Background / Stage Documentation

Documentation for version 1.1 (2010)

Updated 28 July 2010

Contents

Overview

A background in M.U.G.E.N consists of one or more background elements, and zero or more background controllers. These are combined with stage settings to produce the appearance and functionality of a stage for use in M.U.G.E.N.

The stage settings control general parameters such as the size of the stage, movement of the camera, starting position of characters and camera in relation to the stage, shadow intensity, floor reflections, etc. Most fighting games maintain consistent values of these background parameters across their constituent stages, especially in regards to stage size and camera movement.

A background element is an individual unit of graphical data that is displayed on screen. All the visible parts of the stage are formed out of background elements. Each background element has individual settings for its sprite data, transparency, location, tiling, scroll speed with respect to the camera, and animation (if applicable). "Parallaxing" elements add a raster effect when scrolling the camera. Background elements may be of any size, although in practice, backgrounds are usually pieced together from several moderately-sized elements.

A background controller will perform additional manipulations on a background element, such as change its position or velocity, make it invisible or invisible, etc. Applications of background controllers include making characters stroll back and forth in the background, having debris fly across the screen, or creating changes in the background at set times (such entering a cave when the characters are fighting on a ship). In general, background controllers may allow you to create more advanced effects for a stage or optimize certain animations to reduce memory consumption, although many stages will do quite well without any controllers at all.

When creating a stage, you must assure that the visible area on the screen is "covered" by background elements at all times. Leaving holes in the background will cause a hall-of-mirrors effect in that location. Enabling the `debugbg` parameter will fill all holes with a bright shade of magenta, which should assist you in locating and fixing them.

Besides the obvious use in stages, backgrounds objects are used in M.U.G.E.N for the backgrounds of all system screens such as the title screen and the character select screen, as well as for use in storyboards. These types of backgrounds are referred to as "system backgrounds" and "storyboard backgrounds" respectively. Background objects used for stages are called "stage backgrounds". Besides their intended use, there is essentially no difference between the different types of backgrounds. This document covers stage settings as well as the background object format.

Description of stage settings

Stage settings define the background's relationship with the stage. The stage settings must be in the same stage file as the background. There are several groups that make up the settings. They are:

(all parameters are required unless specifically marked as "optional")

`Info` Group

`name` = *stage_name*
: This parameter sets the name of the stage. The parameter must be enclosed in double-quotes.

`displayname` = *display_name*
: This parameter sets the name to display in the stage select list. The parameter must be enclosed in double-quotes. If omitted, defaults to *stage_name*.

`versiondate` = *version*
: This parameter sets the version of the stage. The formats accepted are either a date (MM,DD,YYYY) or a x.xx version number, e.g. 1.23. This parameter is for informational purposes and is currently not used by the engine.

`mugenversion` = *mugen_version*

: This parameter sets the target version number of M.U.G.E.N that the stage is designed for. Valid values are `2002,02,14`, `1.0` or `1.1`. Defaults to `2002,02,14`.

: A version number of `2002,02,14` restricts camera movement to whole units of the stage coordinate space, whereas `1.0` and higher allows the camera position to assume fractional values.

`author` = *author_name*
: This parameter sets the name of the author of the stage. The parameter must be enclosed in double-quotes. This parameter is for informational purposes and is currently not used by the engine.

`Camera` Group

`startx` = *pos_x*
: Sets the camera's starting `x`-position. Should be set to 0.

`starty` = *pos_y*
: Sets the camera's starting `y`-position. Should be set to 0.

`boundleft` = *min_x*
: This is the minimum `x`-value that the camera can move to. It must be negative. Adjust this value to limit how far left the camera can scroll.

`boundright` = *max_x*
: This is the maximum `x`-value that the camera can move to. It must be positive. Adjust this value to limit how far right the camera can scroll.

`boundhigh` = *min_y*
: Controls the minimum `y`-value that the camera can move to. It must be negative. Adjust this value to limit how far up the camera can scroll.

