Effector Sets™ I & II

for AlphaMania™ 2

User Manual

Table of Contents

Part I: Overview

What is an Effector Set and why do you want one?

The Effector Sets are Xtras from Media Lab that work in conjunction with AlphaMania 2 (or later) to give Director users incredible power and real-time control over special effects. This might be as simple as quickly rotating a button 45 degrees using only one castmember or as complex as making a graphic ripple like water.

Best of all, the effects in each Effector Set have been thoroughly tailored to the needs of Director users designing interactive content. Between AlphaMania 2's advanced drawing methods and the variety of special effects available, authors have never had as much power at their fingertips.

Effector Sets Are Easy To Use

Included with each Effector Set Xtra is a point-and-click user interface called the SetFX movie that allows even non- programmers powerful access to the various features of the included effects. Director 6 drag-and-drop behaviors are also ideal ways for novices to get complex interactive functionality from effects. For this reason, many behaviors are included with the products.

About This Manual

Part 2 of this manual contains detailed information about the nine effects included with Effector Set I, and Part 3 covers the five effects in Effector Set II. Each effect is described in two parts. A "Basics" part describes in general how the effect works. An "Advanced" part lists in detail every possible argument to this effect and includes a table you can use to determine default values for omitted arguments, and also which arguments are required for each mode.

In part 4 and beyond you will find reprinted sections from the AlphaMania 2 User Guide that explain the features of AlphaMania that are important to understand in order to take full advantage of AlphaMania 2 with Effects. (You'll also find the license agreement and Media Lab contact info.) If you are not familiar by now with AlphaMania 2 concepts like "Drawing Method" and "Animation Mode" or the basics of adding effects to castmembers and sprites you'll need to go back and read the AlphaMania 2 manual or read the final sections of this manual.

Installing an Effector Set

Many versions of the Effector Sets come with installers that will automatically place the necessary files in the necessary places. If you would like to install manually, you will need to place the following in your Director "Xtras" folder:

AlphaMania 2.x32	The Xtra
Register Effector Set _.dxr	Allows you to unlock the Xtra once you have purchased the appropriate unlock code.
SetFX.dir	Included with AlphaMania 2. Allows editing of effects without lingo.
FX Set 1.fxm	Required to use SetFX movie with Effector Set I.
FX Set 2.fxm	Required to use the SetFX movie with Effector Set II.

Registering an Effector Set

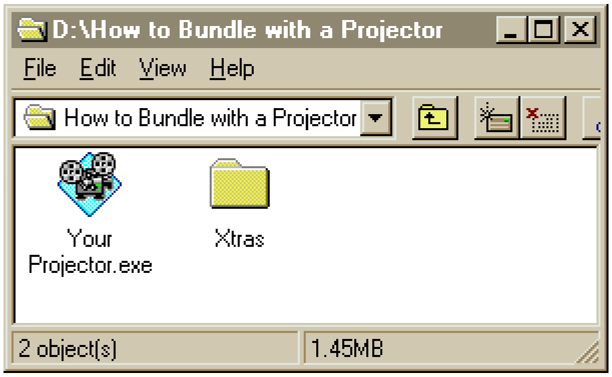
Choose "Register Effector Set I" or "Register Effector Set II" from the Xtras menu, as appropriate. Each of the effector sets must be registered seperately. When you have finished reading the license agreement click "I Agree". You will then be able to enter your unlock code and register the product. Until you register the product, AlphaMania 2 castmembers with effects from an Effector Set applied will have a box drawn around them at all times.

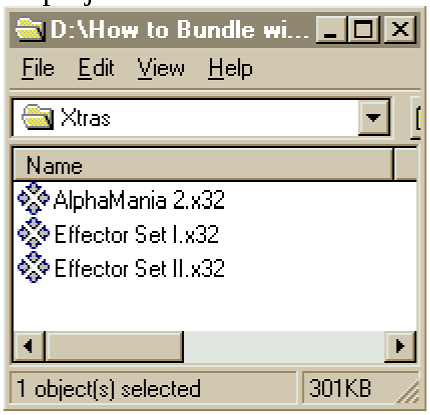
System Requirements

Use of this product requires Director 5 or Director 6 and at least 8-bit color.
MacOS: System 7 or later. (68K or PowerPC)
Windows: 95, 98, NT3.51, NT4, or later.

Packaging an Effector Set with a Projector

An Effector Set Xtra must be distributed with any of your projectors that contain AlphaMania castmembers that use effects in that Xtra. An Effector Set Xtra & AlphaMania 2 must reside in a folder named "Xtras" that appears in the same folder as your projector.


D:\How to Bundle with a Projector
File
Edit
View
Help
How to Bundle with a Projector
Your Projector.exe
Xtras
2 object(s)
1.45MB



The best and most reliable way to do this is the following:

1. (Director 6 and later) When creating a projector, turn OFF the flag for 'Check Movies for Xtras' that appears in the 'Options' dialog when selecting files.
2. Once the projector is created, see that it is accompanied by a directory named 'Xtras' (note the spelling with an 's'). And make sure that directory contains the AlphaMania 2 Xtra and Effector Set Xtra(s).

The second best way of doing this (Director 6 and later only):

1. When creating a projector, turn ON the flag for 'Check Movies for Xtras' that appears in the 'Options' dialog when selecting files. This should automatically bundle the AlphaMania Xtra with your projector.
2. From the Create Projector dialog, select the 'Add' button, and go choose the Effector Set Xtra(s) that your movie will need.

When using the second method the Projector will bundle the Xtras into itself and when it runs it will automatically unpack the Xtras, run, and when quitting it deletes them. However, this has a few problems associated with it such as Director 6's notorious Error -35 bug.

Part 2: Guide to Effector Set I’s Effects

Effector Set I contains many effects that are as useful as any tool Director developers have ever had access to. These include rotate, magnify, RGB, and HSB. At the same time we didn’t want this pack to be too boring so we threw in some peacock effects that are useful simply because they’re very cool like seurat, and swirl.

We’ve divided each effect into a basic information section and a more complete reference section which includes a table of all possible arguments to an effect along with their defaults. We recommend browsing the basics to familiarize yourself with what the effects can do, playing with the effects, and then using the reference sections when you want to look up a specific technique or argument name.

We’ll start with everyone’s favorite effect, rotate:

Rotate Basics

Effect Type: Complex

With the rotate effect you can rotate your AlphaMania sprites freely! The rotate effect is fast, powerful, and very flexible. Sprites can be rotated to an angle, by an angle, rotate relative to some other point on the stage, or just set to rotate infinitely round and round. The rotate effect can even flip the sprite as it rotates. You can give rotation degree or radian arguments, and it even contains special functions to help you figure out things like angular velocity.

Uses

Every Director user has wanted live rotation for a long time, so we think you’ll be putting this effect to good use. But in addition there are a number of uses that people don’t tend to think of. With the RelToPoint animation mode (see below) things like needle-meters, clocks, and guns that follow targets become very easy. In fact, we’ve provided a bunch of behaviors to accommodate you. We’ve included a simple Asteroids game with four asteroids spinning in different directions at different speeds and a user-controllable rotating spaceship. This movie uses only two AlphaMania castmembers and two behaviors!

Techniques

Shadows and Highlights on Rotating Objects

If you have a dial with a cool shadow and a nice highlight you’ll probably be a tad disappointed when you attempt to rotate said dial only to have the shadow and highlight rotate too. One solution is to tell your audience that the dial not only controls whatever it controls but also controls the angle of the sun, relative to the dial. Another solution is to NOT rotate the highlights and shadows. To do this, make a separate AlphaMania castmember that is just the highlights and shadows, and put it on top of your rotating sprite. Remember, AlphaMania castmembers are made do semi-transparent effects like shadows and highlights. If you need the shadows to compress and expand a little as the object rotates, use the scale effect on them. If the object you are rotating is unusually shaped, then this technique may not work for you.

The third solution to this problem is to use Effector Set II, which contains dynamic shadow and highlight effects. Simply apply these effects after the rotate effect, and then it will all work seamlessly.

Special Considerations

The Sprite Rectangle
In Director, no sprite can draw outside of its rectangle. This can be a problem when rotating a sprite because any shape that is not a circle will change the dimensions of the rectangle needed to contain it as it rotates to different angles. When the rotate effect is added to a castmember, the default sprite rectangle will automatically be increased so that the sprite will not be cropped when it rotates. But if the effect is added at the sprite level (like a Behavior), then it may crop as it rotates. The solution is simple: make the sprite rectangle bigger!

Using Rotate with Scale in Follow Mode
If you are using rotate on a sprite that also has the scale effect in follow mode, then you will see the sprite rectangle cropping even if rotate is applied to the castmember. Adjust the percentage of the scale effect to approximately 70%. The exact percentage value needed to avoid the cropping is dependent on the dimensions of the sprite and how much it is rotating, so you may need to tweak and experiment.

Confusion Over Relative To Point
In the RelToPoint animation mode (see below), the sprite turns to face a point somewhere on the stage. But very often user's want an effect more like clock hands, where not only does the hand always face the center of the clock, but also revolves around that point. There are, in fact, two motions going on, the clock hand is rotating (upon itself) to change it facing/attitude, and it is also actually MOVING around the center point. This second motion, which we call revolution, requires that the sprite be moved on the stage.

Custom Animation Modes

The rotate effect has two custom animation modes: infinite and relativeToPoint.

Infinite
In rotate's implementation of the infinite animation mode, sprites are made to rotate infinitely around and around. The only required parameter is the number of frames in which to complete a revolution, if this number is positive then it rotates clockwise, if negative then counterclockwise. The SetFX movie can set up these settings, or the 'Rotate Forever' behavior can be used.

RelativeToPoint
In this mode, rather than rotating to or by angles, the sprite will rotate to face some point on the stage. This mode is VERY useful for setting up clocks, pointers, meters, and all sorts of 'relative' rotation effects. It's all automatic, you need to only provide a location to face, and optionally an offset angle (how much to 'face away'). That's it. There are two behaviors, 'Face Mouse' and 'Face Sprite' ready to implement this mode in an easy way.

Behaviors

A number of behaviors that implement the rotate effect for you are included in the Effector Set I behavior library. If you use them, don't forget to enlarge the sprite rectangle of the sprite!

Rotate
This behavior let's you simply set an angle, and it will rotate the sprite to that angle in static mode. You can also specify the interpolation.

Rotate Forever
This behavior will set up the infinite mode for you. With it you can set the speed, direction, and interpolation of the rotation.

RotateOnClick
With this behavior the sprite will rotate when it's clicked and continue to rotate while the mouse is down. When the mouse is up, it stops. If clicked again, it resumes rotation. It let's you set the speed of the rotation, it's direction, and the interpolation.

FaceMouse
This behavior is a great one that makes the sprite constantly face the mouse. With it you can set the offset angle (how much it 'faces away').

FaceSprite
This behavior is another winner that makes the sprite constantly face another sprite. You must specify a sprite to face, and can optionally set the offset angle.

DialPush
Rotate a dial with the mouse!

Custom Functions
Rotate has a plethora of custom functions you can call and manipulate. Here is a complete listing:

UseRadians
UseRadians(sprite x, #rotate, true or false)
Calling this function will make the sprite treat all subsequent custom function calls as if their degree arguments are in radians rather than degrees. Call this function with a 'false' argument to reset the sprite to regular degree usage.

Flip
Flip(sprite x, #rotate)
Flip(sprite x, #rotate, flipFlag)
If this function is called without the flipFlag argument, then the sprite will have it's flip state toggled. When the flipFlag argument is used, then the sprite will have its flip state set to that argument.

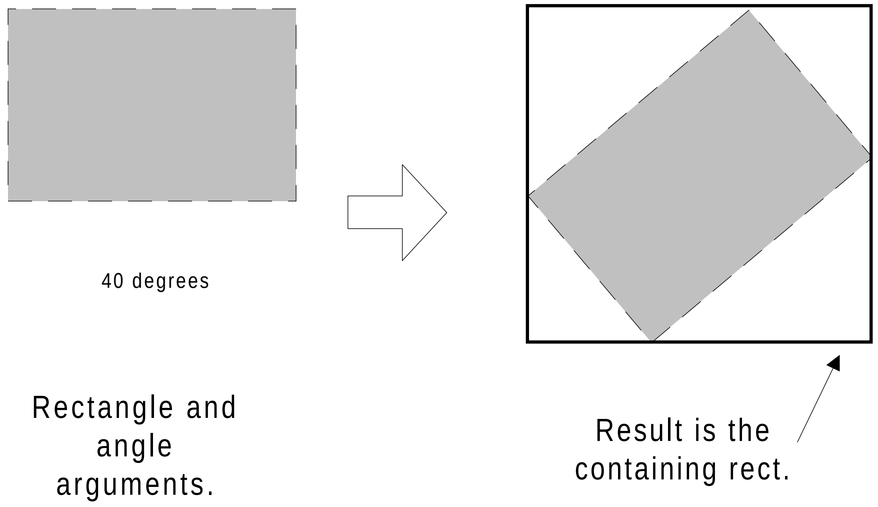
GetAngle
GetAngle(sprite x, #rotate)
This argument will return the current angle of the sprite.

GetVector
GetVector(sprite x, #rotate, distance)
GetVector(sprite x, #rotate, distance, degrees)
This function converts a distance and an angle into the sub vectors X and Y. This function returns a point with the X and Y vectors in the x and y positions. If no angle is provided, the current angle of the sprite is used. If you are programming a rotating spaceship and want it to travel 5 pixels at its current angle, just call this function and then add the result to your ships current location. No trigonometry required!

GetRotatedRect

GetRotatedRect(sprite x, #rotate)
GetRotatedRect(sprite x, #rotate, degrees)
GetRotatedRect(sprite x, #rotate, degrees, rect)

This function takes a rect and an angle and calculates the new containing rect. If no angle is provided it uses the sprites current angle, and if no rect is provided it uses the rect of the castmember at the sprite's location.


40 degrees
Rectangle and angle arguments.
Result is the containing rect.

IsPointInRotatedRect

IsPointInRotatedRect(sprite x, #rotate, point)
IsPointInRotatedRect(sprite x, #rotate, point, degrees)
IsPointInRotatedRect(sprite x, #rotate, point, degrees, rect)

This function returns true or false if a point is in the rotated rectangle of the sprite. The current rectangle and angle of the sprite are used as defaults, but if desired the user can pass in custom arguments.

RelToPoint

RelToPoint(sprite x, #rotate, xLocation, yLocation)
RelToPoint(sprite x, #rotate, xLocation, yLocation, offsetAngle)
RelToPoint(sprite x, #rotate, xLocation, yLocation, offsetAngle, flipFlag)
RelToPoint(sprite x, #rotate, xLocation, yLocation, offsetAngle, flipFlag, interpolation)

This is the mode accessor function for the #relativeToPoint mode. It causes this mode to immediately become active in sprite, and the parameters to take place. This is provided as an option to the more traditional calling of the rotate effect (ie. Rotate(sprite x, [animMode:#relToPoint, …]), but provides no unique functionality.

