KoolMoves

User Guide

Buttons

Lucky Monkey Designs, LLC
South Carolina, USA
www.koolmoves.com

The information in this document is subject to change without notice and should not be construed as a commitment by Lucky Monkey Design. While every effort has been made to insure the accuracy of the information contained herein, Lucky Monkey Design assumes no responsibility for errors or omissions. Lucky Monkey Design assumes no liability for damages resulting from use of the information contained in this document.

Flash™ is a registered trademark of Adobe, Inc. in the United States and other Countries.

The software described in this document is furnished under license and may be used or copied only in accordance with the terms of that license.

This publication may be reproduced, photocopied, stored on a retrieval system, or transferred without the expressed prior written consent of Lucky Monkey Design.

PRINTING HISTORY

November 2006   Version 0.1
November 2006   Version 0.2

Table of Contents

Preface: About This User Guide

This user guide is an adjunct to the KoolMoves manual, and is not intended as a stand alone document. The sole purpose of this user guide is to illustrate a narrow area of functionality in the KoolMoves application. Users of this guide are expected to have reasonable access to the KoolMoves user manual, as this guide contains references to the operation of KoolMoves and its interface, both of which are covered extensively in the manual. This user guide has been composed and typeset in Windows™ Word®, and exported to the Adobe™ Portable Data File (PDF®) format.

Any questions about or comments on this user guide can be Private Messaged to pherbrick via the KoolMoves support forum at http://www.flashkit.com/board/forumdisplay.php?forumid=24.

Introduction

Modern computer software cannot operate without **Buttons**. Buttons are graphic "hot spots" that make things happens. Flash buttons are used in interactive animations, websites, and client applications written in ActionScript. There are many commercial applications that do nothing but create Flash buttons for use in websites.

In KoolMoves, any shape, group of shapes, imported image, movie clip or static text object can be turned into a button with a single mouse click. Additionally, KoolMoves comes with about 70 professionally designed buttons that can be previewed and added to a movie with just a few mouse clicks.

The power and complexity of the button actions that can be created with KoolMoves are only limited by the skill and knowledge of the user. On the other hand, beginners can make powerful buttons without any knowledge of ActionScript, or much experience with KoolMoves either, by using dialog windows.

The Button Properties section provides a detailed look at how buttons are handled in KoolMoves. Read the Creating Buttons section for step by step guides to making buttons. The three ways that actions can be applied to buttons are covered in the Button Actions section. To add emphasis to your buttons, check out the Adding Sound to Button States section.

If you have any questions about buttons that are not covered in this user guide, please search the KoolMoves support forum at http://www.flashkit.com/board/forumdisplay.php?forumid=24 before posting your question (and be descriptive with the subject field – this helps make the forum more useful). ActionScript questions can also be researched in the KoolMoves forum – many useful ActionScript references, both online and hardcopy, are identified in various posts there, as well as code samples for specific problems.

Button Properties

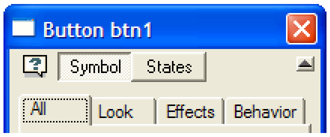
Buttons cannot be created dynamically, though many properties can be changed dynamically through ActionScript programming. In KoolMoves, the properties of objects placed on the Stage at authoring time are edited with the Properties Data View.

Symbol / Library Symbol

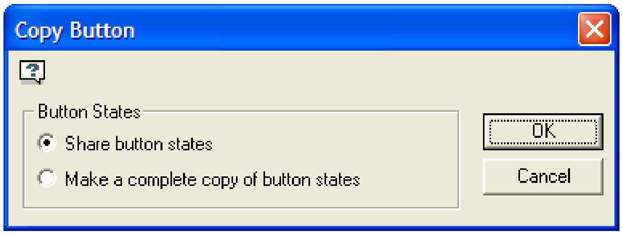
There are Symbols, and then there are Library Symbols. Buttons can be both. However, how a button acts as a symbol depends on how it is copied.

Copying Buttons

In KoolMoves, a symbol is a shape that is shared among several instances. By default a button is always a symbol; in the Properties data view you can switch between the symbol's properties or its states.



When a button is copied from one key frame to another, the appearance and behavior of the buttons are permanently linked. Changes to the appearance and / or actions in one instance of the button are reflected in all instances of the button. However, you are not limited to just making clones; copying and pasting a button to its source key frame opens the Copy Button window. Here you can choose between two additional types of copied buttons.



Button States

● Share button states

○ Make a complete copy of button states

A button pasted with the Share button states option has linked appearances with its parent button. Changes to a button's appearance will be reflected in all instances of the button. Changes to an instance's behavior will not be reflected in all other instances – in fact, these copies start with no actions defined.

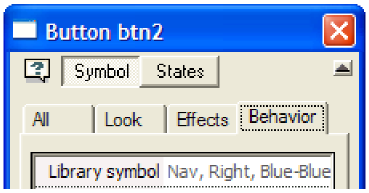
A button pasted with the Make a complete copy of button states option is a brand new button that starts with the same appearance and actions of the source button. Changes to this button will not be reflected in its parent button or any instances linked to the parent button.

To briefly recap the three types of button copies that can be made:

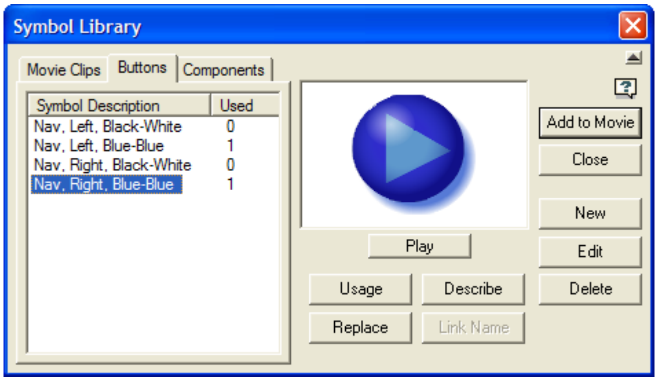
- Paste copy to new key frame – you automatically get linked appearance and behavior,
- Paste copy to source key frame – choose linked appearance, and
- Paste copy to source key frame – complete copy with no links to other buttons.