`boundlow` = *max_y*
Controls the maximum `y`-value that the camera can move to. It must always be set to 0.
`tension` = *h_dist*
This is the horizontal distance a player can get from the left and right edges before the camera starts to follow.
`tensionhigh` = *top_dist* ; (Version 1.1 and higher)
This is the vertical distance a player can get from the top edge before the camera starts to follow. The use of the `tensionhigh` and `tensionlow` parameters activates a method of vertical scrolling based on edge tension introduced in Version 1.1. If `tensionhigh` and `tensionlow` are specified, `verticalfollow` and `floortension` will be ignored.
`tensionlow` = *bottom_dist* ; (Version 1.1 and higher)
This is the vertical distance a player can get from the bottom edge before the camera starts to follow.
`verticalfollow` = *closeness*
This value affects the vertical movement of the camera to follow the highest player. It should be set between 0 and 1. A value of 0 means the camera does not move up at all. A higher value makes the camera track the higher player closer. A value of 1 makes the camera track the player as close as possible. This parameter is ignored if `tensionhigh` and `tensionlow` are used.
`floortension` = *v_dist*
This is the minimum vertical distance the highest player has to be from the floor (given as a positive value), before the camera starts to move up to follow him. This parameter is used only with the `verticalfollow` camera scrolling mode. This parameter is ignored if `tensionhigh` and `tensionlow` are used.
`overdrawhigh` = *top_pixels* ; (Version 1.0 and higher)
See below.
`overdrawlow` = *bottom_pixels* ; (Version 1.0 and higher)
Number of pixels beyond the top and bottom of the screen that may be drawn. Overdraw specifies the how much can be seen during an EnvShake. Overdraw pixels will also be used when the screen aspect is taller than the stage aspect.
`cuthigh` = *top_pixels* ; (Version 1.0 and higher)
See below.
`cutlow` = *bottom_pixels* ; (Version 1.0 and higher)
Number of pixels into the top and bottom of the screen that may be cut from drawing when the screen aspect is shorter than the stage aspect. These parameters suggest a guideline, and the actual number of pixels cut depends on the difference in aspect. If `cuthigh` and `cutlow` are omitted, the engine will attempt to guess a reasonable set of values.
`startzoom` = *start_zoom_scale* ; (Version 1.1 and higher)
This specifies the initial scale factor for the stage. The effect of this parameter is most visible if there are no characters that affect the camera. Defaults to 1.
`zoomout` = *zoom_out_scale* ; (Version 1.1 and higher)
This specifies the scale factor for the stage when the camera is fully zoomed out. Note that a smaller number allows the camera to zoom out more, e.g. `zoomout = .5` specifies that the camera can zoom outwards by a factor of 2. Defaults to 1.
`zoomin` = *zoom_in_scale* ; (Version 1.1 and higher)
This specifies the scale factor for the stage when the camera is fully zoomed in. Defaults to 1.

`PlayerInfo` Group

`p1startx` = *x_start*
Sets the starting `x`-position for Player 1.
`p1starty` = *y_start*
Sets the starting `y`-position for Player 1.
`p2startx` = *x_start*
Sets the starting `x`-position for Player 2.
`p2starty` = *y_start*
Sets the starting `y`-position for Player 2.
`p1facing` = *facing_flag*
Controls which way Player 1 faces at the start. Set to 1 to face right, or -1 to face left.
`p2facing` = *facing_flag*
Controls which way Player 2 faces at the start. Set to 1 to face right, or -1 to face left.
`leftbound` = *x_min*

Sets the minimum allowable x-position for all players.

`rightbound` = *x_max*

Sets the maximum allowable x-position for all players.

Bound Group

`screenleft` = *min_dist_left*

Minimum allowable distance between players and the left edge of the screen. Must be positive.

`screenright` = *min_dist_right*

Minimum allowable distance between players and the right edge of the screen. Must be positive.