InterpolateNow

InterpolateNow(sprite x, #rotate)

This function causes the sprite to immediately redraw interpolated, but does not change the flag for the sprites overall interpolation setting.

SetInterpolation

SetInterpolation(sprite x, #rotate, interpolation)

Rotate Reference

Rotate Effect Argument Table
Effect Symbol: #rotate

Properties	Type	Value	Default	Exclusive
#animMode	symbol	#static, #range, #infinite, #pendulum, #relativeToPoint	#static	
#numFrames	integer	1+	-	required in pendulum and range
#framesPerRevolution	integer	0 not allowed, negative means rotate counter-clockwise	-	infinite mode only
#interpolation	integer	0, 1, 2	0	-
#degrees	integer	positive values are clockwise	0	(A)
#radians	integer	positive values are clockwise	0	(A)
#startDegrees	integer		current	valid in pendulum, infinite, and range only
#startRadians	integer		current	valid in pendulum, infinite, and range only
#endDegrees	integer		#degrees	
#endRadians	integer		#radians	
deltaDegrees	integer	positive values are clockwise	0	(B) valid in static, pendulum, and range only
deltaRadians	integer	positive values are clockwise	0	(B) valid in static, pendulum, and range only
#flip	integer	true or false	false	
#relativeTo	point	stage coordinates	(0,0) on stage	
#relativeToX	integer			
#relativeToY	integer			
#offsetAngle	integer	in degrees	0	valid in RelToPoint mode only
#offsetAngleRad	integer	in radians	0	valid in RelToPoint mode only
#easeIn	integer	frames	0	valid in pendulum and range only
#easeOut	integer	frames	0	valid in pendulum and range only

(A) parameters cannot be used with (B) parameters.

Rotate Parameters by Mode:
The following parameters are valid in these modes:

All Modes:
interpolation
flip

Static:
degrees
radians
deltaDegrees
deltaRadians
To set the rotation to an angle use the degrees or radians parameter. But to rotate BY an angle use the deltaDegrees or deltaRadians parameters.

Range:
numFrames
degrees
radians
endDegrees
endRadians
deltaDegrees
deltaRadians
startDegrees
startRadians
easeIn
easeOut

In #range mode the sprite will rotate to the endPosition from it's current (or start) position. If the endPosition is a number greater than its starting position then the sprite will rotate clockwise. If it is less than the starting position then it will rotate counter-clockwise. No abbreviating occurs, so if the sprite is at 0 degrees and it is given an endPosition of 720 degrees then it will rotate three times clockwise. It's new starting position will be 720 degrees for the next call to range.

Pendulum
numFrames
degrees
radians
endDegrees
endRadians
deltaDegrees
deltaRadians
startDegrees
startRadians
easeIn
easeOut

In #pendulum mode the sprite will rotate back and forth between a start and end point.

Infinite
framesPerRevolution
startDegrees
startRadians
easeIn

In #infinite mode the sprite will simply rotate forever. A positive value for 'framesPerRevolution' will cause it to rotate clockwise, a negative value will cause it to rotate 'counter-clockwise'.

RelToPoint
relativeTo
relativeToX
relativeToY
offsetAngle
offsetAngleRad

This is a custom mode for the rotate effect only. In this mode, the sprite will always 'face' the point at relativeToPoint, no matter where the sprite is on-screen. If desired, an offsetAngle argument can be provided, in which case it will 'face away' by that parameter.

Custom Functions:

RelToPoint
RelToPoint(sprite x, < position or #rotate>, xLocation, yLocation)
RelToPoint(sprite x, < position or #rotate >, xLocation, yLocation, offsetAngle)
RelToPoint(sprite x, < position or #rotate >, xLocation, yLocation, offsetAngle, flipFlag)
RelToPoint(sprite x, < position or #rotate >, xLocation, yLocation, offsetAngle, flipFlag, interpolation)
This is the mode accessor function for the #relativeToPoint mode.

InterpolateNow
InterpolateNow(sprite x, < position or #rotate >)
This function causes the sprite to immediately redraw interpolated, but does not change the flag for the sprites overall interpolation setting.

SetInterpolation
SetInterpolation(sprite x, < position or #rotate >, interpolation)

UseRadians
UseRadians(sprite x, <position or #rotate >, true / false)
Calling this function will make the sprite treat all subsequent function calls as if their degree arguments are in radians rather than degrees. Call this function with a 'false' argument to reset the sprite to regular degree usage.

Flip
Flip(sprite x, <position or #rotate >)

Flip(sprite x, <position or #rotate >, flipFlag)
If this function is called without the flipFlag argument, then the sprite will have it's flip state toggled. When the flipFlag argument is used, then the sprite will have its flip state set.

GetAngle

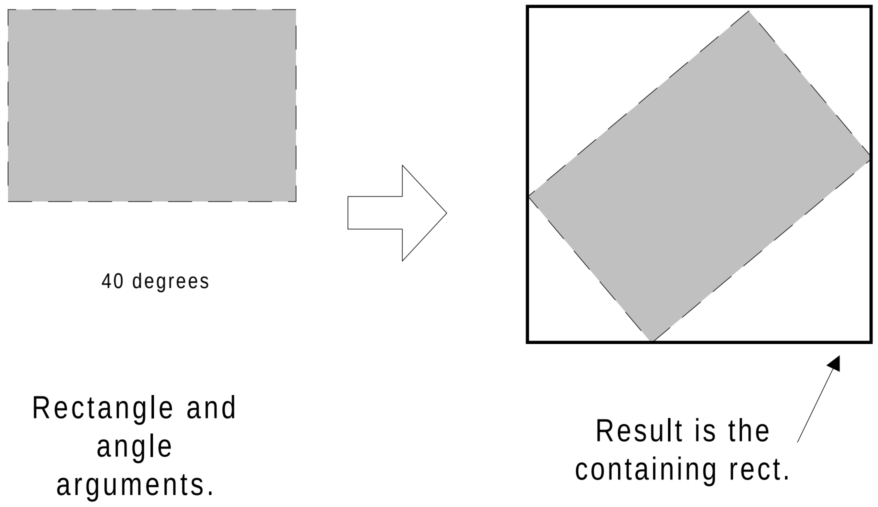
GetAngle(sprite x, <position or #rotate >)
This argument will return the current angle of the sprite.

GetVector

GetVector(sprite x, <position or #rotate >, distance)
GetVector(sprite x, <position or #rotate >, distance, degrees)
This function converts a distance and an angle into the sub vectors X and Y. This function returns a point with the X and Y vectors in the x and y positions. If no angle is provided, the current angle of the sprite is used.

GetRotatedRect

GetRotatedRect(sprite x, <position or #rotate >)
GetRotatedRect(sprite x, <position or #rotate >, degrees)
GetRotatedRect(sprite x, <position or #rotate >, degrees, rect)
This function takes a rect and an angle and calculates the new containing rect. If no angle is provided it uses the sprites current angle, and if no rect is provided it uses the rect of the castmember at the sprite's location.


40 degrees
Rectangle and angle arguments.
Result is the containing rect.

IsPointInRotatedRect

IsPointInRotatedRect(sprite x, <position or #rotate >, point)
IsPointInRotatedRect(sprite x, <position or #rotate >, point, degrees)
IsPointInRotatedRect(sprite x, <position or #rotate >, point, degrees, rect)
This function returns true or false if a point is in the rotated rectangle of the sprite. The current rectangle and angle of the sprite are used as defaults, but if desired the user can pass in custom arguments.

Arguments

framesPerRevolution
Used in #infinite mode, this argument simply tells the effect how long to take to complete a revolution. The smaller the argument, the faster the sprite rotates. Normally, the sprite will rotate in the clockwise direction, but by making this argument negative you can make the sprite rotate counter-clockwise. Any non-zero number is valid. This argument is redundant with the 'numFrames' parameter. If you'd rather you can use the 'numFrames' parameter in the infinite animation mode.

degrees
This is the destination to which you make the sprite rotate. Positive values progress clockwise (0 degrees being at 'twelve o'clock', 90 degrees is at 3 o'clock, etc.), and negative values progress counter-clockwise (-90 degrees at 9 o'clock, -270 degrees at 3 o'clock, etc.).

radians
Instead of degrees, you can use radians as an argument to rotate. This is especially convenient if you are using Director's trigonometric functions which work in radians.

deltaDegrees
deltaRadians
Rather than supplying the 'degrees' argument above, this argument can be provided in #static, #pendulum, and #range modes. It simply tells the sprite to rotate BY 'deltaDegrees', rather than TO 'degrees'.

flip
When this flag is true then the sprite is 'flipped' left-right before any rotation occurs. Valid values are true (1) or false (0). To flip a sprite around a particular axis, turn the flip flag on and rotate the sprite 2ϕ where ϕ is the angle of the axis.

interpolation
This arguments sets the interpolation setting for this effect. With interpolation on the rotated sprite will be smoother and nicer looking, but draw slower. With interpolation off, the sprite will draw faster, but be rougher. There is a get-best-of-both-worlds interpolation setting that will make the sprite only interpolate when paused. A pause state occurs whenever the sprite is in #static mode, when in #range mode after it has finished the animation, or when the effect has been paused with a call to PauseEffect or PauseAllEffects. Note, the normal lingo command 'pause' has no effect on the pause state of this effect.

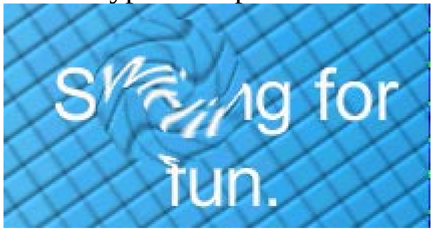
Settings:	
Interpolation Off:	0
Interpolation On:	1
Interpolate When Paused	2

relativeTo
relativeToX, relativeToY
These two arguments set the point which the sprite will rotate to 'face' when in #relativeToPoint mode. These coordinates are stage relative.

offsetAngle
when in #relativeToPoint mode the sprite constantly ‘faces’ a defined point. If desired the sprite can be made to constantly face away from that point by an angle defined by this argument.

Swirl Basics

Effect Type: Complex



What can words say that the above picture doesn't illustrate? With swirl you can swirl your sprites. Swirl is very straightforward, and very cool. The amount of swirl is specified by the degrees or radians to twist around the center point.

Recommendations

The swirl effect always performs a circular swirl, never an elliptical one. For this reason, you will probably have the most satisfying results when using swirl with objects that aren't too dramatically oblong. Swirl is particularly impressing in apply method, swirling everything it's dragged over. We recommend using swirl values that are lower than 180 degrees. More than that results in a whole lot of swirl.

Behaviors

Swirl has a few behaviors prepared for you.

Swirl

This behavior let's you set up a simple static mode swirl. You can set the degrees of the swirl and turn the interpolation on or off.

Rollover Swirl

This behavior swirls a sprite, and then, if the mouse is over, it dramatically unswirls. Simply set the initial swirl, and how fast you want the sprite to unfold.

Custom Functions

UseRadians

UseRadians(sprite x, #swirl, true / false)

Calling this function will make the sprite treat all subsequent function calls as if their degree arguments are in radians rather than degrees. Call this function with a 'false' argument to reset the sprite to regular degree usage.

GetAngle

GetAngle(sprite x, #swirl)

This argument will return the current angle of the sprite swirling.

InterpolateNow

InterpolateNow(sprite x, #swirl)

This function causes the sprite to immediately redraw interpolated, but does not change the flag for the sprites overall interpolation setting.

SetInterpolation

SetInterpolation(sprite x, #swirl, interpolation)

Swirl Reference

Effect Symbol: #swirl

Properties	Type	Value	Default	Exclusive
#animMode	symbol	#static, #range, #infinite, #pendulum	#static	
#numFrames	integer	1+	-	required in pendulum, infinite, and range
#interpolation	integer	0, 1, 2	1	-
#degrees	integer		0	
#radians	integer		0	
#startDegrees	integer		current	
#startRadians	integer		current	
#endDegrees	integer		#degrees	
#endRadians	integer		#radians	
#deltaDegrees	integer		0	
#deltaRadians	integer		0	
#easeIn	integer	frames	0	valid in pendulum, infinite and range only
#easeOut	integer	frames	0	valid in pendulum, infinite and range only

Custom Functions

GetAngle
GetAngle(sprite x, <position or #swirl>)
This argument will return the current angle of the sprite.

InterpolateNow
InterpolateNow(sprite x, < position or #swirl >)
This function causes the sprite to immediately redraw interpolated, but does not change the flag for the sprites overall interpolation setting.

SetInterpolation
SetInterpolation(sprite x, < position or #swirl >, interpolation)

UseRadians
UseRadians(sprite x, <position or #swirl >, true / false)
Calling this function will make the sprite treat all subsequent function calls as if their degree arguments are in radians rather than degrees. Call this function with a 'false' argument to reset the sprite to regular degree usage.

Arguments

degrees
The amount of swirl is specified by the degrees to twist around the center point. Positive values progress clockwise (0 degrees being at ‘twelve o’clock’, 90 degrees is at 3 o’clock, etc.), and negative values progress counter-clockwise (-90 degrees at 9 o’clock, -270 degrees at 3 o’clock, etc.).

radians
Instead of degrees, you can use radians. This is especially convenient if you are using Director’s trigonometric functions which work in radians.

deltaDegrees
deltaRadians
Rather than swirling TO an angle, you can swirl BY an angle using the deltaDegrees or deltaRadians parameters.

interpolation
This arguments sets the interpolation setting for this effect. With interpolation on the scaled sprite will be smoother and nicer looking, but draw slower. With interpolation off, the sprite will draw faster, but be rougher. There is a get-best-of-both-worlds interpolation setting that will make the sprite only interpolate when paused. A pause state occurs whenever the sprite is in #static or #follow modes, when in #range mode after it has finished the animation, or when the effect has been paused with a call to PauseEffect or PauseAllEffects. Note, the normal lingo command ‘pause’ has no effect on the pause state of this effect.

Settings:	
Interpolation Off:	0
Interpolation On:	1
Interpolate When Paused	2

HSB Basics

Effect Type: Simple

HSB stands for Hue, Saturation, and Brightness. You may be familiar with this as an alternate way to describe a color for the computer. The HSB effect lets you adjust the hue, saturation, and/or brightness levels of every pixel of your AlphaMania sprite or member, thus ‘colorizing’ the sprite. Not only can you adjust and shift these values, you can force them to be a certain value. This effect is great for creating animated duotone effects, grayscale to color fade ins, and can even be used to cycle colors in any color depth. This is one of the most powerful effects in Effector Set I.

When using this effect, you will mostly be adjusting hue, saturation, and brightness values. Hue is measured in degrees. 0 degrees is red, 120 degrees is green, and 240 degrees is blue. Of course, many of the manipulations you will perform with this effect will be ‘relative’ to the pixel’s original hue, so a value may not mean an absolute color (though it can if you use the ‘forceHue’ parameter). Saturation and Brightness are both measured on a scale of 0 - 255.