Buttons & the Symbol Library

Library symbols are movie clips, buttons, and components that have been stored in the symbol library. The library symbol field indicates whether a button instance is based on a symbol library button; clicking on this field does not do anything. Use View > Symbol Library or press F11 to open the symbol library.



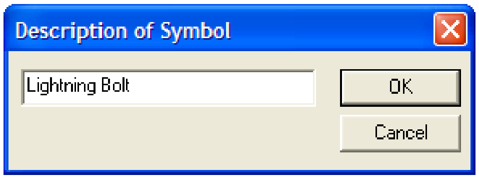
Library symbol Nav, Right, Blue-Blue



Symbol Description	Used
Nav, Left, Black-White	0
Nav, Left, Blue-Blue	1
Nav, Right, Black-White	0
Nav, Right, Blue-Blue	1

Under the Buttons tab there is a list of all buttons stored in the symbol library; their descriptions and the number of times they have been added to the movie are displayed. The symbol library is used to organize, track and edit button designs used throughout a movie. A change to a library button's appearance is reflected in all instances, but each button has unique actions like the linked appearance buttons described in Copying Buttons.

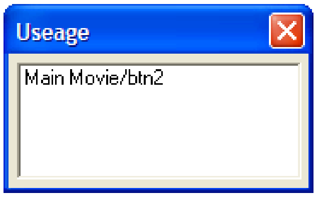
On the stage select a button and press Ctrl-F11 (Shapes > Add to Symbol Library) to add it to the symbol library. Give the button symbol a descriptive name, either about its appearance (if multiple copies with different actions will be used) or its purpose (if all copies will have the same or similar actions).



Lightning Bolt

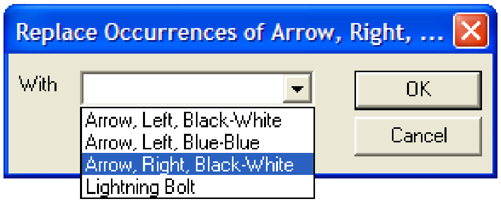
Regarding the button library commands:

- Click on Play to preview a button's states in an internal player.
- Click on New to create a button from scratch (see Creating a Button from Scratch).
- Click on Edit to change the appearance of a button (see Creating a Button from Scratch).
- Click on Delete to remove a button from the Symbol Library.
- Click on Usage for a listing of that symbol's instance names in the movie.
- Link Name is grayed out – it doesn't apply to buttons.
- Click on Describe to edit the button's description.



Main Movie/btn2

- Click on Replace to replace all instances of the button in the movie with another button (button actions are not affected). Select a replacement button from the drop down list.

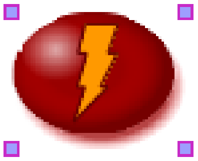


Arrow, Left, Black-White
Arrow, Left, Blue-Blue
Arrow, Right, Black-White
Lightning Bolt

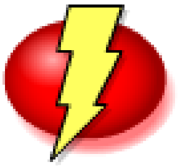
States

A button can have three different appearances depending on the mouse actions. These are known as button **States**. Ideally, each state has a unique appearance.

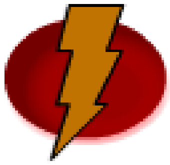
The **Up state** is the default state, and reflects the normal appearance of the button - the mouse pointer is not currently over or clicking the button.

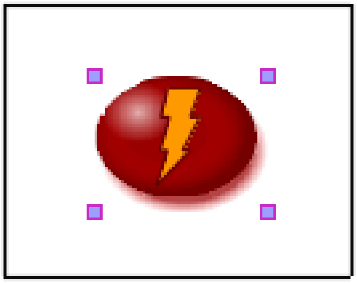


The **Over state** occurs when the mouse pointer is over the button's "hit area".

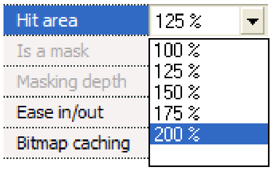


The **Down state** occurs when the mouse is left-clicked on the button's hit area. Button actions are usually executed when the button is clicked or released.





The button area that is receptive to the mouse pointer is called the **Hit Area**. The hit area is expressed as a percentage of the area of the **Up state** of the button, and defaults to 100%. A 200% hit area is twice as big horizontally and vertically as the outline of the button object.



Hit area 125 %
Is a mask
Masking depth
Ease in/out
Bitmap caching

100 %
125 %
150 %
175 %
200 %

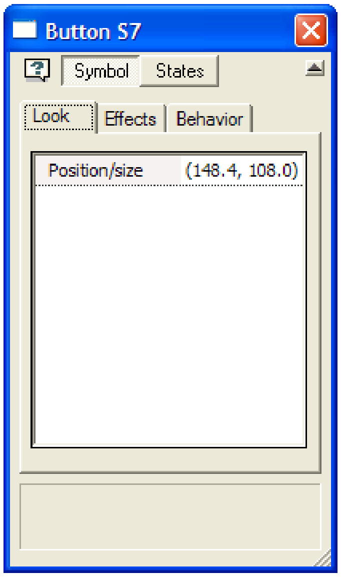
Up

The mouse pointer usually changes appearance while over the hit area (the actual pointer appearance depends on local PC settings).