StageInfo Group

`zoffset` = *v_dist*

v_dist is the vertical distance of the ground level from the top of the screen, given in pixels. The `zoffset` parameter is inappropriately named, but has been left unchanged for compatibility purposes.

`zoffsetlink` = *elem_ID*

If this parameter is specified, it links the `zoffset` value to the `y`-position of a background element with the specified ID number. For example, you can link the value to a dummy element (see Other background element types) with a `sin.y` parameter to make the characters bob up and down in a sinusoidal fashion.

`autoturn` = *turn_flag*

Leave this parameter to 1 to make characters automatically turn to face each other. Setting this to another value will cause undesirable behavior.

`resetBG` = *reset_flag*

If set to 1, backgrounds will be reset between rounds. If set to 0, backgrounds will continue playing.

`localcoord` = *width*, *height*

Dimensions of the coordinate space of the stage. Defaults to 320, 240 if omitted.

`xscale` = *xscale*

Horizontal scaling factor of offsets, velocities, sprites and animations.

`yscale` = *yscale*

Vertical scaling factor of offsets, velocities, sprites and animations.

Shadow Group

`intensity` = *darkness_val* ;(optional) (int)

This controls how dark the shadow is. Valid values range from 0 (lightest) to 256 (darkest). Defaults to 128 if omitted.

`color` = *r,g,b* ;(optional) (deprecated)

This parameter is no longer supported as of M.U.G.E.N 1.1 and will be ignored.

`yscale` = *scale_y* ;(optional) (float)

This is the vertical scale factor of the shadow. Use a larger *scale_y* to make the shadow longer. You can use a negative *scale_y* to make the shadow fall on the other side of the players. Defaults to `0.4` if omitted.

`fade.range` = *top_y, bot_y* ;(optional) (int, int)

This parameter lets you set the range over which the shadow is visible. It is used to create an effect of the shadow fading as the player gets farther away from the ground. The first value is the high level, and the second is the middle level. Both represent y-coordinates of the player. A shadow is invisible if the player is above the high level, and fully visible if below the middle level. The shadow is faded in between the two levels. If omitted, defaults to no effect (shadow is always fully visible). Take note that y-coordinate values are negative, and TOP_Y should be less than BOT_Y.

`xshear` = *shear* ;(optional) (float)

Specifies the amount of horizontal shearing to apply to the shadow. A positive value shears the shadow to the left of the screen. Defaults to `0`.

Reflection Group

`reflect` = *reflect_flag* ;(optional)
Set `reflect` to 1 to enable reflection shadows, 0 to disable them. Reflection shadows create a "shiny floor" effect. Defaults to 0 if omitted.

`Music` Group

`bgmusic` = *bgm_filename* ;(optional)
bgm_filename is the name of the music file to play in the stage. Music files are usually put in the `sound/` directory. Leaving *bgm_filename* blank will result in no music being played. If the music file does not exist, no music will be played. To play CD audio, put the track number followed by ".da". Using a track number of 0 will play a random audio track. For example, to play track 3 from a music CD, use `bgmusic = 3.da`. If omitted, defaults to no music.
`bgvolume` = *volume_offset* ;(optional)
This parameter adjusts the volume of the BGM being played. 0 is normal, negative for softer, and positive for louder (only for mp3, mods and CDA). Valid values are from -255 to 255. If omitted, defaults to 0.

Description of background elements

A background is made up of one [BGDef] group and one or more [BG] groups. The [BG] groups are called background elements.

Exactly one `BGDef` group is required in the stage's `DEF` file. An example of the format:

```
[BGDef]
spr = my_stage.sff
debugbg = 0
```

Replace `my_stage.sff` with the name of the `SFF` file containing your stage's sprite data. The `SFF` file should be placed in the same directory as the stage `DEF` (i.e. `stages/`). If `debugbg` is set to 1, the screen is cleared to magenta before the stage is drawn. This makes "holes" in the background more apparent. To improve runtime speed, set `debugbg = 0` before releasing a stage for distribution.