Uses

HSB has a long string of potential uses, especially when used in tandem with other effects. But a short list of immediate uses are to

1. eliminate extra castmembers when they need to differ only in their color. Just use one castmember to spawn many sprites each with unique coloring achieved by HSB.
2. colored lights that illuminate other objects
3. color cycling (in any color depth). While this isn’t exactly traditional 8 bit color cycling, it’s still fast and since it works in all color depths, much more useful.
4. duotone or grayscale imagery from full color originals - fade in between. Don’t forget by using the HSB effect and AlphaMania castmembers with the apply or reveal draw methods, your artwork can all be regular bitmap castmembers, and yet still receive the benefits of these effects.
5. colorize objects. Cars, clothes, hair, even skin tones can all be adjusted subtly or sensationally.

Special Considerations

Absolute Force vs. Relative Shift adjustments

The HSB effect can use the ‘forceHue’ parameter to set the hue values for all of the effected pixels to the same value, or it can use the ‘hueShift’ parameter to shift the hue values for all the pixels by the same amount. HSB is capable of absolute forcing or relative shifting any component of an image, and can mix different adjustment types at the same time, and can animate any of these.

Animating (Fading) Between Colors

When fading between two colors using the HSB effect, you may notice that the fade may pass through intermediate colors. This is a side effect of the HSB model. If you want direct fading, use the RGB effect.

Custom Animation Modes:

Infinite

HSB provides a custom implementation of the infinite animation mode. In this mode, the hue will cycle through a complete revolution (360 degrees) in the number of frames provided. The other vectors (saturation and brightness) will advance to their endpoint, and then re-start, looping this way continuously. By just putting the infinite mode on a sprite with only the 'numFrames' parameter you get instant color cycling.

Using The Set FX Movie with HSB

Because of the sprite limitation imposed by Director 5, the current incarnation of the Set FX movie only supports the 'relative' adjustments. It also does not have support for the infinite mode. But both of these are accessible through behaviors. A future version of the Set FX movie (for Director 6) will incorporate the other controls.

Behaviors

HSB Shifter

This behavior lets you set the relative shifting of the three components.

Duotone

This behavior lets you set the forcing of the hue to create duotone effects.

Cycle Hue

This behavior gives you instant color cycling. Simply tell it how fast you want the colors to cycle. This works in ANY color depth. This behavior will cycle using relative shift adjusting, but if you apply the Duotone behavior to the sprite first, it will cycle using forced absolute adjustments.

Color Fader

This behavior lets you fade a sprite in and out from grayscale over time.

Rollover Color Fader

This behavior lets you fade a sprite in or out as the mouse comes over the sprite. You can choose to start gray and fade to color as the mouse approaches, or start color and fade to gray. And you can control the speed.

Rollover Hue Shifter

This behavior lets you shift the hue of a sprite as the mouse passes over it. You can choose the mouse over and mouse off shift amount, and the speed of the fade.

HSB Reference

Effect Symbol: #hsb

Properties	Type	Value	Default	Exclusive
#animMode	symbol	#static, #range, #infinite, #pendulum	#static	
#numFrames	integer	1+	-	required in infinite, range, and pendulum
#hueShift	integer	measured in degrees, 0 is no change	0	
#satShift	integer	-255 to 255, 0 is no change	0	
#brightShift	integer	-255 to 255, 0 is no change	0	
#startHue	integer	measured in degrees, 0 is no change	current	valid in pendulum and range only
#startSat	integer	-255 to 255, 0 is no change	current	invalid in static mode
#startBright	integer	-255 to 255, 0 is no change	current	invalid in static mode
#endHue	integer	measured in degrees, 0 is no change	#hueShift	
#endSat	integer	-255 to 255, 0 is no change	#satShift	
#endBright	integer	-255 to 255, 0 is no change	#brightShift	
#easeIn	integer	frames	0	valid in pendulum and range only
#easeOut	integer	frames	0	valid in pendulum and range only
#forceHue	integer	0 to 360,	-	can't be used with hueShift
#forceSat	integer	0 to 255	-	can't be used with satShift
#forceBright	integer	0 to 255	-	can't be used with brightShift

Hue, Saturation, and Brightness

Every pixel can have it's color described by three components, hue, saturation, and brightness. Unlike RGB, these components do not all fall on the same scale.

Hue is measured in degrees. 0 degrees is red, 120 degrees is green, and 240 degrees is blue, and 360 degrees is red again. Hue can go round and round with no interruption.

Saturation and Brightness are both measured on a scale of 0 to 255, and have discreet endpoints.

Arguments

hueShift, satShift, brightShift
These arguments result in a relative shift adjustment in the color components. So a brightShift of 10 makes everything 10 steps brighter. This is in contrast to the absolute force adjustments described below.

forceHue, forceSat, forceBright
These arguments result in an absolute forcing of a color component. So, a forceBright of 10 forces every pixel to have a brightness component of 10, making everything very dark. Forcing all three components will result in simply making the entire sprite one color.

startHue, startSat, startBright
These arguments form the start points for a shift type adjustment. So a startBright of 10 and an endBright of 20 means start by raising the brightness of every pixel by 10 and raise it by a net of 20 across time.

endHue, endSat, endBright
These arguments form the end points for a shift type adjustment. They are redundant with the hueShift, satShift, and brightShift parameters, and are included for readability.

Mixing Shift and Force Parameters

The ‘shift’ and the ‘force’ arguments can be mixed together. In any mode you can mix shift and force arguments that aren’t affecting the same component. For example you could shift the hue, and force the saturation at the same time.

But the use of shift and force parameters on the same component is limited and dependent on which animation mode the sprite is using.

In static mode, no single component can be shifted and forced at the same time. In other words, when calling HSB with the static mode, hueShift and forceHue cannot be used at the same time.

In the animating modes, the shift and force arguments for a single component can be used at the same time. In these cases, the force value will be treated as the starting point, and the shift will be a net delta shift adjustment. Example:

```
hsb(sprite x, [animMode:#range, numFrames:15, forceHue:120, hueShift:30])
```

In the above statement the sprite would have it’s hue forced to 120 (green) and then across the next 15 frames would have it’s hue adjusted by 30 to 150 (greenish blue).
Since the force parameters take the place of the starting point, in the animating modes, the start point and force parameters for a single component cannot be used at the same time. Example:

```
hsb(sprite x, [animMode:#range, numFrames:15, startHue:15, forceHue:20]) - - bad code!!
```

The above statement will NOT work.

RGB Basics

Effect Type: Simple

RGB stands for red, green, and blue. It is the standard model for describing color to the computer. And now, it’s the name for an exciting new effect in Effector Set I. RGB is very similar to HSB. It let’s you adjust the red, green, and blue components of every pixel in the sprite, thus colorizing the sprite. All sorts of color fades and effects are possible. Unlike HSB, RGB when animating (fading) between colors will take the shortest path through the RGB colorspace, rather than passing through intermediate colors.

Uses

RGB has a number potential uses, especially when used in tandem with other effects. Here are a few:

1. eliminate extra castmembers when they need to differ only in their color. Just use one castmember to spawn many sprites each with unique coloring achieved by RGB.
2. colored lights that illuminate other objects
3. button hilites, and other colorizing needs.

Behavior

RGB Shifter

This behavior provides a quick way to adjust colors of a sprite. It defaults to the static animation mode.

RGB Reference

Effect Symbol: #rgb

Properties	**Type**	**Value**	**Default**	**Exclusive**
#animMode	symbol	#static, #range, #infinite, #pendulum	#static	
#numFrames	integer	1+	-	required in infinite, pendulum, and range
#redShift	integer	-255 to 255, 0 is no change	0	
#greenShift	integer	-255 to 255, 0 is no change	0	
#blueShift	integer	-255 to 255, 0 is no change	0	
#startRed	integer	-255 to 255, 0 is no change	current	invalid in static mode
#startGreen	integer	-255 to 255, 0 is no change	current	invalid in static mode
#startBlue	integer	-255 to 255, 0 is no change	current	invalid in static mode
#endRed	integer	-255 to 255, 0 is no change	#redShift	
#endGreen	integer	-255 to 255, 0 is no change	#greenShift	
#endBlue	integer	-255 to 255, 0 is no change	#blueShift	
#easeIn	integer	frames	0	invalid in static mode
#easeOut	integer	frames	0	invalid in static mode

Arguments:

redShift, greenShift, blueShift, (and variants)

These arguments determine what will be added the red, green, and blue pixels of each component to adjust it's color. These can be positive or negative numbers. The pixel components range from 0 - 255, but any value can be used for the shift arguments.

Magnify Basics

Effect Type: Simple

The Effector Set I magnify effect is unlike any magnification you may have seen in the past. It can zoom in or out to any percentage (like 117%), not just even multiples (200%, 300%, etc.), and it does so smoothly. You can also set the center point of the magnification, it can be anywhere on the stage. Magnify is unlike the scale effect included with AlphaMania in that the actual size and shape of the magnifying sprite doesn't change. (A magnifying glass doesn't actually make something bigger, it just let's you see it better.)

Uses

Besides using this effect to create magnifying glasses, you can also use this effect to create a 'bump under the rug' embossing effect. Taking a round, slightly feathered sprite, put the AlphaMania castmember into the apply draw method, and magnify the sprite at about 110%. As it is dragged over the stage, a little 'bump' will appear.

Special Considerations

Inverse Magnification

Inverse magnification occurs when the magnify effect is used with percentages less than 100%, sort of like looking through the back end of a telescope. Inverse magnification can cause some interesting side effects.

1. We don't recommend doing inverse magnification on sprites using the normal draw method. As the sprite shrinks away, you will see the white or black square of empty data that contains the sprite. This tends to not be terribly attractive.
2. On the other hand, we highly recommend using inverse magnification on sprites with the reveal or apply methods. In apply method, the sprite can become a miniature version of the stage, complete with moving sprites, etc..
3. When using inverse magnification and the apply method if there are any MIAW's expect to see very weird things go on as Director juggles the offscreen buffers.

Behaviors

Magnify

This simple behavior let's the user set up a sprite into static mode and apply some percentage of magnification, and turn on or off the interpolation.

Custom Functions

SetMagPoint

SetMagPoint(sprite x, #magnify, x, y)

This function sets the magnification point to be the center of magnification for the sprite. The x and y terms are in coordinates relative to the sprite's center, not global stage coordinates.

InterpolateNow

InterpolateNow(sprite x, #magnify)

This function causes the sprite to immediately redraw interpolated, but does not change the flag for the sprites overall interpolation setting.

SetInterpolation

SetInterpolation(sprite x, #magnify>, interpolation)

Magnify Reference

Effect Symbol: #magnify

Properties	Type	Value	Default	Exclusive
#animMode	symbol	#static, #range, #infinite, #pendulum	#static	
#numFrames	integer	1+	-	required in infinite, range and pendulum
#interpolation	integer	0, 1, 2	1	-
#percentage	integer	1+, 100 = no magnify	100	
#startPercentage	integer	1+, 100 = no magnify	current	valid in pendulum, infinite and range only
#endPercentage	integer	1+, 100 = no magnify	#percentage	
#magPoint	point	relative to sprite center	sprite center	
#magPointX	integer	relative to sprite center		
#magPointY	integer	relative to sprite center		
#easeIn	integer	frames	0	valid in pendulum, infinite and range only
#easeOut	integer	frames	0	valid in pendulum, infinite and range only

Custom Functions

SetMagPoint

SetMagPoint(sprite x, < position or #magnify >, x, y)

This function sets the magnification point to be the center of magnification for the sprite. The x and y terms are in coordinates relative to the sprite's center, not global stage coordinates.

InterpolateNow

InterpolateNow(sprite x, < position or #magnify >)

This function causes the sprite to immediately redraw interpolated, but does not change the flag for the sprites overall interpolation setting.

SetInterpolation

SetInterpolation(sprite x, < position or #magnify >, interpolation)

Arguments

percentage
The percentage is the central parameter to the magnify effect. Magnify by what percentage. Values greater than 100 will result in 'enlargement' or magnification, and values less than 100 will result in 'shrinking' or inverse magnification (like looking through a telescope backwards). Values can be anything, and don't have to be even multiples. For example magnifying by 117 percent is perfectly allowable and looks great.

magPoint
This is a point type parameter that tells the sprite where to center the magnification. This parameter is relative to the center of the sprite, so point(0, 0) is the very center of the sprite. In the apply and reveal drawing methods the magnification point can be outside the sprite itself. In fact, in apply mode the magnification point can be anywhere on screen (but remember the point is still relative to the sprite).

magPointX, magPointY
If you'd rather not use a point type, you can pass in these individual parameters. (See magPoint above) You are not required to use both, if you simply want to adjust the magnification point in one direction..

Seurat Basics

Effect Type: Complex

Georges Seurat (1859-1891) was a famous artist who pioneered the painstaking and arduous pointillist painting technique. Now you can imitate his genius with a cheap hackneyed special effect, but this one animates! This effect is one of the simplest to use, and yet very impressive. The speed, dot size, and intensity of the effect can all be controlled.

Techniques

Wavy Liquid

Take two sprites that are the same AlphaMania castmember, put one exactly on top of the other and apply the Seurat effect to it. You'll now see a 'wavy liquid' type distortion going on. If the castmember has a lot of smooth gradients in it, then the effect will be more apparent.

Special Considerations

No Animation Modes

The Seurat effect does not have an animMode parameter and does not fit the static/range/pendulum/infinite model of the other effects. Instead, Seurat simply has a speed parameter. Setting this to 0 will 'freeze' the effect.

No Semi-Transparency

The Seurat effect blows right through transparency settings, so it may not be the best effect to put on glass buttons.

Intensity Directly Impacts Performance

A high intensity setting may slow down this effect, and a low one may speed it up.

Seurat Parameters

Intensity

The intensity parameter controls the number of dots. Higher values will produce a denser packing of dots, but may slow the effect down

Speed

This is the relative speed of the dots as they bounce around and travel.

Radius

This is the dot size. The dots may be 1, 2, or 3 pixels wide.

Behaviors

Seurat

This behavior quickly and completely maps all the possible settings for the Seurat effect onto a sprite, so it's an easy and powerful behavior to use.

Seurat Reference

Symbol: #seurat

Properties	Type	Value	Default	Exclusive
#speed	integer	0+	7	
#radius	integer	1-3	3	
#intensity	integer	1 - 16	4	

Speed
This is the relative speed of the dots as they bounce around and travel.

Radius
This is the dot size. The dots may be 1, 2, or 3 pixels wide.

Intensity
The intensity parameter controls the number of dots. Higher values will produce a denser packing of dots, but may slow the effect down

Snow Basics

Effect Type: Simple

This effect makes your sprites into a snow globe or a static filled TV. And if you want to show snow on Mars, you'll be glad to know you can change the color of the snow to red or even have it randomly change.