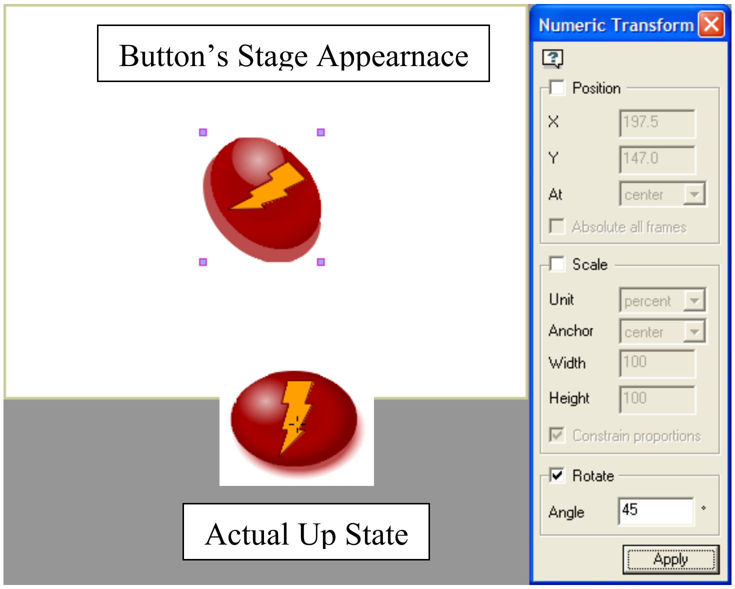
Over/Down

Look

Under the Look tab, click on the button's coordinates to open the Numeric Transform window (Ctrl-Alt-N, or from the menu Transforms > Numeric…). Use this for precision adjustments to a button's position, dimensions, and rotation angle.



Position/size (148.4, 108.0)

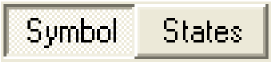


☐ Position
X 197.5
Y 147.0
At center
☐ Absolute all frames
☐ Scale
Unit percent
Anchor center
Width 100
Height 100
☑ Constrain proportions
☑ Rotate
Angle 45 °

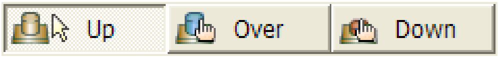
Caveat: Changes made through the Position/size field affect the ***entire*** button; and are reflected on the stage. Scale and rotation changes are not reflected in the individual button states.

To effect changes to a single state, or to maintain a WYSIWYG relationship between a button and its states, perform all edits within the button states.

To edit a button state, first click on the States button.



You can now make changes to the Up state's appearance, or select the Down or Over states to edit.



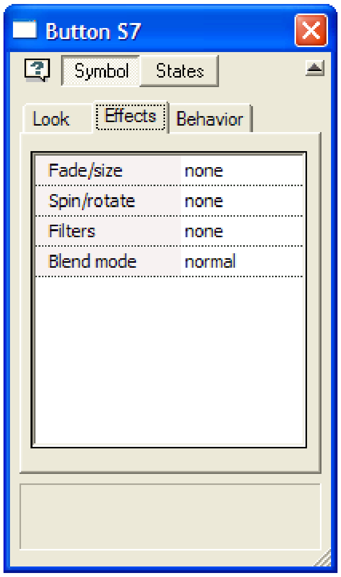
See Creating a Button from Scratch for details on editing button states.

Effects

The **Fade/size** and **Spin/rotate** effects use tweening to impart motion and changes to an object's appearance. Adjust the number of "tween" frames between key frames to regulate the speed of these effects. These effects are covered in Chapter 5 of the KoolMoves user manual.

Flash **Filters** allow for the quick creation of compelling designs; with tweening, the filtering can actually be animated. Because the filters are rendered in real time, there is an associated CPU processing cost. For details on how to use this feature, and some sample images, please see the KoolMoves Filters User Guide.

Blend modes combine the colors of the destination image with the source image, on a pixel by pixel basis, to produce the final image. For details on how to use this feature, and some sample images, please see the KoolMoves Blend Mode User Guide.



Fade/size	none
Spin/rotate	none
Filters	none
Blend mode	normal

Behavior

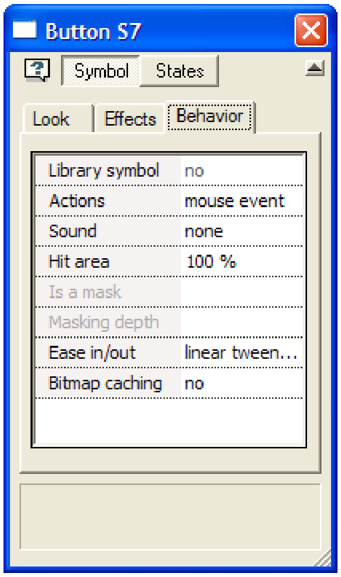
The Behavior tab contains those properties that define a button's performance. The Ease in/out and Bitmap caching fields are covered in this section, while the other fields are covered elsewhere.

The Library symbol field was covered in the Symbol / Library Symbol subsection.

The Hit area field was covered in the States subsection.

The Actions field is covered in the Button Actions section.

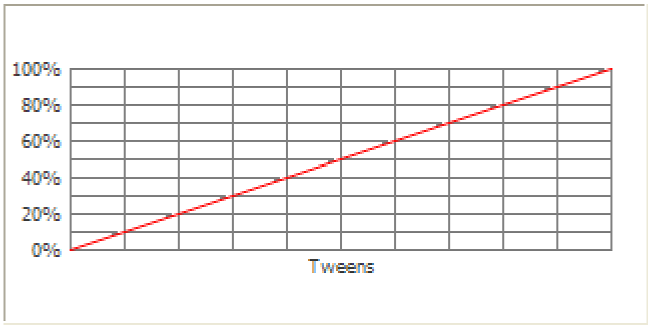
The Sound field is covered in the Button Sounds section.



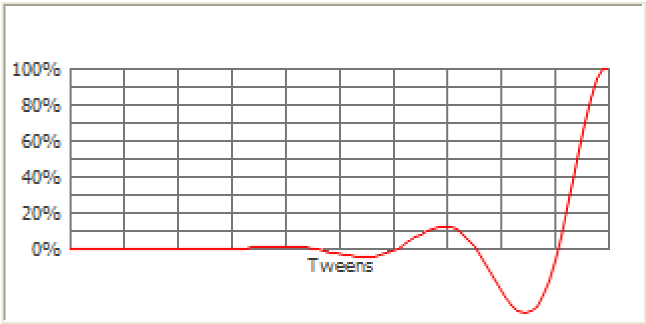
Library symbol	no
Actions	mouse event
Sound	none
Hit area	100 %
Is a mask	
Masking depth	
Ease in/out	linear tween...
Bitmap caching	no

Ease in/out

The Ease In, Ease Out property is used to set the tweening acceleration profiles of an object. KoolMoves comes with 31 acceleration profiles, including the default Linear tweening.