Once the `BGDef` definition is created, everything below it in the `DEF` file is considered to belong in the `BGDef` section. In the `BGDef` section, you should specify one or more background elements. Background elements are drawn in the order they appear in the `DEF` file (with later elements drawn over earlier elements), so the rearmost elements should be defined first.

Format of background elements

An example for specifying a static background element is as follows.

```
[BG my_element_name]
type = normal
spriteno = 0,0
start = 0,0
delta = .5, .5
mask = 0
```

This example displays sprite 0,0 from the background's SFF file at an initial position of 0,0. For each unit of camera movement, the sprite will move .5 stage units.

Every background element is defined by a group starting with a header `[BG my_element_name]`. `my_element_name` should be replaced with a distinct and descriptive name, as the group header is what will be displayed in error messages.

Parameters are to be specified in the lines following the group header.

Static background element parameters

`type` = *normal* ; (required) (string)
: This specifies that this background element is a static sprite.

`spriteno` = *groupno, imageno* ; (required) (int, int)
: *groupno, imageno* specifies the sprite in the SFF to display for this background element.

`layerno` = *layer* ; (optional) (int)
: If *layer* is 0, the background element will be drawn behind the characters. If *layer* is 1, the element will be drawn in front of the characters. Within each layer, background elements are drawn back-to-front in the order they appear in the `DEF` file. Defaults to `0`.

`start` = *x,y* ; (optional) (float, float)
: Specifies the background element's starting position with respect to the top center of the screen. Positive *x* values go right; positive *y* values go down. The values are given in stage units, which correspond to pixel units in typical use. Defaults to `0,0`.

`delta` = *dx,dy* ; (optional) (float, float)
: Specifies how many pixels the background element should scroll for each pixel of camera movement in the horizontal and vertical directions, respectively. Setting `delta=1,1` will cause the background element to move at the same speed as the camera. *1,1* is an appropriate value for the ground under the characters' feet. For elements off in the distance, use smaller values of `delta` to create the illusion of depth. Similarly, elements in the foreground (`layerno = 1`) should usually be given `delta`s larger than 1. Defaults to `1,1`.

`mask` = *mask* ; (optional) (boolean)
: If *mask* is set to `1`, color 0 of the sprite will not be drawn. This is used in drawing objects which are not rectangular in shape. For performance reasons, `mask` should be set to 0 when not needed. This parameter is ignored for RGB sprites. Defaults to `0` normally. For historical reasons, *mask* will default to `1` if the `trans` parameter is set to `add` or `sub`.

`tile` = *x_tile,y_tile* ; (optional) (int, int)
: Specifies if the background element is to be repeated ("tiled") in the horizontal and/or vertical directions, respectively. A value of 0 specifies no tiling, a value of 1 specifies infinite tiling, and any value greater than 1 will cause the element to tile that number of times. If this line is omitted, no tiling will be performed.

`tilespacing` = *x_spacing,y_spacing* ; (optional) (int, int)
: If tiling is enabled, this line specifies the space to put between separate instances of the tile in the horizontal and vertical directions, respectively. There is no effect if tiling is not enabled. `tilespacing` defaults to `0,0`.

Animated background elements

The format for specifying an animated background element is almost exactly the same as for a normal background element. The differences are described below.

```
[BG my_animated_element]
type = anim
actionno = 55
```

All other parameters other than `spriteno` are the same as for static background elements.

First, for the element to animate, a type of `anim` must be specified. Second, an "action number" (`actionno`) must be specified. This replaces the `spriteno` parameter that would be used for a normal background element. The value of `actionno` must be for an animation that is defined in the `DEF` file. In this example, since `actionno` is 55, Action 55 must be defined in a manner similar to the following:

```
[Begin Action 55]
0,0,0,0,5
0,1,0,0,5
```

The format is the same as specifying animations in the `AIR` file, so details will be omitted here. The `Action` definition can be placed anywhere below the original `[BGDef]` group. Typical strategies are either to define

the action immediately after the element it belongs to, or else to collect all the stage's actions together at the beginning or end of the `[BGDef]` group.