Techniques

Film Grain

By using a low strength white or black snow you can subtly distort an image, as if there is a constant film grain effect going on. Remember, AlphaMania castmembers can be put into the apply method, so this effect can be live rendered onto QuickTime movies, etc.

Television Static

Use the chroma value of 2 for random color. Raise the strength, intensity, and speed parameters to taste. Not to be too redundant, but don't forget that AlphaMania castmembers can be put into the apply method, so this effect can be live rendered onto QuickTime movies, etc.

Special Considerations

No Animation Modes

The Snow effect does not have an animMode parameter and does not fit the static/range/pendulum/infinite model of the other effects. Instead, Snow simply has a speed parameter. Setting this to 0 will 'freeze' the effect.

Intensity Directly Impacts Performance

A high intensity setting may slow down this effect, and a low one may speed it up.

Using The Set FX Movie with Snow

Because of the late addition of some parameters to the snow effect, the 'chroma' parameter that lets you adjust the color of the snow is not included in the Set FX movie. A future version will correct this oversight. In the interim, if you want easy access to this feature, use the included behavior.

Snow Parameters

Chroma
The 'chroma' parameter to the snow effect is, unfortunately, a little obtuse. But here are the possible values:

0	white/lightening
1	black/darkening
2	random color
3	blue-green
4	magenta
5	blue
6	yellow
7	green
8	red

The 'random color' most resembles the multi-colored TV static. When using the random color chroma setting, you may want to increase the 'strength' parameter.

Strength
The strength parameter controls how much the snow 'flakes' displace the original color. At a low setting, the snow effect can appear to be a subtle distortion or film grain, at the high setting the picture can nearly be obliterated by the flakes.

Intensity
The intensity parameter controls the number of dots. Higher values will produce a denser packing of dots, but may slow the effect down

Speed
This is the relative speed of the dots as they bounce around and travel.

Behaviors

Snow
This simple but powerful behavior let's you access all the parameters of the snow effect.

Snow Reference

Symbol: #snow

Properties	Type	Value	Default	Exclusive
#speed	integer	0+	7	
#strength	integer	0-6	4	
#intensity	integer	1 - 16	4	
#chroma	integer	0, 1, 2, 3-8	0	

Speed

This is the relative speed of the dots as they bounce around and travel.

Strength

The strength parameter controls how much the snow 'flakes' displace the original color. At a low setting, the snow effect can appear to be a subtle distortion or film grain, at the high setting the picture can nearly be obliterated by the flakes.

Intensity

The intensity parameter controls the number of dots. Higher values will produce a denser packing of dots, but may slow the effect down

Chroma

The 'chroma' parameter to the snow effect is, unfortunately, a little obtuse. But here are the possible values:

0	white/lightening
1	black/darkening
2	random color
3	blue-green
4	magenta
5	blue
6	yellow
7	green
8	red

The 'random color' most resembles the multi-colored TV static. When using the random color chroma setting, you may want to increase the 'strength' parameter.

Mosaic Basics

Effect Type: Simple

This effect is handy for disguising peoples identities, creating futuristic video effects, etc. It's an animated version of the mosaic effect seen on commercial video SEG's, or Photoshop's mosaic filter. The horizontal and vertical 'resolution' may be controlled independently and animated over time.

Techniques

Animating the Colors Only

You may find that you want to use the mosaic effect, but really have no need to change the resolution of the effect over time. Thus, you might, for that reason want to use the static mode. But if you want to add some excitement to the effect, try instead using the infinite or pendulum modes, setting the start and end points of the animation to be the SAME, and turning on the 'changeColorFlag'. Now, you'll have a mosaic effect that isn't changing resolution, but is flipping colors in the boxes! This makes it look more like 'real video'.

Recommendations

Use Reveal or Apply Drawing Methods

While mosaic will work fine in the normal drawing method, you may occasionally see white or black on the edges of your graphic. This is a side effect of the normal method and this type of effect.

Using The Set FX Movie with Mosaic

The Set FX move, to simplify any confusion associated with this effect, does not let you control the x and y resolutions separately.

Behaviors

Mosaic

This behavior let's you set the x and y resolution of this effect and puts it into static mode.

Mosaic Reference

Effect Symbol: #mosaic

Properties	Type	Value	Default	Exclusive
#animMode	symbol	#static, #range, #infinite, #pendulum	#static	
#numFrames	integer	1+	-	
#xResolution	integer	1+, resolution in pixels	1	
#yResolution	integer	1+	1	
#startXRes	integer	1+	current	valid in pendulum, infinite and range only
#startYRes	integer	1+	current	valid in pendulum, infinite and range only
#endXRes	integer	1+	#xResolution	
#endYRes	integer	1+	#yResolution	
#changeColorFlag	integer	true or false	false	valid in pendulum, infinite and range only

Arguments

xResolution, yResolution
The x and y resolution parameters set the height and width of each 'box' of the mosaic effect, measured in pixels.

changeColorFlag
When this flag is true, the sprite will choose a different source pixel in each 'box' (defined by the x and y resolutions) each time it draws. Note, if setting this flag true in the #range and #infinite modes, the 'numFrames' parameter is not needed.

LineSkip Basics

Effect Type: Complex

This effect let’s you skip lines or sets of lines when drawing the sprite. Useful for creating transitions that don’t stop other animations, futuristic video effects, a poor man’s blending (but FAST), or simply drawing sprites quicker. You can control the number of lines to alternately skip and draw, as well as the ‘offset’ in to create ‘rolling’ effects. All of these parameters can be animated.

Using The Set FX Movie with LineSkip

Because of the late addition of some parameters to the lineskip effect, the ‘offset’ parameter is not included in the Set FX movie. A future version will correct this oversight.

Behaviors

LineSkip

This simple behavior let’s you set up the number of lines to skip and the number of lines to draw. It puts the sprite into the static animation mode.

Line Skip Reference

Effect Symbol: #lineskip

Properties	Type	Value	Default	Exclusive
#animMode	symbol	#static, #range, #infinite, #pendulum	#static	
#numFrames	integer	1+	-	required in pendulum, infinite, and range
#linesToSkip	integer	0+	0	
#linesToDraw	integer	1+	1	
#offset	integer	0+	0	number of lines to skip at top of sprite before drawing.
#startSkip	integer	0+	current	valid in pendulum, infinite and range only
#startDraw	integer	1+	current	valid in pendulum, infinite and range only
#startOffset	integer	0+	current	valid in pendulum, infinite and range only
#endSkip	integer	0+	#linesToSkip	
#endDraw	integer	1+	#linesToDraw	
#endOffset	integer	0+	#offset	

Arguments

linesToSkip
This is the number of lines or rows of the sprite to skip before drawing..

linesToDraw
This is the number of lines or rows of the sprite to draw before skipping.

offset
This is the number of lines to skip at the very top of the sprite before beginning to skip more lines. You can use this parameter to cause a 'rolling' of the lines through the sprite, or to interlace two different lineskipped sprites together.

Part 3: Guide to Effector Set II's Effects

Effector Set II contains five very powerful effects: Bevel, Blur, Drop Shadow, Ripple, and Roil. Particular attention has been given to these effects with respect to lighting, quality, and flexibility.

Like the last section, each effect has a Basic description followed by a drier reference section.

Bevel Basics

Effect Type: Simple

The bevel effect brings your AlphaMania sprites into the third dimension. This effect will bevel the edges of your castmembers, and control the position, strength, and color of the light source. Almost every feature of this effect can be animated, complete with `easeIn` and `easeOut`.

Special Considerations

Raised from the Surface or Depressed Below

The bevel effect can accept negative bevel values to make your sprite appear to be depressed below the surface of the stage, rather than extruded out above the stage. However, much of this illusion is dependent upon the lighting. You may find that the illusion of the bevel being extruded or depressed is best affected by positioning the light source above or below the sprite, and when using the point light source, positioning the light source inside the visible area of the sprite rather than outside will impact this illusion significantly.

Lighting

The key to cool bevels is in the lighting. To this end the bevel effect supports many different light settings:

`pointOrRay` - this flag determines what type of light source you use. A ray type light source is easier to use, and perhaps more conventional, but the point light source is VERY dramatic and the rewards reaped with its usage are high.

`relativeToSprite` - this flag determines whether or not the light source is positioned in stage-based coordinates, or simply by the distance away from the upper left of the sprite.

`lightLoc, lightLocX, lightLocY` - these parameters determine the position of the light source.

`strength, radius, red, green, blue` - these parameters determine the power and color of the light source. They all interact. For instance, if you want a narrow light with a real hot spot in the center, try bringing the radius down, but raising (equally) the red, green, and blue to values like 500. They also affect the specular highlighting of the beveled sprite.

Coloring

If using colored lighting you may get 'negative' coloring in the dark areas. To avoid this, raise the strength up to 128 (or higher).

Changing the Bevel Width over Time (Bevel Cropping)

The Bevel effect supports `startBevel` and `endBevel` arguments which will change the bevel over time, but the performance may not be acceptable. Instead we recommend using the bevel cropping args (`bevCrop, startBevCrop, endBevCrop`). The value of the bevel crop is the maximum bevel height, and by manipulating this value over time you can make any beveled graphic appear to raise up off the stage smoothly and with fast performance. Use the `easeIn` and `easeOut` arguments for added drama.

Curving the Bevel

Use the `curve` argument to bend the bevel out (puffy) or in (like a mesa). Fun!

Bevel Reference

Bevel Effect Argument Table

Effect Symbol: #bevel

Properties	Type	Value	Default	Exclusive
#animMode	symbol	#static, #range, #infinite, #pendulum	#static	
#numFrames	integer	1+	-	required in pendulum, infinite, and range
#radius	integer	any	100	NOT measured in pixels, relative to strength and rgb.
#startRadius				
#endRadius				
#strength		any	0	
#startStrength				
#endStrength				
#lightLoc	point		-20, -20	
#lightLocX	integer			
#lightLocY	integer			
#lightRelToSprite	integer	0, 1	1	0 = relative to stage 1 = relative to sprite
#bevCrop	integer	0-255	255	
#startBevCrop	integer			
#endBevCrop	integer			
#bevel	integer	any	5	can't be used w/ bevelRect
#bevelRect	rect	any	0,0,0,0	can't be used w/ bevel; recommend setting #bevel to 0.
#startBevel	integer			USE BEV CROP INSTEAD
#endBevel	integer			USE BEV CROP INSTEAD
#pointOrRay	integer	0, 1	0	0 = point light source, 1 = ray light source.
#curve	integer	-8 to 8	0	negative values are convex, positive concave.
#red	integer	any	255	
#green	integer	any	255	
#blue	integer	any	255	
#startRed	integer	any		
#startGreen	integer	any		
#startBlue	integer	any		
#endRed	integer	any		

#endGreen	integer	any	
#endBlue	integer	any	
#easeIn	integer		0
#easeOut	integer		0

Arguments:

bevel
This argument determines the width (in pixels) of the bevel. Negative arguments can be used to cause a 'depressed' rather than 'extruded' illusion. Note that much of this illusion depends upon light position.

bevelRect
This argument can be used to override the `bevel` argument and let you specify the amount of beveling on each side of the sprite. The left, top, right, and bottom values of the `bevelRect` are used as the amount of beveling to perform on the left, top, right, and bottom of the sprite. This argument is very useful for achieving an 'angled' extrusion (like `rect(0, 0, 10, 10)`) or an orthographic extrusion (such as `rect(6, 1, 6, 1)`). Wildly disparate values may cause the beveling to look poor or unacceptable (such as `rect (-1, 16, -28, 4)`). Set the `bevel` parameter to 0 when using the `bevelRect`.

radius
This argument affects the overall radius of the light source. However, the radius of the light source is not measured in pixels and will be inter-dependent on the `strength, red, green`, and `blue` settings.

strength
Overall strength of the light source. Interacts with the `radius, red, green`, and `blue` settings. When using colored light sources, recommend setting the `strength` to 128 to avoid reverse coloring of the dark areas.

pointOrRay
Determines whether the light source is a point-based light or a ray type light. When using point-based light the light has a very specific location where it is bright and then generally falls away. When using the ray-based light the light has a general location that is applied equally to the whole sprite. Arguments like `radius` and light location, and `lightRelToSprite` are relevant in both lighting models.

lightRelToSprite
This argument determines whether the light location parameters are in stage based or sprite based coordinates. If in sprite based coordinates, then no matter where the sprite moves it's lighting will be constant and unchanging (assuming you aren't actually changing the light location parameters), but when using stage based coordinates, the sprite can move 'into' and 'out of' the light (even in the ray source model).

bevCrop
The maximum bevel height. Useful for making the bevel appear to change over time. Gives better performance than the `startBevel` and `endBevel` args. Example:
`Bevel(sprite x, [bevel:8, animMode:#range, numFrames:20, startBevCrop:0, endBevCrop:255, easeIn:7])` This line will make the sprite appear to rise off the stage.

curve
Let's you curve the bevel. Subtle but worthwhile.

red, green, blue

Affects the coloring of the light source, but also interacts with the overall power of the light (along with `strength` and `radius`). These values are not limited to 0 to 255, but can be negative, and/or large (such as 500).

lightLoc

lightLocX, lightLocY

Specifies the location of the light source. If `lightRelToSprite` is 0, then these coordinates should be stage based, but if it is 1, then these coordinates should be relative to the sprite.

DropShadow Basics

Effect Type: Complex

Finally, animated, colorized, feathered, and fully controllable drop shadows! This effect even has a #relativeToPoint animation mode to let you specify a light source location!

Special Considerations

DropShadow Should Be Last Effect

Under most circumstances if you are using multiple effects, be sure to add dropshadow last. This is especially true if using it in conjunction with an effect like rotate.

Working With a Light Source

In the `#relativeToPoint` animation mode you can use the light location parameters to define the location of the light source. In this case the `xOffset` and `yOffset` parameters will be used determine the overall length and offset of the shadow, but not the actual x and y offsets. The proximity of the light source to the sprite will not cause foreshortening or lengthening of the drop shadow.

Custom Functions

PickColor

```
PickColor( sprite x, #dropshadow)
PickColor( sprite x,  #dropshadow, initialRed, initialGreen, initialBlue)
```

PickColor puts up the color picker dialog and returns a comma-delimited string of three numbers which are the red, green, and blue values chosen by the user. If the string comes back empty, the user cancelled the dialog. Initial red, green, and blue values can optionally be passed into this function.

This function doesn't modify the drop shadow in any way, it's simply a way to bring up the color picker dialog for use in the SetFX movie. Use it in your own movies!