By default, an object that is animated using tween frames moves at a uniform (linear) rate in the tween frames between two key frames. But in the acceleration profile below (Elastic In), the object moves at an uneven rate; in fact, the object actually moves backwards a couple times (like a spring).



Bitmap caching

A major performance limiting factor has been Flash's requirement to update vector graphic images with each screen refresh, even if nothing has changed. In movies with many objects on the screen, this brought down the practical frames per second performance capabilities. Bitmap caching reduces the load on the renderer by storing copies of selected objects in the Flash Player.

The Flash Player will convert the visual contents of buttons into bitmaps and store them in the runtime Flash Player memory. These copies are used for screen refreshes instead of rendering the images from scratch, until the contents of the movie clip change, at which time a new bitmap copy is rendered and stored.

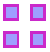
So as long as a button doesn't change, caching can seriously reduce the load on the renderer, especially if the button happens to contain complex vector images (lots of lines, curves, gradient fills, etc.). However, what constitutes a change? If the button moves up or down or sideways – no problem. On the other hand, rotations, scaling, flipping, animated filters, basically any effect, will require the button to be rendered.

Creating Buttons

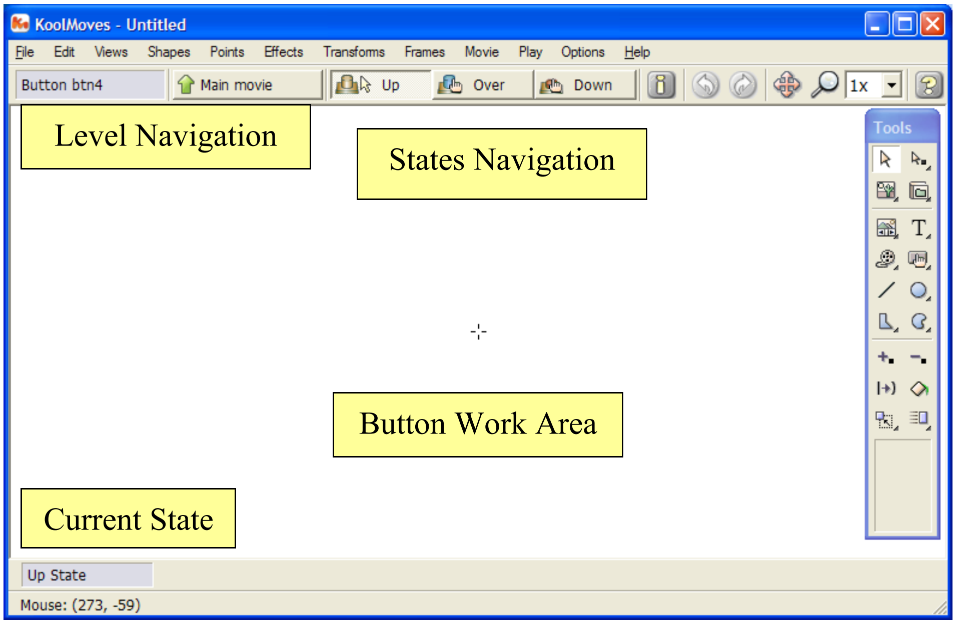
In KoolMoves, any visual element can be turned into a button. However, some elements are better suited than others for specific applications. For example, to test a piece of code, all you need is a visible object on the stage that you can click on – for example a solid circle or word. For fancier buttons that contain effects or animation, it is often easiest to create a movie clip that has the desired appearance and effects, and then convert the clip to a button.

Creating a Button from Scratch

Clicking the Add Button tool creates an empty button and opens the Up State edit screen. Returning to the Main Movie without adding any visual elements to the Up State will result in a 10 pixel by 10 pixel empty button being placed on the Stage. Once the button is not selected it will be completely invisible, and the only evidence of its presence will be its entry in the List of Shapes data view.



Below is the button edit environment for the Up state. At this point you can draw shapes, import an image, or add an empty movie clip. The cross hair indicates the center of the work area (0, 0). The actual button shape will center on the button contents in the Up state, regardless of their location in the button work area. Use the cross hairs, as a reference point, when working with the Numeric Transform tool.