Note that each sprite you specify in the animation has its own axis (specified in the `SFF` file). During animation playback, the axis of each sprite will be lined up to correspond with the axis of the background element itself.

The effect of the `tilespacing` parameter is different for `anim`-type elements compared to `normal`-type elements. For a `normal` element, the x-value of this parameter specifies the horizontal distance between the right edge of the first tile and the left edge of the second tile. In the case of an `anim` element, the x-value specifies the horizontal distance between the *left* edge of the first tile and the left edge of the second tile. This applies similarly for the y-value. The reason for the difference is because the size of an anim is not necessarily constant. `tilespacing` is required for `anim`-type elements that use the `tile` parameter.

Animated elements always have `mask = 1` and cannot be set to non-masked.

Parallaxing background elements

Parallaxing background elements give the illusion of depth for elements that appear to face upwards or downwards. Parallaxing background elements may consist of a single sprite specified by the `spriteno` parameter. In Version 1.1 and higher, parallax elements may be animated by specifying `actionno` like in ordinary animated background elements. Animated parallax may not be used with the `xscale` parameter.

An example of the format is as follows:

```
[BG my_parallax_element]
type = parallax
spriteno = 10, 0
width = 300, 600
scalestart = 1,1
scaledelta = 0,.001
```

Another example:

```
[BG my_parallax_element]
type = parallax
spriteno = 10, 0
xscale = 1,1.5
yscalestart = 100 ;Deprecated!
yscaledelta = 1.2 ;Deprecated!
```

Parameter descriptions follow:

`width` = *top_width, bottom_width* ; (int, int)

top_width and *bottom_width* respectively specify the top and bottom widths of a trapezoid that has the same height as the sprite used. The sprite will be distorted to match the shape and size of the trapezoid. The ratio of *top_width* to *bottom_width* affects the amount of shearing as the camera moves horizontally. Use `width` if the sprite is not preprocessed for perspective distortion. For historical reasons, the x-axis specified in the SFF is ignored. The horizontal center of the sprite is used as the x-axis instead.

`xscale` = *top_xscale, bottom_xscale* ; (float, float) (deprecated)

top_xscale and *bottom_xscale* respectively scale the horizontal delta of the background element's top and bottom edge to create a horizontal shearing effect. For example, `delta = .75, .75` amd `xscale = 1, 2` specifies the top of the sprite would move at .75 * 1 = `.75` stage units per camera unit, and the bottom would move at .75 * 2 = `1.5` stage units per camera unit. This example assumes the sprite axis is at the top of the sprite.

Either `xscale` or `width` is required, but not both at the same time. Use `xscale` if the sprite has already been preprocessed for perspective distortion. Hardware accelerated rendering will have visual artifacts at extreme shear angles.

Animated parallax may not be used when `xscale` is used.

`yscalestart` = *yscalestart* ; (float) (deprecated)

This deprecated parameter controls the change in vertical scale of the element as the camera moves vertically. Its functionality is replaced by the general-purpose `scalestart` and `scaledelta` advanced parameters.

yscalestart is the inverse of the vertical scale of the sprite when the camera is at ground level, represented in percentage. For example, a value of 100 corresponds to a scale factor of 1, and 50 corresponds to a scale factor of 2. The value defaults to 100.

`yscaledelta` = *yscaledelta* ; (float) (deprecated)

Specifies the amount to add from the inverted scale factor for every camera unit the camera moves down, represented in percentage. The final scale factor is calculated using the following formula:

```
scale = 1 / (yscalestart/100 + yscaledelta/100 * camera_y)
```

In the example above, if the camera moves up by one unit, the scale factor will be 1 / (1.00 + .012 * -1) = 1.012, and if it moves up another unit, the scale will be 1 / (1.00 + .012 * -2) = 1.025 and so on.