DropShadow Reference

Effect Symbol: #dropshadow

Properties	**Type**	**Value**	**Default**	**Exclusive**
#animMode	symbol	#static, #range, #infinite, #pendulum, #relativeToPoint	#static	
#numFrames	integer	1+	-	required in pendulum, infinite, and range
#xOffset	integer		4	
#yOffset	integer		4	
#transparency	integer	0-255	200	0 - invisible, 255 - opaque
#red	integer	0-255	0	color of drop shadow
#green	integer	0-255	0	
#blue	integer	0-255	0	
#feather	integer	0 - 4	0	
#startXOffset	integer			
#startYOffset	integer			
#endXOffset	integer			
#endYOffset	integer			
#startTrans	integer	0-255		
#endTrans	integer	0-255		
#startRed	integer	0-255		
#startGreen	integer	0-255		
#startBlue	integer	0-255		
#endRed	integer	0-255		
#endGreen	integer	0-255		
#endBlue	integer	0-255		
#startFeather	integer	0 - 4		
#endFeather	integer	0 - 4		
#lightLoc	point			#relativeToPoint mode only
#lightLocX	integer			
#lightLocY	integer			

Arguments

xOffset, yOffset
The shadow offset. In `#relativeToPoint` mode they both determine the starting length of the shadow.

feather
The feathering of the shadow. Valid ranges are 0 - 4. Technically, higher values can be used, but expect quality to deteriorate significantly.

lightLoc
lightLocX, lightLocY
The position of the light source, in stage-based coordinates. The light position will affect the orientation of the shadow only, not it's overall length (which are always determined by the x and y offset parameters).

red, green, blue
The shadow color. The default is black (0, 0, 0).

Custom Functions

PickColor

```
PickColor( sprite x, #dropshadow)
PickColor( sprite x,  #dropshadow, initialRed, initialGreen, initialBlue)
```

PickColor puts up the color picker dialog and returns a comma delimited string of three numbers which are the red, green, and blue values chosen by the user. If the string comes back empty, the user cancelled the dialog. Initial red, green, and blue values can optionally be passed into this function.

This function doesn't modify the drop shadow in any way, it's simply a way to bring up the color picker dialog. Use it in your own movies!

Blur Basics

Effect Type: Complex

The blur effect defocuses a graphic. This is a very computation-intensive effect and can therefore be slow if applied to a large graphic. Traditionally the more you blur an object, the longer it takes. Our blur doesn't have that drawback, making it very useful in an interactive and animated context. This feature comes at the expense of quality, so blur is controlled by two properties that allow the author to precisely control speed vs. quality.

The blurAmount property determines how much a graphic will be blurred. You can decrease and increase its value without altering the amount of time the blur operation will take.

The blurResolution property determines the quality of the blur. The higher the resolution the longer it will take to perform the blur.

If the resolution of the blur stays the same, the amount of time the blur takes for a given castmember or sprite will remain constant even if the blurAmount changes from its minimum to its maximum.

Blur Reference

Effect Symbol: #blur

Properties	Type	Value	Default	Exclusive
#animMode	symbol	#static, #range, #pendulum	#static	
#numFrames	integer	1+	-	required in pendulum and range
#blurAmount		0-10	1	
#blurResolution		1-5	1	
#startBlur				
#endBlur				

### *blurAmount*

This is the amount of blurring to apply to the graphic. To facilitate animation, any blurAmount always takes the same amount of time to apply the effect to a given graphic at a given resolution. For this reason the larger the blur, the lower the quality of the blur. To increase quality, increase the blurResolution.

### *blurResolution*

This determines how high the quality of the blur is. Larger numbers cause the blur to occur more slowly.

Roil Basics

Effect Type: Simple

roil v.t. to make (water, etc.) unclear by stirring

Roil is one of the more psychedelic effects. It causes waves of light or power to flow across the surface of a graphic. Think of light at the bottom of a swimming pool. Perhaps a haiku will illuminate:

In the mind of the author
Wind begets stillness…
Roiling of a castmember

Roil can be controlled by the properties of speed and stretch. Speed determines how fast the waves of imagination travel over the graphic. Stretch determines how much the pixels will be pulled by the waves.

Only playing with this effect on a variety of graphics can truly impart an understanding of its true nature! Go now, Grasshopper, and roil away!

Roil Reference

Effect Symbol: #roil

Properties	Type	Value	Default	Exclusive
#animMode	symbol	#infinite only	#infinite	
#speed	integer	1+	7	
#stretch	integer	0-8	6	

speed

This is the relative speed of the effect. 7 is the default.

stretch

This determines how much the pixels will be pulled by the waves. 0 is no motion, and the maximum stretch value is 8.

Ripple Basics

Effect Type: Simple

The queen of ostentatious special effects, ripple makes your castmember or sprite appear to be magically painted on a water surface that can be manipulated in dramatic ways. Ripple has no user interface in the SetFX movie because its nature is entirely dynamic. It can be controlled with its behavior library or directly via lingo.

Each time a behavior or lingo invokes the ripple command it causes a circular disturbance in the surface of the water. This disturbance propagates across the water in various ways depending on the properties you have assigned the sprite or castmember. It is a bit like throwing a stone into a pond. Properties issued with the ripple command determine the size of the initial splash, where the splash occurs, the viscosity of the liquid, etc. Once a splash occurs, its ripples travel across the surface of the graphic interacting with the sides and with other ripples. The ripples gradually dissipate until the surface is still again.

Techniques

Pulsations

While ripple can be used to produce realistic water-like effects, you can also alter the arguments to produce some fascinating variations that are decidedly unnatural. If the castmember is circular and you set the ripple's initial radius to be the radius of the castmember, the entire graphic will appear to pulsate in and out of the screen.

Water Skier

Calling ripple continuously with a rapidly moving location causes the ripple to leave a wake like a water-skier. This is particularly fun if the mouse location determines the splash location.

Special Considerations

Speeding up the effect

Ripple is fairly processor-intensive. If you are rippling a large graphic, you may want to decrease the resolution of the ripple. Use the resolution argument to do this, and do it the first time you call the

Ripple Reference

Effect Symbol: #ripple

Properties	Type	Value	Default	Exclusive
Splash Specific				
#xLocation	integer	relative to sprite	-	
#yLocation	integer	relative to sprite	-	
#radius	integer	1+	2	
#height	integer	1 - 32000	400	
General				
#duration	integer	1+	4	
#smoothCount	integer	0+	0	
#stretch	integer	true or false	true	
#resolution	integer	1+	1	valid on first call only

Custom Functions:

Smooth

Smooth(sprite x, < position or #effectSymbol >)
This function causes a one-time 'smoothing' effect . The ripples are expanded and smoothed. This is way cool. Be careful of calling this function too much. Repeated calls may cause a 'snow' effect. We recommend not calling smooth any more frequently than once every 40 frames.

Splash Specific Arguments

The following arguments (xLocation, yLocation, radius, height) control the settings for an individual 'splash' and do not effect the overall settings of the rippling effect.

xLocation, yLocation

These two arguments place the initial 'splash' inside the sprite. These coordinates are local to the sprite, not the stage.

radius

This argument sets the radius of the initial splash. The built in default is 4 pixels. The radius may be clipped automatically if the value will put the initial splash outside the rectangle of the sprite

height

This argument set the height of the initial splash. The built in default is 500. Valid ranges are from 1 to 32000.

General Settings

The following arguments (duration, smoothCount, resolution, stretchFlag) control the settings of the overall ripple effect. All ripples currently underway and any subsequent ones will be affected by these settings. Be careful when using these settings, many of them can have unexpected side effects.

duration

This arguments determines how long a ripple will last before it fades away. The built in default is 4. Valid ranges are from 1 up. The exact duration of a ripple is a combination of it's initial height, radius and duration. The bigger the duration argument and the larger the initial height, then the longer the ripple will last. At low height values, the duration may not be able to increase the longevity of a ripple significantly. Be careful when manipulating the duration value. Setting this value too high may cause 'snow' rather than ripples.

smoothCount

This argument causes automatic smoothing to occur, the effect will automatically trigger it's own Smooth function. This argument determines the number of frames that should pass between calls to the 'Smooth' function. The built in default is -1 (No automatic smoothing). We recommend that you not use small values for the smoothCount, as this will probably result in 'snow'. Reasonable values are dependent on the general height of ripples currently underway and the duration setting, but we would recommend not calling smooth any more frequently than once every 40 frames.

resolution

This argument set the resolution of the splash map. This value can be used to speed up the ripple effect, but at the expense of smoothness and quality. The built in default is 1. Valid ranges are from 1 up. This argument can only be set in the call that initiates the effect. After that it will always be ignored.

stretchFlag

This argument turns on or off the 'stretching' of the ripples. Valid ranges are 0 (no stretch) or 1 (stretch).

The built in default is 1 (stretch). When stretch is on the bitmaps being rippled are actually mapped and stretched over the ripples. When stretch is off the ripples become simple lights and shadows cast onto the bitmap. Turning off the stretch may significantly speed up the ripple effect.

Part 4: AlphaMania 2 and Special Effects

This section discusses the many new features of AlphaMania 2 that facilitate the use of special effects and drawing methods. Subjects are broken down into techniques that don’t require lingo programming and techniques that do. Lingo programmers should read all sections, non-programmers can skip the lingo-oriented sections.

(While the only effect included with AlphaMania 2 is scale, it is still important for anyone who wishes to get full use out of scale and the drawing methods should familiarize themselves with this section.)

Drawing Methods

Introduction

An AlphaMania castmember contains RGB color information and transparency (alpha channel) information. (See “Anatomy of an Alpha Channel” if you are unsure about how AlphaMania’s transparency works.) Normally the color and transparency information are used together to draw an AlphaMania sprite on the stage. If you import an anti-aliased red circle from Photoshop and drag it to the stage you will see an anti-aliased red circle. This is known as the “normal” drawing method.

In AlphaMania 2.0 we have expanded your ability to use AlphaMania castmembers by adding two other drawing methods, “reveal” and “apply”. These two methods ignore the color information in your AlphaMania castmember, and use only the member’s alpha channel. The color information for each pixel comes from some other source.

In the case of the “reveal” drawing method the color information comes from a bitmap castmember that you specify..

In the case of the “apply” drawing method, the color information comes from whatever is underneath the AlphaMania sprite that is in “apply” mode.

The below image shows all three methods in action. The Apply method is shown in conjunction with the HSB effect to achieve the yellowing of a background area. Notice how all three modes draw using the alpha channel of the AlphaMania castmember. The source bitmap shown at the top of the figure shows the bitmap castmember the reveal method is using at the bottom.

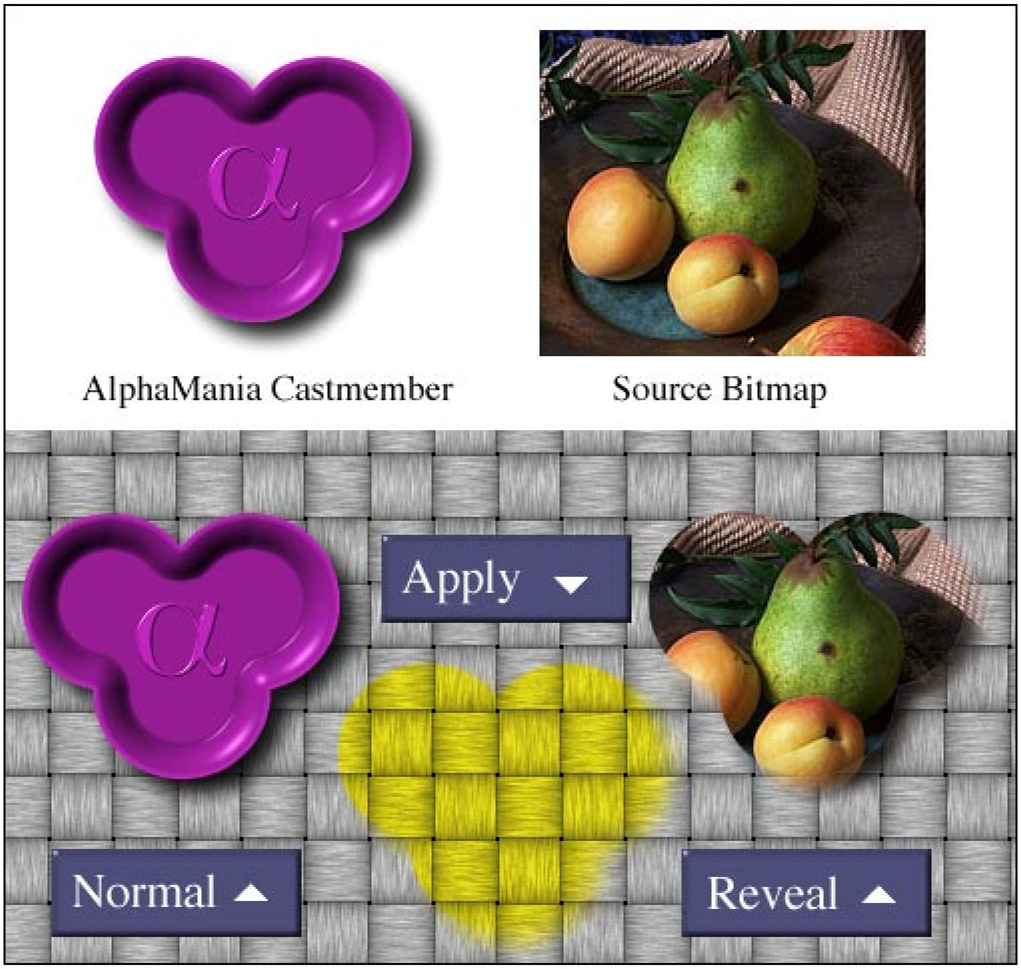


Figure 4. AlphaMania 2 drawing methods.

Reveal Drawing Method Concepts

The reveal method allows you to "cut a hole in the stage" in the shape of the alpha channel of your AlphaMania castmember. Through this hole is revealed part or all of a bitmap castmember that you specify. It is important to understand that reveal does not cut through one sprite to reveal another sprite in a lower channel. What it reveals is another castmember, not another sprite. It is irrelevant whether or not the revealed castmember is also on the stage. By varying the amount of transparency of pixels in the AlphaMania sprite, you can blend the revealed graphic with the stage for dramatic effects.

The SourceMember

You can specify what bitmap to reveal through an AlphaMania sprite using either lingo or the SetFX movie. This property is known as the sourceMember. It is always a castmember property, so all sprites created from a single AlphaMania castmember will reveal the same bitmap when the castmember is set to the reveal drawing method.

The SourcePosition
Usually the bitmaps that you reveal using the reveal drawing method are not exactly the same size and shape as the AlphaMania castmember. If the sourceMember is smaller than the AlphaMania castmember, you may want to control where it appears within the AlphaMania sprite's rectangle on stage. If the sourceMember is larger than the AlphaMania castmember you may want to control which portion of the larger image is revealed through the smaller AlphaMania sprite. Both of these tasks are accomplished by adjusting the sourcePosition property of the AlphaMania castmember using either the SetFX movie or lingo. The sourcePosition property is a point. If you don't specify a sourcePosition, the sourceMember will automatically be centered. It is important to note that it is possible to set the sourcePosition to a point that shifts it out of the visible area of the sprite.