Level Navigation

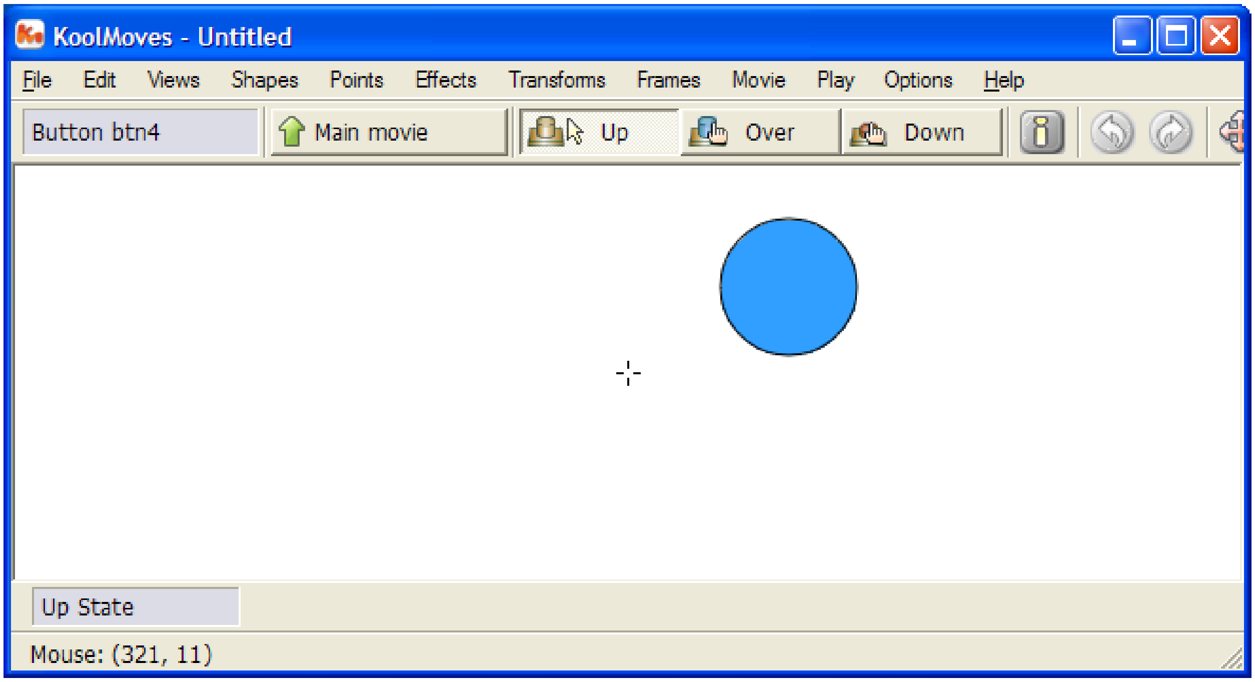
States Navigation

Button Work Area

Current State

The contents of the Over or Down states will not affect the size, shape or location of the button's hit area. The Hit area is solely determined by the contents of the Up state.

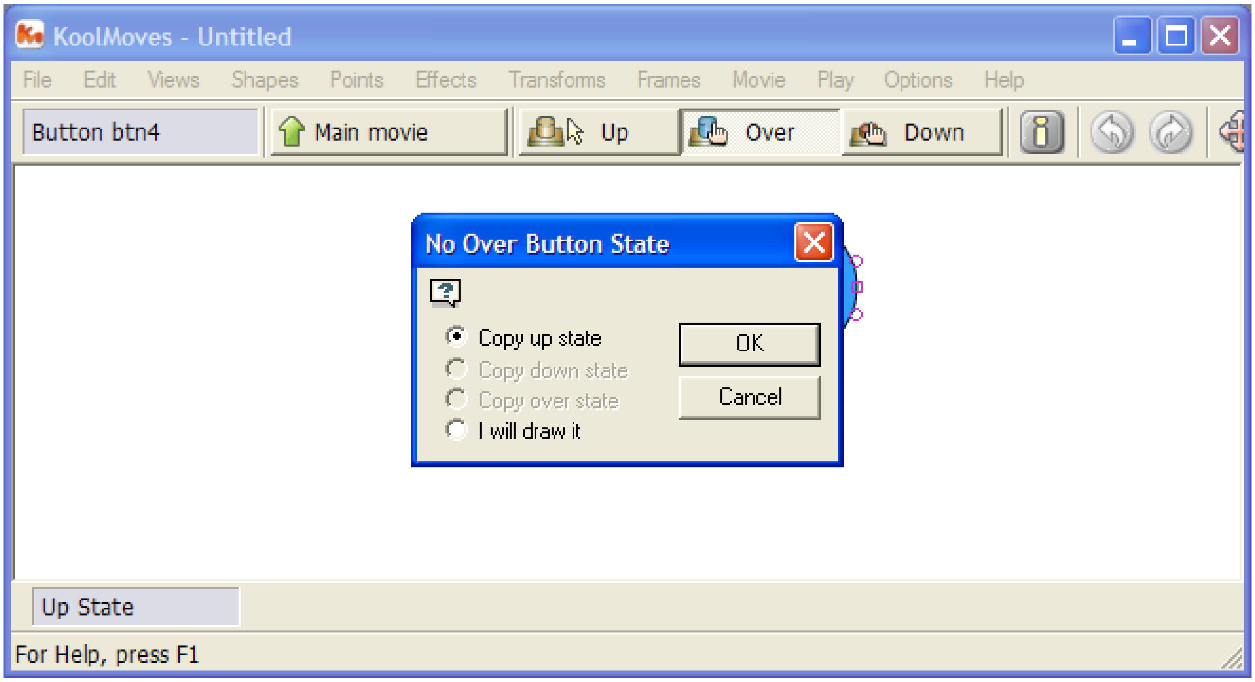
Use the Shape drawing tool to add a circle to the Up State.



Up State

Mouse: (321, 11)

Click on the Over state button, and then click on Copy up state.



No Over Button State

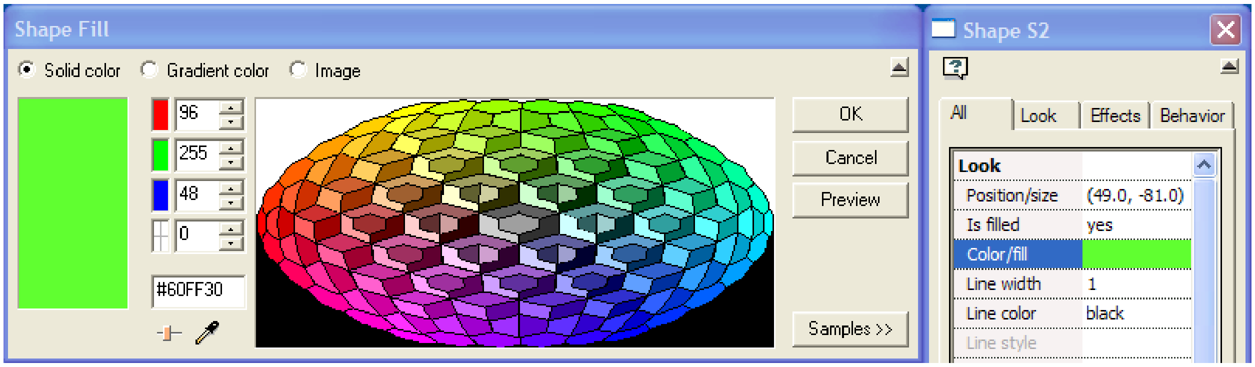
● Copy up state
○ Copy down state
○ Copy over state
○ I will draw it

Up State

For Help, press F1

If you start with a copy of the Up state, and don't change the visual element's dimensions, the contents of the different button states are automatically lined up.

Double click on the circle to open the Properties data view, and change the Over state circle's color.