The unintuitive behavior for the `yscalestart` and `yscaledelta` parameters is the reason for deprecating these parameters in favor of the `scalestart` and `scaledelta` parameters.

Notes:

Other parameters for static elements are also valid for parallax elements, with the exception of `tilespacing`. Additionally, *y_tile* is ignored and forced to be 0.

The result of specifying an angle in an animated parallax element is undefined.

The software renderer does not support animated parallax.

Advanced Parameters

These optional parameters can be added to any background element.

`trans` = *trans* ; (optional) (string)

Specifies the type of transparency blending to perform on the sprite. Transparency modes are `default`, `none`, `add`, `sub`. Respectivelty, these specify default behavior, no transparency, color addition and color subtraction. `default` and `none` both cause non-animated elements to be drawn without any tranparency blending. For animated background elements, `default` will use anim-specific transparency while other values of *trans* will override the transparency flags in the anim. If `add` is specified, the `alpha` parameter may be used to control the blending. Defaults to `default`.

Note: If `mugenversion` is less than 1.0, `none` will behave the same as `default` for animated elements.

`alpha` = *src_alpha,dst_alpha* ; (optional) (int, int)
Specifies the source and destination blend contributions when `trans = add`. Acceptable values are from `0` to `256`. Defaults to `256,256`.

`id` = *id_number* ; (int)
This specifies an ID number for the background element. Multiple elements can share the same ID number. The purpose of the ID number is to allow background controllers to specify which elements they are operating on. Defaults to 0.

`positionlink` = *link_flag* ; (boolean)
Set `positionlink` to 1 to lock this element's position to the position of the element immediately preceding it in the `DEF` file. If set to 1, the values of the `start` parameter for the current element are treated as an offset from the previous element's position. The `delta` parameter will have no effect in this case. This is useful for getting large groups of elements to move in unison; if you edit the `start` and `delta` values of the first element in the positionlink chain, the effects will be seen throughout all elements the chain. Defaults to 0.

`maskwindow` = *x1,y1,x2,y2* ; (int, int, int, int) (Version 1.1 and higher)
: Specifies a masking window for the background element. A masking window is conceptually a rectangular box on the screen in which the sprite is contained. Any part of the sprite that lies outside of the box will not be drawn. *x1,y1* specifies the coordinates of the top-left corner of the window, and *x2,y2* specifies the coordinates of the bottom-right corner of the window. These coordinates are relative to the stage origin (top-middle of screen). These coordinates are "endpoint exclusive", i.e. the pixels along the bottom and right edges of the window will not be drawn. If omitted, window masking will not performed.

`window` = *x1,y1,x2,y2* ; (int, int, int, int) (deprecated)
: This deprecated parameter is included for compatibility and the use of it is not guaranteed to work correctly under all cases of zooming. It specifies a masking window with coordinates relative to the top-right corner of the screen. The coordinates are endpoint inclusive.

`windowdelta` = *window_dx,window_dy* ; (float, float)
: Specifies the movement delta of the masking window. It works similarly to the `delta` parameter for the background element itself. Defaults to `0,0`.

`velocity` = *vel_x, vel_y* ; (float, float)
: Specifies initial `x`- and `y`-velocities for the background element (these default to 0). This functionality is also subsumed by the `VelSet` background controller.

`scalestart` = *scale_x, scale_y* ; (float, float)
: Starting scale. Defaults to `1,1`.

`scaledelta` = *scale_dx, scale_dy* ; (float, float)
: Similar to the `delta` parameter, this is the amount to change the scale of the element for each unit of camera movement. Defaults to `0,0`.

`zoomdelta` = *dzoom* ; (float)
: Specifies the amount that the camera zoom affects the scale of the element. A value of 0 is means the element will not change size during zoom, and a value of 1 will scale the element by the same factor of camera zoom. Defaults to `1`.