If you don't explicitly set the sourcePosition, AlphaMania automatically defaults to auto-center the sourceMember within the sprite or the stage, depending on which the source position is relative to. (See below.) If you set a sourcePosition and then later would like the source bitmap to auto-center again, set the sourcePostion to point(65535, 65535).

The SourcePosition Relative to the Sprite or the Stage
Sometimes you will want different parts of the sourceMember revealed as the AlphaMania sprite moves around on the screen. For example you might have a large image of a human body on the stage along with a small AlphaMania sprite that reveals the skeleton of the body as the user drags it around. In this case the position of the revealed skeleton is constant relative to the stage, and doesn't change as the sprite moves. At other times you may wish the area of the revealed bitmap shown by the sprite to remain the same, even as the sprite moves around. In this case the position is relative to the sprite. The property that determines this is known as the sourceRelToSprite and is either true or false. It can be set with lingo or the SetFX movie. This property is also set at the castmember level and applies to all sprites created from the AlphaMania member. For an example of the use of this property, see the sample movie "basics.dir".

If you don't explicitly set this property, AlphaMania defaults to true, so that the source moves as the sprite does.

Reveal Drawing Method Considerations
If you have specified the reveal drawing method but have not specified a sourceMember or if AlphaMania cannot find the sourceMember, the sprite will appear as a grayscale mask until you either specify a valid sourceMember or change the effect. It is important to note that it is possible to set the sourcePosition to a point that shifts it out of the visible area of the sprite, making the sprite itself invisible.

There is no "background transparent" mode for the source bitmap. If the source bitmap is against white, then the white, too, will be revealed through the AlphaMania sprite.

Because reveal uses a normal bitmap as its source, you can use this method to manipulate any normal bitmap as if it were an AlphaMania castmember. You can make a feathered mask that will anti-alias a standard bitmap, for example. You can also use a mask to rotate a standard bitmap.

Apply Drawing Method Concepts

The apply drawing method uses an AlphaMania sprite's alpha channel as a mask to modify whatever area the sprite is covering on the stage. Exactly what modification occurs depends upon what effects have been added to the AlphaMania sprite or member. The amount of effect applied to each pixel on the stage is proportional to the opacity of the corresponding pixel in the alpha channel of your AlphaMania sprite, like adjustment layers in Photoshop 4. You could, for example, have an AlphaMania castmember that is a feathered mask of a person's hair. If you dragged the AlphaMania castmember over the corresponding part of the original image of the person you would be able to use the HSB effect to change the hair color to anything you wanted! This is especially efficient if you are using an 8-bit only AlphaMania castmember.

The Apply drawing method is only really useful in conjunction with one or more effects. Most effects come from Effector Set Xtras available separately from Media Lab, but the Scale functionality built into AlphaMania 2.0 is also implemented as an effect.

Apply Drawing Method Considerations

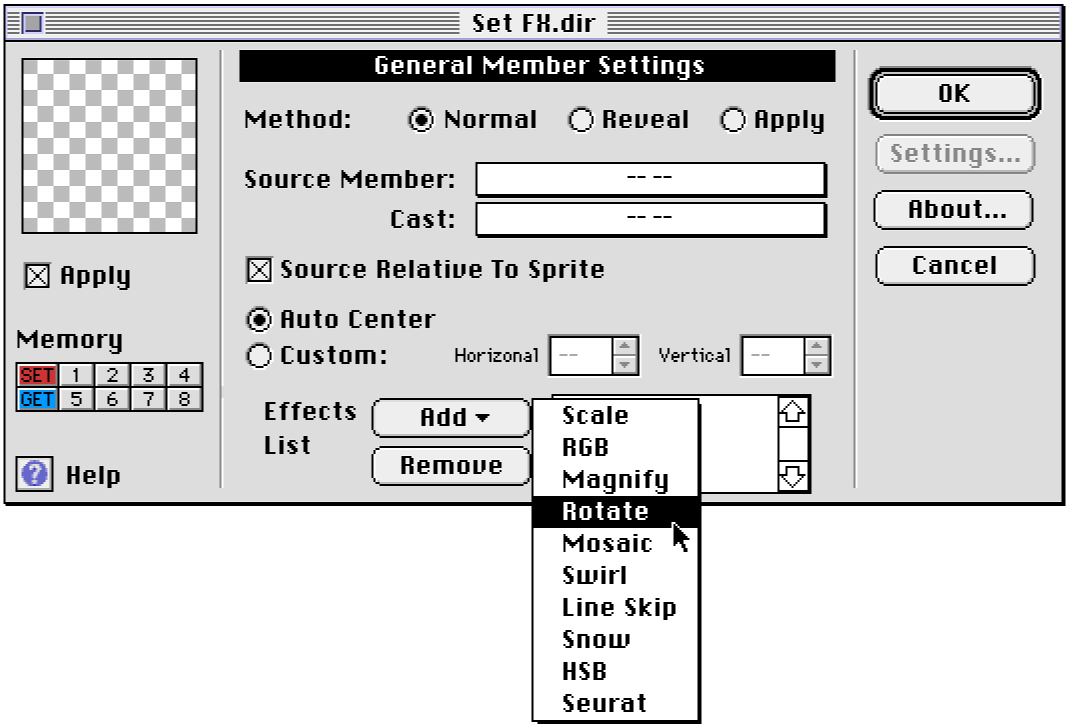
If you have specified the apply drawing method but have not applied an effect to the member or sprite, the sprite will be drawn as a grayscale until you either add an effect to the member or sprite or change the drawing method. It is possible to make the sprite seem to disappear by applying an effect to a background that the effect will not change. For example, if you specify the scale effect using the apply drawing method against a white background the sprite will seem to disappear, because scaling a solid color against itself results in no change to the image.

Because the Apply method affects everything behind it you can use it to treat any area of the stage, no matter what kinds of or how many sprites are stacked there, as if it were an AlphaMania castmember!

Using Drawing Methods with the SetFX movie

Setting the Draw Method

The SetFX movie can be used to set the Drawing method for a castmember. All sprites created from an AlphaMania castmember use the drawing method specified for that castmember. To manipulate the drawing method, select an AlphaMania castmember in the cast and then select “SetFX” from the Xtras menu. You will be presented with the following dialog (seen on next page):



☒ Apply

Method: ● Normal ○ Reveal ○ Apply

Source Member: -- --

Cast: -- --

☒ Source Relative To Sprite

● Auto Center

○ Custom: Horizonal -- Vertical --

Using the controls you see here you can select the drawing method to use and you can also set which bitmap castmember is to be revealed when the Reveal method is enabled. Bitmaps you'd like to reveal must be present in the current movie or a currently loaded cast file in order to appear in the list of source bitmaps. You can also set where that bitmap will be displayed, and whether that coordinate is relative to the stage or the sprite. (See "Reveal Method Concepts" above.)

Under Director 5, AlphaMania identifies the source member by cast file number and castmember number. Thus, if you move this member top a different cast position, AlphaMania will be unable to locate it and you will have to specify it again.

Under Director 6 AlphaMania identifies the source member by a global identification number and can find the member even if it has changed positions or cast files. (The member must be located in a cast file that is linked to the current movie.)

Using this window you may also set the sourcePosition and whether that position is relative to the sprite or the stage.

Using Drawing Methods With Lingo

There are lingo commands to control all aspects of drawing methods.

The drawing method itself is a lingo castmember property called the drawingMethod. The methods are provided as symbols, and so are preceded by '#' signs. To change a member's drawingMethod, simply set this property as you would any other member property. Example:

```
set the drawMethod of member 1 = #normal
set the drawMethod of member 1 = #apply
set the drawMethod of member 1 = #reveal
set curMethod = #apply
set the drawMethod of member 1 = curMethod
```

The other aspects of drawing methods are also lingo properties. The sourceMember is a castmember property that specifies the bitmap castmember to use with the Reveal drawing method. It can be set at any time, even if the reveal method is not the currently selected drawing method. Its value can be set to any legal reference to a bitmap castmember. Example:

```
set the sourceMember of member 1 = member 5 of castLib “English Graphics cast”
set the sourceMember of member 1= member “skeleton”
set curSource = member “innards”
set the sourceMember of member 1 = curSource
```

The sourcePosition is also a property which takes a point as its value. If you give it point(65535, 65535) as its value the position will automatically auto-center. Example:

```
set the sourcePosition of member 1 = point(50,50)
set the sourcePosition of member 1 = point(65535, 65535)  - -  auto-centers
```

You can determine whether the sourcePosition is relative to the sprite or the stage by setting the sourceRelToSprite property of the AlphaMania castmember to true or false. Example:

```
set the sourceRelToSprite of member 1 = true
```

8-Bit Only AlphaMania Castmembers

Now that you understand drawing methods, you will see the usefulness of 8-bit only (a.k.a. alpha-only) AlphaMania castmembers. They take up only a quarter of the space on disk and in memory of a standard AlphaMania castmember, yet they are fully as functional using either the Apply or Reveal drawing methods. The only drawback to 8-bit only AlphaMania castmembers is that because they contain no color data they cannot be used with the normal drawing method. Photoshop users can think of them as masks or even adjustment layers.

Importing 8-bit Only Members

8-bit only AlphaMania castmembers are created by clicking the “8-bit only” checkbox in the AlphaMania import dialog and then choosing a 32-bit file or Photoshop layer. The color data for the file or layer will be ignored and only the alpha channel will be imported.

8-bit Special Considerations

8-bit only castmembers cannot use the normal drawing method. 8-bit only castmembers are drawn in grayscales in reveal mode when no SourceMember is available, and in apply mode when no effect has been applied. Thumbnails for alpha-only members appear as grayscale masks.

If you would like to determine whether a member is 8-bit or not, use the “alphaOnly” property. It returns true or false. Example

```
if the alphaOnly of member 3 = false then set the drawMethod of member 3 = #normal
```

Using Effects

AlphaMania 2’s most exciting new feature is its ability to use dynamic special effects. Every effort has been made to make the Effector Set effects as easy to use as possible. If you want to jump right in simply select an AlphaMania castmember in your cast and choose the Set FX option from your Xtras menu. This is the quickest way to start using the effects right away. But be sure to come back to this point in the documentation because a firm understanding of the concepts discussed here will make you well armed to really push the effects to their maximum potential.

This section explains the concepts behind effects and how to use them with lingo and with the SetFX movie. The scale effect is the only effect that is included with AlphaMania 2. Other effects are available from Media Lab in the Effector Set Xtras. Many of the examples in this section make use of effects found in the Effector Set Xtras. If you do not own them but would like to explore some of the things they can do, free demos are available from Media Lab's web site.

Effect Basics

At the bottom of every Effector Set effect is an AlphaMania (2.0 or later) castmember or sprite. The Effector Set Xtras in a sense simply endow these castmembers and sprites with new abilities much as the Earth's yellow sun does for Superman. So both an Effector Set and the AlphaMania 2 Xtra must be present to use the effects. (Scale is the only effect included with AlphaMania 2.)

The effects themselves are similar to video effects or animated Photoshop filters with a very important difference: you can control the properties of the effects as the movie plays. You can colorize something, rotate it, etc. all in real time under your control or under the control of the end user of your product.

The simple process of using an effect is as follows:

1. Import an AlphaMania castmember with AlphaMania 2.0 and select it in the cast

2. Add an effect to the castmember using lingo or the provided SetFX movie

3. Adjust the properties of the effect to taste

Steps 2 and 3 can be repeated indefinitely as you add multiple effects with different properties to the same castmember. For example you could make a castmember rotate as it becomes more blue and ripples like water. How? Read on...

Common Effect Properties

While all effects are different from each other, they share many things in common. All effects can animate over time, for example. (See "Animation Modes" below for details.) For the effect to know what to do as time passes you must set starting and ending conditions, well as provide a number of frames for this change to occur across. Many effects can ease-in or ease-out over a number of frames you specify, depending on the animation mode. You can these properties using the SetFX movie in the Xtras menu, or you can set them directly with lingo. The documentation for each individual effect discusses the unique nature of its own special abilities, but in this documentation you will always see properties with names #startHue, #endHue, #startPercentage, etc.

Multiple Effects

Effects are like stops on an assembly line that exists between your original castmember graphic and how it will look when it is eventually drawn on the stage as a sprite. At the beginning of the assembly line is the source graphic as specified by the base AlphaMania castmember and the drawing method. Each new effect you add to a member or sprite creates a new stop on the assembly line. The graphic travels to each stop, is changed in some way, and then continues to the next stop until all of the effects have done their magic. The graphic is then drawn to the screen.

This means that the effects are cumulative and that changing the order of effects can alter the final result. Each effect manipulates the graphic that resulted from the previous effects. If the first effect makes the graphic more blue, then it is this blue graphic, not the original graphic that is sent to the next effect.

The first effect you add is the first effect applied, and so on. Effects can be removed at any time from any position in the effects order. If you wish to arbitrarily change the order of effects you must remove all of the effects and add them back from scratch in the order you desire.

Note: The more effects you add to a castmember or sprite, the longer your sprite will take to draw. While all of the effects have been highly optimized for speed, too many effects on a large enough castmember can slow even the fastest PowerPC or Pentium to a crawl.

Castmember Effects vs. Sprite Effects

Adding an effect to a castmember using the SetFX movie or lingo will cause that effect to operate on all sprites that are created from that castmember. This is similar to the width of a bitmap castmember. If you change the width of a bitmap member in the paint window, all sprites from this member will change. But if you change the width of a bitmap sprite, only that sprite is affected. When you add an effect to a sprite with lingo, only that sprite is affected.

Similarly, changes made to an AlphaMania castmember's properties with lingo or the SetFX movie are saved with the movie. Changes made to a sprite's properties with lingo are not. (The SetFX movie can't change sprite properties for this same reason.) If you type scale(member 1) into the message window, that information is saved when you save the movie. If you do the same to a sprite while the movie is playing, it will forget as soon as the movie is stopped or the sprite's span ends.

Simple vs. Complex Effects

There is an exception to the rule that effects are applied in the same order they are added. There are two types of effects, *simple* and *complex*. The only difference between them is that complex effects temporarily manipulate the alpha channel of the AlphaMania castmember in the process of applying the effect. This does not change the castmember, but it does mean that these effects must come at the end of the order of effects. AlphaMania manages this limitation for you, but if you notice that you added an effect after another effect and that when you check the effects list the order has reversed, it is because you added a complex effect before a simple effect.

Interpolation

Many effects support interpolation. This is programmer jargon for smoothing. Interpolation slows drawing down a bit, so AlphaMania supports three settings for you to choose from. The first turns interpolation off, which means that an effect will play at its fastest but look a bit worse. The second mode turns interpolation on and makes things look a lot better at the expense of speed. The third mode is a nice between the first two: interpolate when paused. Often when an effect is animating it doesn't need interpolation because it is moving too fast for the eye to detect any blockiness. The third mode turns interpolation off until the effect stops animating. The animation referred to here is effect animation, not score animation. See “Animation Modes” for details.