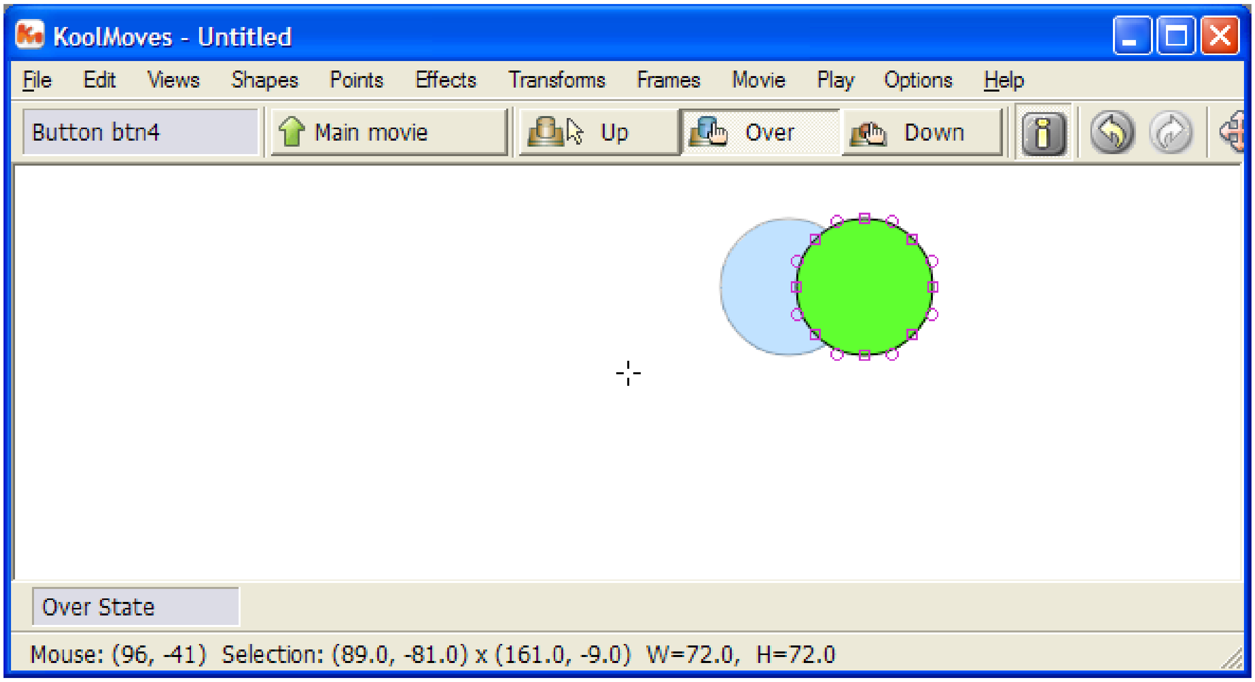


● Solid color ○ Gradient color ○ Image

96
255
48
0
#60FF30

Look	
Position/size	(49.0, -81.0)
Is filled	yes
Color/fill	🟩
Line width	1
Line color	black
Line style	

After you've changed the Over state circle's color, and while the circle is still selected, hold down the Shift key and press the right arrow a few times. You will see a faint image of the Up state circle to the left. This onion skinning effect helps you align on the Up state when you draw your other states from scratch. Okay, now Shift-left arrow the green circle back over the blue circle.



Button btn4

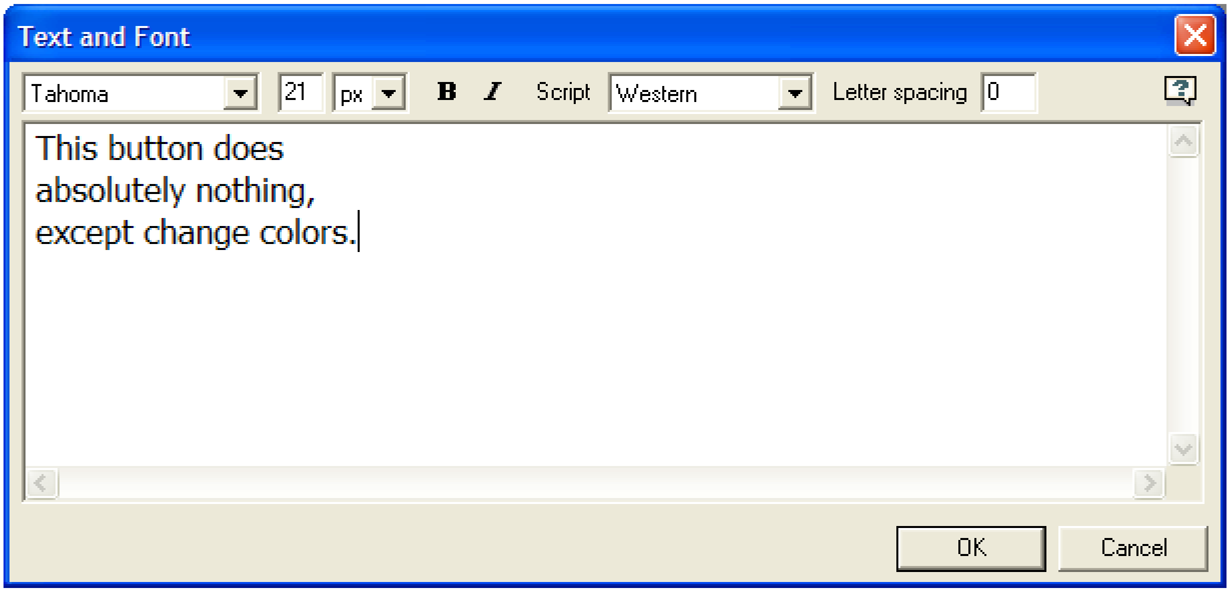
Over State

Mouse: (96, -41) Selection: (89.0, -81.0) x (161.0, -9.0) W=72.0, H=72.0

Note: The Over and Down states do not affect the shape, size or location of the hit area - the hit area is always centered on the Up state contents, and the size will not change to match the contents of the Over or Down states.