`sin.x` = *amplitude, period, phase* ; (float, float, float)
: Specifies sinusoidal movement for the element in the `x`-direction. The first parameter is the amplitude, the second parameter is the period of the motion in game ticks, and the third parameter specifies the initial phase shift of the sinusoidal motion (defaults to 0, i.e., the element will start in the exact middle of its sinusoidal range). This parameter is basically superseded by the `SinX` background controller.

`sin.y` = *amplitude, period, phase* ; (float, float, float)
: Works the same as the `sin.x` parameter, but in the `y`-direction.

Other background element types

Besides `normal`, `anim` and `parallax` background types, there is also a `dummy` type. As its name implies, a `dummy`-type background has no associated graphics. A dummy element's position is affected just like any other element type. A dummy element with an `ID` parameter may serve as a placeholder for the effect of the `zoffsetlink` parameter in the `StageInfo` group.

Background controllers

Background controllers operate on an internal timer that starts at 0 when the round starts, and increases by 1 for every game tick. When the timer reaches the controller's start time, then the controller becomes active. When the timer reaches the controller's end time, then the controller deactivates. If a positive looptime is specified for the controller, then the controller's internal timer will reset to 0 when the looptime is reached.

Background controllers must be grouped under a parent `BGCtrlDef`. You can use multiple `BGCtrlDef`s to separate the controllers into several groups. Each block of `BGCtrlDef` and background controllers may be placed anywhere within the `[BGDef]` section of the `DEF` file. The general format for these blocks is as follows.

```
[BGCtrlDef my_controller_name]
looptime = GLOBAL_LOOPTIME
ctrlID = DEFAULTID_1, DEFAULTID_2, ...

[BGCtrl my_controller_1]
type = CONTROLLER_TYPE
time = START_TIME, END_TIME, LOOPTIME
```

```
ctrlID = ID_1, ID_2, ...
(controller-specific parameters here)

[BGCtrl my_controller_2]
(etc.)
```

GLOBAL_LOOPTIME specifies the number of ticks after which the BGCtrlDef should reset its internal timer, as well as the internal timers of all BGCtrls it contains. To disable the looptime, set it to -1 or omit the parameter.

DEFAULTID_1, DEFAULTID_2, etc., specify the IDs of background elements to be affected by any BGCtrl that doesn't specify its own list of ctrlIDs. You can list up to 10 ID numbers for this parameter. If the line is omitted, then the default will be to affect all background elements.

START_TIME, END_TIME, and LOOPTIME are the times at which the background controller should start acting, stop acting, and reset its internal timer, respectively. If LOOPTIME is omitted or set to -1, then the background controller will not reset its own timer. (Its timer can still be reset by its parent BGCtrlDef if a GLOBAL_LOOPTIME is specified.) The background controller will be continuously active between START_TIME and END_TIME. START_TIME is required, but if END_TIME is omitted then it will default to the same value as START_TIME (so the controller will be active for 1 tick only).

ID_1, ID_2, etc., specify the IDs of background elements for this controller to act on. This list, if specified, overrides the default list specified in the BGCtrlDef. The maximum number of IDs specifiable is 10.

Below is the list of BGCtrl types and their associated parameters.

- null

  As the name implies, this controller does nothing. It is useful mainly for debugging, when you want to quickly disable a controller without commenting the whole thing out. Simply change the type to null and comment out the old type. This controller has no additional parameters.

- Visible

  value = *visible_flag*

  > Sets the visibility status of the elements.

  While active, this controller sets the affected background elements to be invisible (0) or visible (1). Time will still pass for invisible elements (meaning, in the case of animated elements, that the animation will continue to progress even though it can't be seen).

- Enabled

  value = *enabled_flag*

  > Sets the enabled status of the elements.

  This controller either disables (0) or enables (1) the affected background elements. When an element is disabled, it is invisible and time does not pass for it (so, in the case of animated elements, any animation is paused when it's disabled).