The Sprite Rectangle

Some effects, namely rotate, can change the size of your sprite rectangle when added. This is done on purpose because the effect will be changing the size of your graphic and is stopping the sprite from cropping. For example with rotate, any non-circular object will take up a varying amount of space as it rotates. When you add rotate to a castmember, it increases the sprite rectangle of sprites made from the castmember so that no matter how you rotate the graphic it will always fit into the new rectangle. (Adding the effect at the sprite level with lingo does not do this and can cause cropping. See the documentation of the Rotate effect for details.)

Effects And Behaviors

Director 6's new drag-and-drop behaviors are incredibly well suited to making complex interactions with effects very easy. We have included a large number of behaviors for you to use and learn from, so be sure to try them out. Behaviors allow complex interactivity with an effect as the movie plays. One of the included examples is a simple game of asteroids with rotating flying meteors and a user-controllable spaceship created entirely from only two behaviors and two bitmaps!

Effect Animation Modes

While all of the effects in Effector Set I are unique there are some common elements to most of them. One of the key ones to understand is the 'animation mode'. All of the effects can be animated in different fashions, and all share some of the four common animation modes: static, range, infinite, and pendulum.

There is a sample movie, 'Methods and Modes' or 'MethMode.dir', that let's user's play with drawing methods and animation modes. Playing with it is a great way to see the different animation modes in action.

Important: AlphaMania effect animation modes are independent of score animation. An effect will change over time or not only as you specify using lingo or the SetFX movie. Score animation that makes a sprite move across the stage over a certain number of frames does not affect the animation modes discussed in this section.

Static

In static mode an effect is simply applied at some defined setting and then never changes. There is no animation involved. As an example, think of the rotate effect - in static mode you might specify for it to rotate to 45 degrees. It will immediately redraw angled at 45 degrees and then stay that way.

Static mode is 'low power'. Effects that are in static mode do not eat up any processor time if they aren't moving or having to redraw.

Range

In range mode an effect is applied, changes over time, and then stops. The settings must therefore include a number of frames across which the effect occurs. The effect plays for that number of frames and then becomes static. As an example of this, think again of the rotate effect - in range mode you might specify for it to rotate to 45 degrees across 10 frames. It will then, over the next 10 frames, rotate to an angle of 45 degrees (from whatever angle it started) and then stop.

Once the effect destination is reached, the sprite will stop and behave as if it were in static mode. Most effects can also optionally provide a 'start' point too. So instead of rotate from whatever rotation the sprite starts at to 45 degrees, it could be made to rotate from 120 degrees to 45 degrees. Many effects support additional options like ease in and ease out frames for this mode.

Pendulum

In pendulum mode (sometimes known as Ping-Pong mode) the effect goes out to the specified endpoint and then returns, and then it starts all over again. In pendulum mode the effect runs repeatedly, never stopping. Using rotate as an example, in pendulum mode you might specify 45 degrees and 10 frames. The sprite will then, on the next ten frames, rotate to an angle of 45 degrees (from whatever angle it started), and then across the ten frames after than, swing back to the starting angle, and so on forever.

As with the range animation mode, most effects can optionally provide a start point for this mode, and many effects support relevant option like ease in and ease out frames.

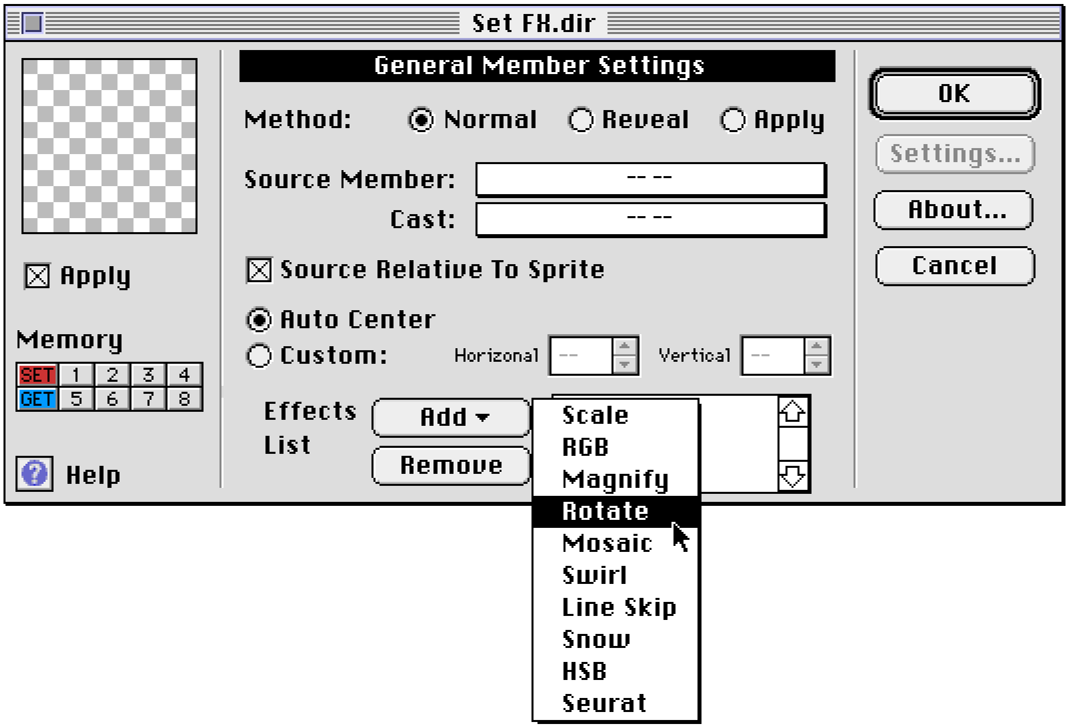
Infinite

In infinite mode the effect goes out to a specified endpoint and then the effect starts over. It does this repeatedly, never stopping. With most effects this will produce a rather jarring effect. Some effects provide a custom implementations of infinite mode. Rotate, for example, when in infinite mode, uses the number of frames is to determine how fast it performs a complete 360 degree rotation. Other effects (like HSB) may also provide custom implementations of this animation mode.

Like pendulum, many effects can optionally provide a start point for this mode, and many effects support relevant option like ease in and ease out frames.

Using Effects with the SetFX Movie

This section discusses the SetFX movie, a Director movie that is included with AlphaMania 2 and should be found in the Xtras menu. If it does not appear there, see “Troubleshooting.” This movie is the simplest way to add and modify castmember effects.



Method: ● Normal ○ Reveal ○ Apply

Source Member: -- --

Cast: -- --

☒ Apply

☒ Source Relative To Sprite

● Auto Center

○ Custom: Horizonal -- Vertical --

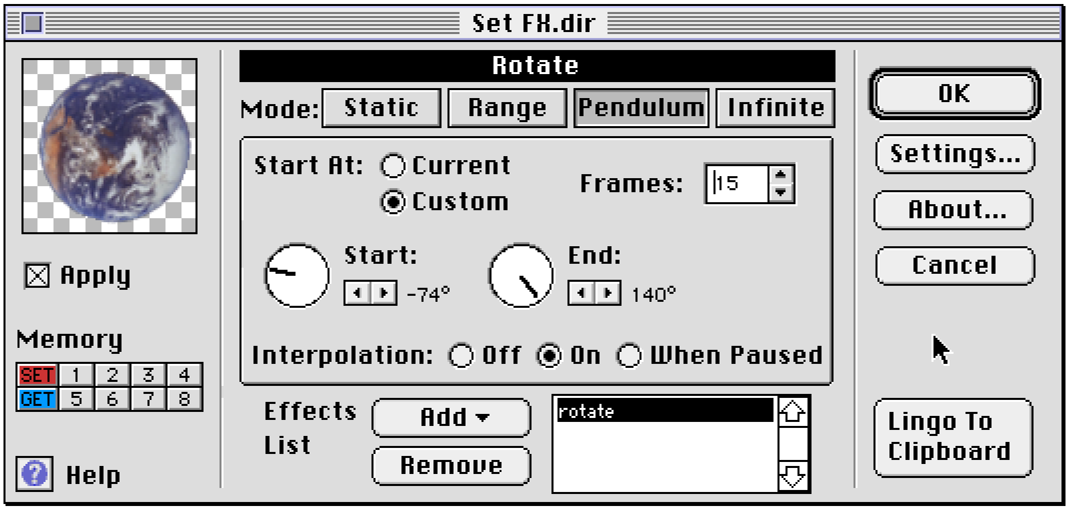
Figure 5. SetFX general settings.

Adding an Effect

To use the SetFX movie, select an AlphaMania castmember in the cast and select "SetFX" from the Xtras menu. You will be presented with the basic SetFX window that allows you to set and modify the drawing method for the castmember. (See "Drawing Methods")

Examine the bottom of this window. There are buttons labeled "Add" and "Remove." You will also see an empty list (assuming you haven't jumped ahead and added effects already.) Clicking on the Add Button will cause a popup menu to appear which contains all of the effects you currently have installed. (If you only have AlphaMania 2 and no Effector Set Xtras, only Scale will appear in this list.)

When you add an effect from this menu, the name of the effect will appear in this list, and the control panel for this effect will appear in the upper portion of the window. This panel is slightly different for each effect, but all effects share certain attributes.



Start At: ○ Current ● Custom

Frames: 15

Start: -74°

End: 140°

Interpolation: ○ Off ● On ○ When Paused

rotate

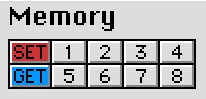
Figure 6. Rotate Settings

Editing an Effect
Adding an effect automatically takes you to the appropriate effect editing panel. To edit an effect you have previously added, double-click on the name of the effect in the scrolling list at the bottom of the SetFX window and you will be taken to the appropriate panel.

Once you have reached the panel for controlling the effect you will see buttons at the top of the window that allow you to determine the animation mode for the effect. Clicking on them can cause the control panel to change depending on the animation mode you choose. (See "Animation Modes") Within the effect control panel you can set starting and ending properties, number of frames, etc.

Effects Preview
A preview of the settings you choose appears in the preview window at the upper right of the SetFX window. The preview is animated and you can start it over from the beginning by clicking on it.

Underneath the preview is a checkbox labeled "Apply". This does not mean the apply drawing mode! It causes the current settings to be applied temporarily to any sprites on the stage created from the castmember you have currently selected so that you can preview the effect with your original graphic. The stage preview does not run the animation but can show you the start and end points as you adjust them.



The Memory Buttons
At the left of the window is are a number of small buttons labeled "Memory." These allow you to save and restore effects settings. You can save up to eight effects configurations. To save, click and hold the "Set" button and drag the cursor to the number you would like to assign to the current settings. To restore a configuration, click and hold the "Get" button and drag the cursor to the number you would like to restore. The configurations that are stored include all effects in the current effects list, not just the effect you are currently editing.

Moving Between Effects

If you have added several effects, to edit any one of them simply double-click on the name of the effect in the effects list at the bottom of the screen. This will take you to the appropriate control panel.

Removing Effects

Find the name of the effect to be removed in the effects list at the bottom of the window and select it. Click the "Remove" button to remove this effect from the effects list.

Other Controls

The "Settings" button takes you back to the panel which allows you to set the drawing method and associated settings for the castmember. The "Lingo to Clipboard" button is available only when editing a specific effect. It copies the lingo command required to create the current effect with the current settings to the clipboard and also puts it to the message window. This can be very helpful for both learning about the lingo used with effects, and as a time saver to actually create your lingo commands.

SetFX Special Considerations

The SetFX movie can only add, move, and manipulate effects at the castmember level. This means that changes you make to a castmember's effects will be alter all sprites generated from that member. To control effects at the sprite level you can use the behaviors included with the various Effector Set Xtras, or you can write your own lingo.

Using Effects with Lingo

While lingo novices will be able to accomplish quite a bit with simple commands, lingo experts will be able to do almost anything they imagine with the variety and power of the advanced lingo access to effects.

Adding Effects

Basic use of effects with lingo is surprisingly easy. The format for adding and manipulating effects is the same for all effects. Type the name of the effect you would like to create or change, followed in parentheses by the castmember or sprite you wish to modify and a property list containing the information required by the effect. Any arguments that are omitted are given default values or ignored. Most non-numeric arguments are passed in and out as Director symbols, and so are preceded by a "#" sign. Example:

```
scale(sprite 1, [#animMode : #follow] )
 – the above causes sprite 1 to scale into the sprites rectangle

rotate(member 4, [ #degrees:45, #interpolation: false ] )
-- the above rotates by 45 degrees all sprites created from member 4
```

The reference manual for the Effector Set Xtra containing the effect you wish to control lists and describes all possible arguments to that effect. It is important to note that, depending on the effect and the mode, certain arguments cannot be omitted. If you issue an effect command to a castmember or sprite and nothing happens, you probably forgot a required argument. The reference manual entry for an effect will also list what arguments are required for each animation mode, etc. When in doubt, use the SetFX movie "Lingo to Clipboard" command and compare the lingo it produces to your own.

Built-In Effect Help (DescribeEffect)

Within Director you can get a description of the various arguments and animation modes supported by an effect using the describeEffect command. This command must be called with any AlphaMania castmember and the effect you want to know about as arguments. No effect needs to be applied to the castmember you use this command with. The description is returned in the result and may be put to the message window. Example:

```
put describeEffect(member 4, #rotate)
```

Checking the Parameters of an Effect (GetEffectArgs)

Once you have applied an effect, you may wish to check on the arguments you sent to that effect as parameters. The lingo command GetEffectArgs accomplishes this. To use it, pass in an AlphaMania castmember and the name of an effect that has been added to it. Like DescribeEffect it returns its information in the result, which can be put to the message window. Example:

```
rotate(member 1, [#degrees:45, #interpolation: false])
put GetEffectArgs(member 1)
-- [#degrees: 45, #interpolation: 0 ]
```

Modifying an Effect After Adding It

Modifying an effect you have already added to a castmember is simple. AlphaMania automatically keeps track of effects that have been added to a castmember, so when you issue a second command using the same effect, AlphaMania changes the effect you have previously added, rather than creating a new effect. (To create multiple versions of the same effect see the 'Naming Effects' section.) Example:

```
rotate ( member 1, [#degrees:45])
put GetEffectArgs(member 1)
-- [#degrees: 45]
rotate [ member 1, [#degrees: 90])
put GetEffectArgs(member 1)
-- [#degrees: 90 ]
```

Managing Multiple Effects

As the above examples demonstrated, you add an effect to a sprite or member by issuing a command corresponding to the name of the effect and taking the sprite or member as an argument along with optional parameters supplied in a list.

Each time you do this with a different kind of effect, the effect you specify is added to a list of effects that will be performed each time the sprite is drawn.