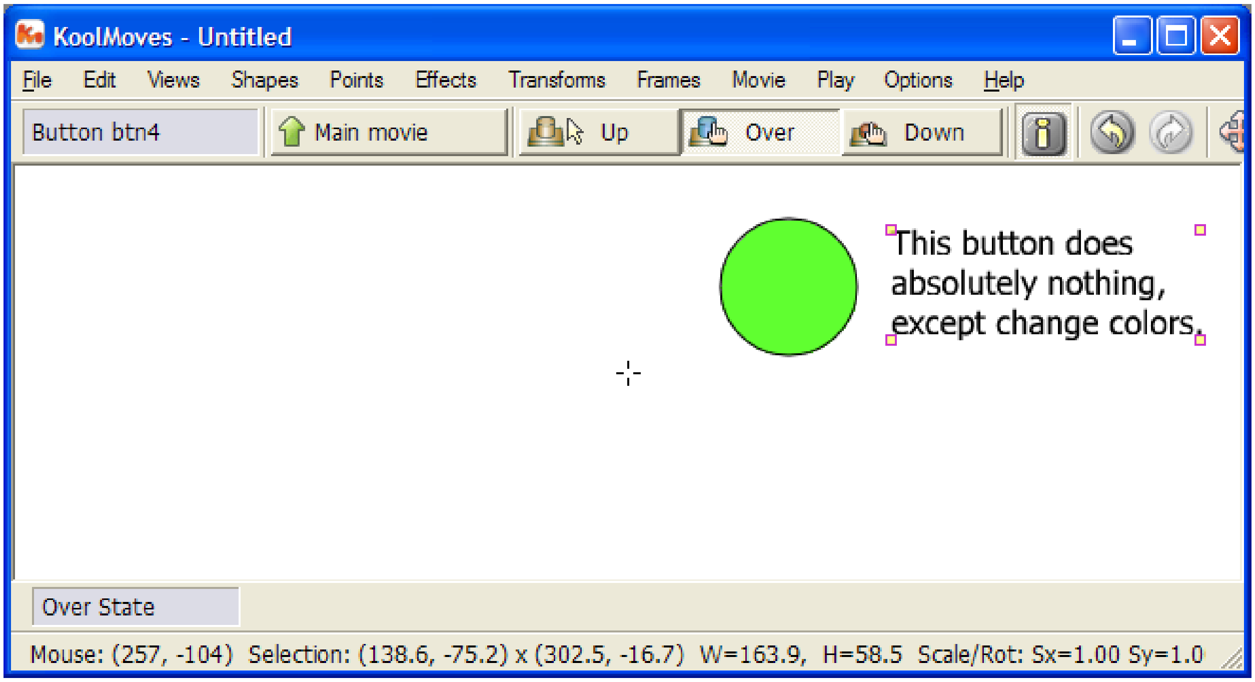
Let's add some text to the Over state. Click on the text tool, and then click in the work area to the right and lined up with the top of the green circle, and add the following text:

This button does
absolutely nothing,
except change colors.



This button does
absolutely nothing,
except change colors.

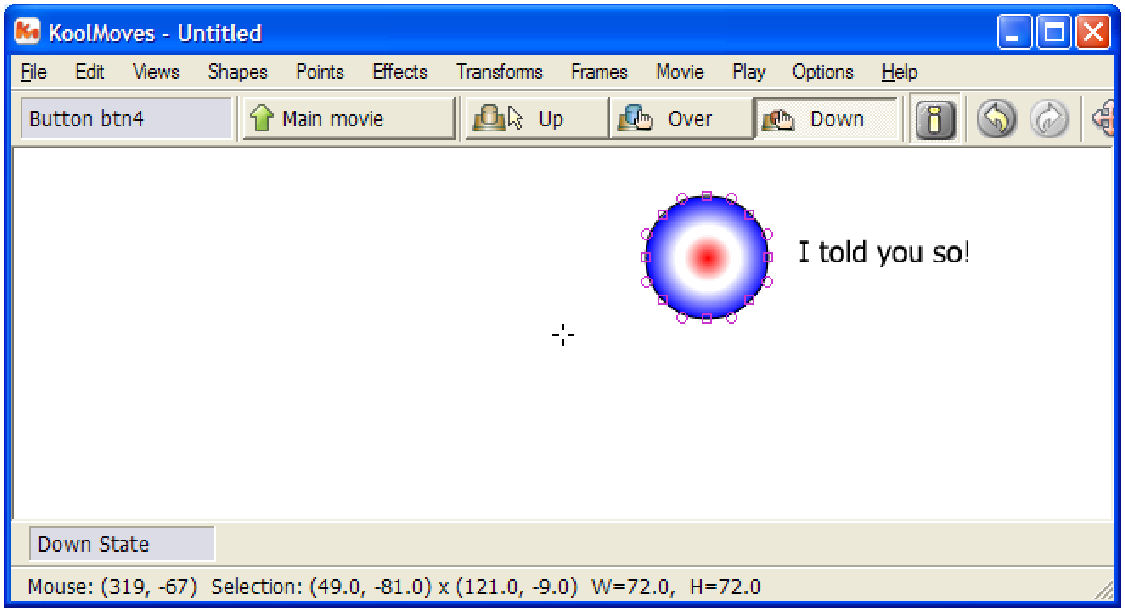
Then click on OK.



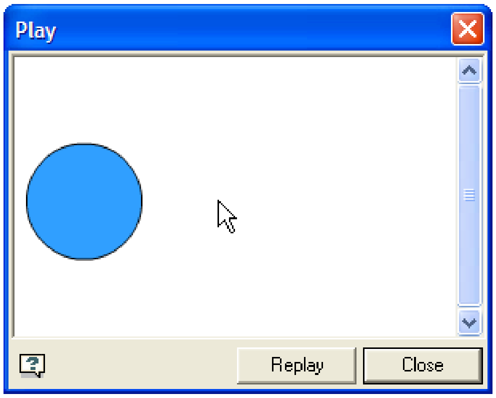
This button does
absolutely nothing,
except change colors.