- VelSet

  x = *vel_x*

  > Sets the x-velocity of the elements.

  y = *vel_y*

  > Sets the y-velocity of the elements.

  This controller will set the x/y velocity of the affected background elements to the specified values. Velocities are measured in pixels per game tick. You can specify either or both of the x and y parameters. If either is omitted, the element's velocity in that direction will not change.

- `VelAdd`

  `x` = *vel_incr_x*

  Changes the `x`-velocity of the elements by *vel_incr_x*.

  `y` = *vel_incr_y*

  Changes the `y`-velocity of the elements by *vel_incr_y*.

  This controller will add the specified values to the `x`/`y` velocity of the affected background elements. You can specify either or both of the `x` and `y` parameters. If either is omitted, the element's velocity in that direction will not change.

- `PosSet`

  `x` = *pos_x*

  Sets the `x`-position of the elements.

  `y` = *pos_y*

  Sets the `y`-position of the elements.

  This controller will set the `x`/`y` coordinate of the affected background elements to the specified values. You can specify either or both of the `x` and `y` parameters. If either is omitted, the element's coordinate on that axis will not change.

- `PosAdd`

  `x` = *x_displacement*

  Displaces the `x`-coordinate of the elements.

  `y` = *y_displacement*

  Displaces the `y`-coordinate of the elements.

  This controller will displace the `x`/`y` coordinate of the affected background elements by the specified values. You can specify either or both of the `x` and `y` parameters. If either is omitted, the element's coordinate on that axis will not change.

- `Anim`

  `value` = *action_no*

  Changes the animation displayed by the affected elements to the specified animation number.

- `SinX`

  `value` = *amplitude, period, offset*

  Changes the amplitude, period, and phase offset for the affected elements' sinusoidal motions in the `x`-direction. These values have the same effect as they do for the `sin.x` background element parameters.

- `SinY`

  `value` = *amplitude, period, offset*

  Changes the amplitude, period, and phase offset for the affected elements' sinusoidal motions in the `y`-direction. These values have the same effect as they do for the `sin.y` background element parameters.

Simple Example

Suppose we want to make a person walk back and forth from (-300,0) to (300,0), right behind the main characters. We'll use background controllers to accomplish this task.

First, define the walking animations. Say that the character's walking sprites are 10,0 through 10,3 and that they face to the right.

```
; Walk right
[Begin Action 10]
10,0,0,0,6
10,1,0,0,6
10,2,0,0,6
10,3,0,0,6

; Walk left
[Begin Action 11]
10,0,0,0,6,H
10,1,0,0,6,H
10,2,0,0,6,H
10,3,0,0,6,H
```

Now start the character off at the far left edge of his range.

```
[BGDef]
(...)

[BG Peregrinator]
type = anim
actionno = 10
id = 10
start = -300, 0
delta = 1,1
```

Let's give Peregrinator a comfortable ambling speed of 2 pixels per tick. The one-way distance for his walk is 600 pixels, which will take 300 ticks. In total, it'll take him 600 ticks to make the round trip. Using this knowledge, set up the background controllers appropriately: since the entire situation repeats every 600 ticks, we can set the global looptime to 600.

```
[BGCtrlDef Peregrinator]
; reset the whole deal every 600 ticks.
looptime = 600
ctrlID = 10

; Set velocity of 2 pixels/sec rightward at time 0.
[BGCtrl Walk Right]
type = VelSet
time = 0
x = 2

; Set velocity of 2 pixels/sec leftward at time 300.
[BGCtrl Walk Left]
type = VelSet
time = 300
x = -2
```

And that's it! You can make the walk look better by having Peregrinator slow down and display a turning animation at each end of his walk. This would entail use of the `VelAdd` and `Anim` controllers. If you want Peregrinator to stop and start at regular intervals as he goes from one end to the other, you could create more `VelSet` and `Anim` controllers with their own individual looptimes (to get the behavior to repeat at regular intervals).