Checking the Effects List (GetEffectList)

Once you have added some effects to a sprite or castmember, you can call GetEffectList() on that castmember or sprite to see which effects have been added and in what order. This command returns a lingo list. Example:

```
rgb(member 1, [redShift:50, blueShift:25])
put GetEffectList(member 1)
-- [#rgb : #rgb]
rotate(member 1, [ #animMode : #infinite,  #numFrames : 15])
put GetEffectList(member 1)
-- [ #rgb : #rgb , #rotate : #rotate ]
```

Naming Effects

Notice that the first call returned [#rgb : #rgb]. This does not mean that there are two rgb effects applied to this castmember, it means that there is one rgb effect that you have not given a name to, so it took the default name “rgb”. You name an effect by specifying the name when you add the effect. Example:

```
rgb ( member 1, #blueDude, [ blueShift : 50 ] )
put GetEffectList(member 1)
-- [ #blueDude : #rgb ]
```

Now you can refer to the effect by name if you prefer. Example:

```
rgb ( member 1, #colorDude, [ blueShift : 50 ] )
put GetEffectList(member 1)
-- [ #colorDude : #rgb ]
put GetEffectArgs( member 1, #colorDude)
-- [ #blueShift : 50 ]
blueDude(member 1, [ redShift : 20] )
put GetEffectArgs( member 1, #colorDude)
-- [ #blueShift : 50, redShift : 20 ]
```

This allows you to add multiple versions of the same effect to s sprite and then refer to them by name.

An important thing to be aware of is that when you add an effect to a sprite that effect is added to the sprite's effect list which already contains any effects which have been added to the castmember. The following example assumes that sprite 1 is created from castmember 1:

```
rgb(member 1, [blueShift:50] )
put GetEffectList(member 1)
-- [#rgb : #rgb]
 rotate( sprite 1, [#animMode : #infinite, #numFrames : 15] )
put GetEffectList(sprite 1)
-- [#rgb : #rgb, #rotate : #rotate]
-- the sprite now has two effects in its list
put GetEffectList(member 1)
-- [#rgb : #rgb]
-- the member still has just one effect
```

Even though the GetEffectList command returns a lingo list, modify this list directly with the standard lingo list commands does not affect the effects themselves. It is simply for your information.

Removing Effects (RemoveEffect, RemoveAllEffects)

If you need to remove an effect from the effect list for a sprite or castmember, use the RemoveEffect command. It takes the castmember or sprite as the first argument and the name of the effect as the second argument. Example:

```
rgb(member 1,[blueShift:50])
put GetEffectList(member 1)
-- [ #rgb ]
rotate(member 1, [#animMode  :#infinite,  #numFrames:15])
put GetEffectList(member 1)
-- [ #rgb : #rgb, #rotate : #rotate ]
removeEffect( member 1, #rotate )
put GetEffectList(member 1)
-- [ #rgb : #rgb ]
```

If you want to remove all of the effects from a sprite or castmember, use the RemoveAllEffects command. Example:

```
removeAllEffects( member 1 )
```

Manipulating Active Effects (PauseEffect, ContinueEffect, ResetEffect, PauseAllEffects, ContinueAllEffects, ResetAllEffects)

Once an effect is active on a sprite, there are a number of manipulations that can be done. These manipulations must be performed at the sprite level, and cannot be done at the cast level. The three built in manipulations that work with all effects are pausing, continuing, and resetting:

```
ResetEffect(sprite x, #effectSymbol)
ResetAllEffects(sprite x)
```

Typically, resetting an effect will zero or normalize any parameters and return the sprite to it's normal state, but the effect is still present and can be accessed and manipulated.

```
PauseEffect(sprite x, #effectSymbol)
PauseAllEffects(sprite x)
```

Effects that are running in the #static animation mode will be unaffected by this, but animating effects will stop animating until told to continue.

```
ContinueEffect(sprite x, #effectSymbol)
ContinueAllEffects(sprite x)
```

Effects that are paused will resume their animation.

Custom Effect Functions

Many effects provide custom functions to aid lingo programmers in controlling the effects' unique special capabilities. Any effect that interpolates will support two custom functions, `InterpolateNow` and `SetInterpolation`, which allow you a more convenient and precise level of control over interpolation. The rotate effect in particular supports a wealth of custom functions which make common rotation programming tasks much easier. The documentation for each effect describes the custom functions unique to that effect.

Alternative Calls to Effects

Most effect functions are typically called `FunctionName(sprite x, #effectSymbol, additionalArgs)`.
For example: `SetInterpolation(sprite 4, #rotate, 1).`
But if desired, the `#effectSymbol` can be replaced by either the effects custom name (if using a named effect), or number. An effects number is determined by it's order of performance on the sprite, which will match the order listed if `GetEffectList` were called on that sprite.
Examples:

`SetInterpolation(sprite 4, 2, 1)` would call `SetInterpolation` on the second effect of the sprite.

`SetInterpolation(sprite 4, #bob, 1)` would call `SetInterpolation` on whatever effect is named #bob.

Part 5: Register, Purchase, Distribute, Contact Info

How to Register an Effector Set

Effector Sets I & II are distributed as unlockable demos, which can be unlocked with a code number which you receive at time of purchase. The demo versions of Effector Sets I & II deface the AlphaMania sprites with effects applied by drawing a box around them. See Where To Find Effector Sets below for common locations to find up to date versions. Download the demo to try it out and then call in to order.

To purchase the produce and deactivate the defacing function, you will first need to determine the last ten digits of your Director serial number:

With that number handy, simply phone us (***1-800-282-5361***) to place your order for Effector Set I or Effector Set II and we'll give you your product unlocking code. Or you can order via fax or email by filling out the order form that comes with the downloadable versions, being sure to include the serial number digits.

Select the “Register Effector Set I” or “Register Effector Set II” movie from the Xtras menu and click “I Agree” after reading the license agreement. Enter the unlocking code into the “unlock” field and click “register.”! Keep that code safe somewhere in case you have to re-install Effector Set or Director for any reason.

IMPORTANT: Effector Set locks itself to the copy of Director in which it is installed. Moving it to a different copy of Director with a different serial number will cause it to revert to “demo” mode, reactivating the defacing function. So make sure you are getting your product ID from the copy of Director you expect to be using Effector Set with.

Purchasing

The easiest way to purchase the on-line version of Effector Set is with a credit card via the automated sales system on our Web page. (www.medialab.com) Your card will be processed immediately, and you will be issued an unlock code.

If you prefer to talk to a real person, or have any problem ordering via the Web site, call ***1-800-282-5361*** to purchase Effector Set using your Visa or MasterCard, at which time you will be given an unlock code. Alternatively, you may order by email or fax and receive the code number via electronic mail. If you are ordering Effector Set without having already downloaded and installed a demo version, we encourage you to have the last 10 digits of your Director 5 serial number ready when you call. This is not required, but will mean you won’t have to call back for an unlock code once you receive the product.

Ordering by Phone

In the U.S: *800.282.5361*
International: *+1.303.499.5411*

Orders can be taken Monday through Friday, 9 AM to 5 PM Mountain Standard Time.

Ordering by E-Mail

xtra-sales@medialab.com

There is an order form (ORDER.TXT) provided with the electronic download. Fill in all the information it requests, including your Product ID (see above), and e-mail back to xtra-sales@medialab.com

Ordering by Fax

+1.303.497.9454

There is a order form (ORDER.TXT) provided with the electronic download. Fill in all the information, including your Product ID (see above), and fax it back to us.

Distributing an Effector Set

There are no licensing fees for distributing Effector Set I or II with your products.

An Effector Set Xtra must be distributed with any of your projectors that contain AlphaMania castmembers that use effects in that Xtra. An Effector Set Xtra & AlphaMania 2 must reside in a folder named "Xtras" that appears in the same folder as your projector. See the chapter on Packaging Effector Set With a Projector for more detailed instructions.

Contacting Media Lab, Inc.

Media Lab, Inc.
31 S. 80th St.
Boulder, CO 80303 USA
Phone: 800.282.5361 to purchase PhotoCaster
+1.303.499.5411 all other calls
Fax +1.303.497.9454
Website: http://www.medialab.com/software/
Distribution Inquiries: sbruce@medialab.com
Press and Advertising: sbruce@medialab.com
email: xtra-help@medialab.com Technical support.
xtra-sales@medialab.com Place orders for PhotoCaster
xtra-bugs@medialab.com Report any bugs or request new features.

Part 6: Legal Stuff

Licensing

When you purchase Effector Set I, you are purchasing the license to use Effector Set I for authoring with a single copy of Director on a single machine. This fee also covers distribution of the Xtra with your Director and ShockWave projects. There is no fee associated with distributing Effector Set I for playback only.

Copyright Information

Effector Set I ©1996 As Is Software, Inc. Effector Set I is distributed by Media Lab, Inc.. Any distribution related inquiries should be emailed to sbruce@medialab.com.

Trademarks

Effector Set I and PhotoCaster are trademarks of Media, Lab, Inc. Adobe Photoshop is a trademark of Adobe Systems, Inc. Director and xRes are registered trademarks of Macromedia, Inc. Macintosh and MacOS are registered trademarks of Apple Computer, Inc., and Windows 95, 98 and Windows NT are trademarks of Microsoft Corporation. Electric Image is a trademark of Electric Image Inc.

License Agreement

You should carefully read the following terms and conditions before proceeding. By choosing the "Agree" button below you are agreeing and indicating your acceptance of these terms and conditions. If you do not agree with them you should choose the "Disagree" button below and the Program will not proceed .

Media Lab, Inc. (hereinafter referred to as Company), a Colorado based Corporation, provides the Effector Set I software (hereinafter referred to as Program) contained on the medium in this package and licenses its use. You assume full responsibility for the selection of the Program to achieve your intended results and for the installation, use and results obtained from the Program.

License

This license agreement applies to the unlocked Effector Set I software only. You are allowed to distribute the demo Effector Set I software freely. You are prohibited, however, from giving your unlock code to any other user or from using the unlocked Program on more than one computer simultaneously. The unlocked Program is licensed to be used with a single copy of Macromedia Director.

A. In consideration of the payment of a license fee, you are granted a personal, nontransferable and nonexclusive license to use the Program under the terms stated in the Agreement. You own the diskette or other physical media on which the Program is provided under the terms of this Agreement, but all title and ownership of the Program and enclosed related documentation (hereinafter referred to as Documentation), and all other rights not expressly granted to you under this Agreement, remain with the Company.

B. The Program may be used by you only on a single computer. Therefore, you must treat this software just like a book. With the exception of Item D below, just like a book means that this software may be used by any number of people and may be freely moved from one computer location to another, so long as there is no possibility of it being used at one location while the same copy is at the same time being used at another location. Just as the same copy of a book cannot be read by two different people in two different places at the same time, neither can the same copy of software be used by two different people in two different places at the same time. Contact Media Lab, Inc. for details.
C. You and your employees and agents are required to protect the confidentiality of the Program. You may not distribute or otherwise make the Program or Documentation available to any third party.
D. You may not copy or reproduce the Program or Documentation for any purpose except to make one (1) archival copy of

the Program, in machine readable or printed form for back up purposes only in support of your use of the Program on a single computer. You must reproduce and include the Company copyright notice on the backup copy of the Program.

E. Any portion of the Program merged into or used in conjunction with another program will continue to be the property of the Company and subject to the terms and conditions of this Agreement. You must reproduce and include Company's copyright notice on any portion merged in or used in conjunction with another program.

F. You may not sublease, assign or otherwise transfer the Program or this license to any other person without the prior written consent of Company.

G. You acknowledge that you are receiving on a LIMITED LICENSE TO USE the Program and Documentation and that the Company retains title to the Program and Documentation. You acknowledge that Company has a valuable proprietary interest in the Program and Documentation. You may not use, copy, modify or transfer the Program or Documentation or any copy, modification or merged portion in whole or in part except as expressly provided for in this Agreement. If you transfer possession of any copy modification or merged portion of the Program or Documentation to another party, your license is automatically terminated.

Term

This license granted to you is effective until terminated. You may terminate it at any time by returning the Program and Documentation to Company together with all copies, modifications and merged portions in any form. The license will also terminate upon conditions set forth elsewhere in the Agreement, or if you fail to comply with any term or condition of this Agreement. You agree upon such termination to return the Program and Documentation to Company together with all copies, modifications and merged portion in any form. Upon termination, Company can also enforce any rights provided by law.

The provision of this Agreement which protects the proprietary rights of Company will continue in force after termination. Termination of this license either voluntary or involuntary does not entitle you to a refund of your purchase cost except as provided elsewhere in this License Agreement.

Limited Warranty

Company warrants, as the sole warranty, that the medium on which the Program Is furnished will be free from defects in materials and workmanship under normal use and conditions for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt. No distributor, dealer or any other entity or person is authorized to expand or alter either this warranty or this Agreement. Any such representations will not bind the Company. Company does not warrant that the functions contained in the Program will meet your requirements or that the operation of the Program will be uninterrupted or error-free.

Except as stated above in this section, the Program and Documentation are provided as is without warranty of any kind either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. You assume entire risk as it applies to the quality and performance of the Program and Documentation. Should the Program prove defective you and not Company, authorized Company Distributor or dealer, assume the entire cost of all necessary servicing repair or correction. This warranty gives you specific legal rights and you may also have other rights which vary from state to state. Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Limitation of Remedies

Company's entire liability and remedy will be:

A. The replacement of any medium not meeting Company's Limited Warranty explained above and which is returned to Company or an authorized Company distributor or dealer with a copy or your receipt; or

B. If Company is unable to deliver a replacement medium which conforms to the warranty provided under this Agreement, you may terminate this Agreement by returning the Program and Documentation to Company, authorized Company Distributor, or dealer from whom you obtained the Program and your license fee will be refunded.

Product Returns

If you must ship the Program and Documentation to an authorized Company Distributor, dealer or to Company, you must prepay shipping and either insure the Program and Documentation or assume all risk of loss or damage in transit. To replace a defective medium during the ninety (90) day warranty period, if you are returning the medium to Company, please send us your name and address, the defective medium and a copy of your receipt at the address provided below. In no event will Company be liable to you for any damages direct, indirect, incidental or consequential, including damages for any lost profits, lost savings or other incidental or consequential damages arising out of the use or inability to use such Program and Documentation, even if Company has been advised of the possibility of such damages or for any claim by any other party. Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you. In no event will Company liability for damages to you or any other person ever exceed the amount of the license fee paid by you to use the Program regardless of the form of the claim.

US Government Restricted Rights

The Program and Documentation are provided with restricted rights. Use, duplication or disclosure by the US Government is subject to restrictions as set forth in subdivision (b) (3) (ii) of the Rights in Technical Data.

General

This Agreement is governed by the laws of the state of Colorado (except federal law governs copyrights and register trademark(s)). If any provision of this Agreement is deemed invalid by any court having jurisdiction, that particular provision will be deemed deleted and will not affect the validity of any other provision of this Agreement. Should you have any questions concerning this Agreement, you may contact Media Lab, Inc. at the address below.

Media Lab, Inc., 31 S. 80th Street, Boulder, CO Phone: (303) 499-5411, Fax: (303) 497-9454
http://www.medialab.com