is a shorthand syntax for setting both edge width and player width simultaneously. This may only be used if the edge and player parameters are not specified.

Notes:
When collision box display is enabled, the edge width bar is displayed in orange, and the player width bar is displayed in yellow. Where they overlap, the overlapping region is displayed in bright yellow.

Example:
none