Mouse: (257, -104) Selection: (138.6, -75.2) x (302.5, -16.7) W=163.9, H=58.5 Scale/Rot: Sx=1.00 Sy=1.0

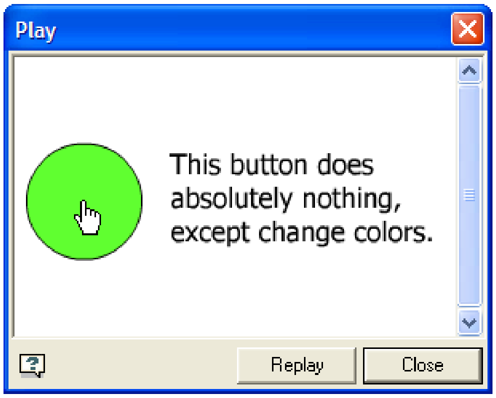
Next, click on the Down state button, copy the Over state, change the circle's color, change the text to "I told you so!" and vertically center the text object relative to the circle.


KoolMoves - Untitled
File Edit Views Shapes Points Effects Transforms Frames Movie Play Options Help
Button btn4
Main movie
Up
Over
Down
I told you so!
Down State
Mouse: (319, -67) Selection: (49.0, -81.0) x (121.0, -9.0) W=72.0, H=72.0

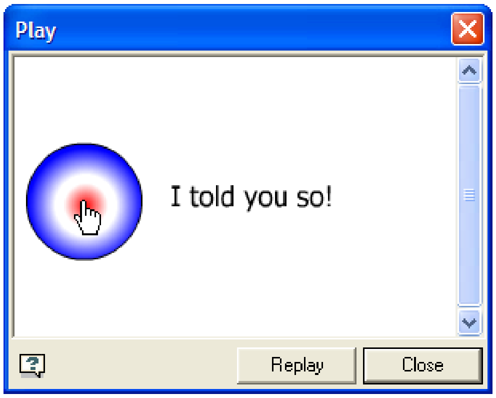
Click on the Main movie button, position the button on the stage so it will display properly, and then Ctrl-Spacebar to see the button in action.



Up



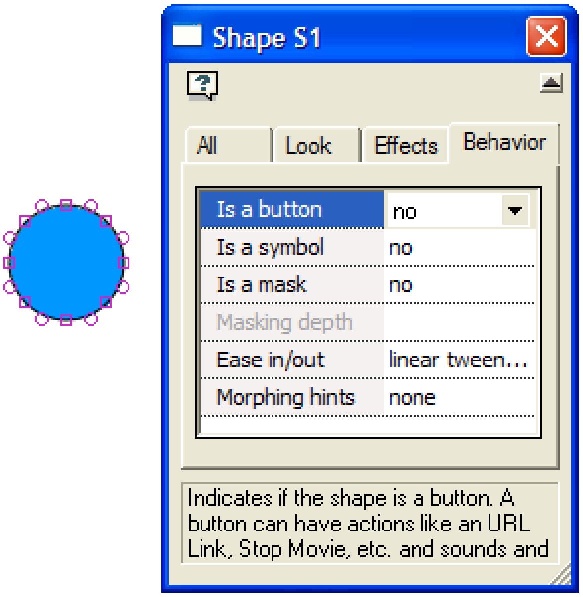
Over



Down

Creating a Button via Property Data View

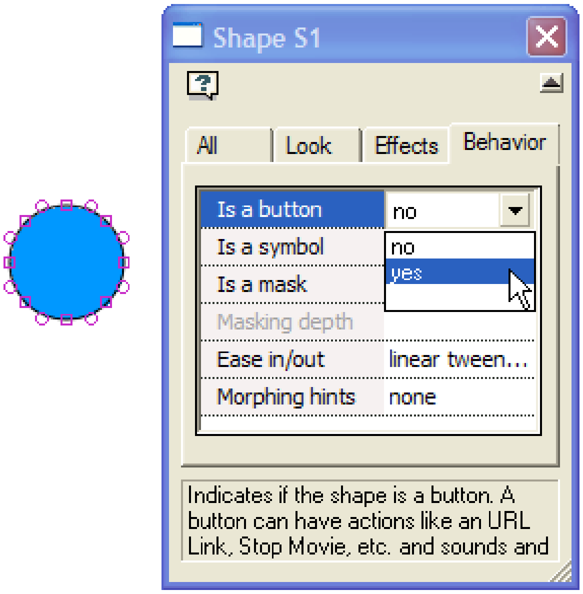
Shapes, shape symbols, static text objects, and groups of shapes can be converted to buttons by setting the "Is a Button" Behavior field to yes in the Properties Data View.



Shape S1

Is a button	no
Is a symbol	no
Is a mask	no
Masking depth	
Ease in/out	linear tween...
Morphing hints	none

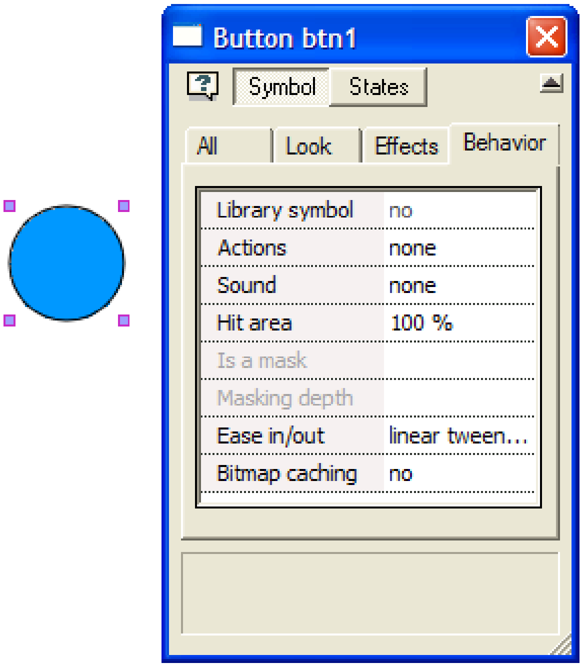
Indicates if the shape is a button. A button can have actions like an URL Link, Stop Movie, etc. and sounds and



Shape S1

Is a button	no
Is a symbol	no yes
Is a mask	
Masking depth	
Ease in/out	linear tween...
Morphing hints	none

Indicates if the shape is a button. A button can have actions like an URL Link, Stop Movie, etc. and sounds and



Button btn1

Library symbol	no
Actions	none
Sound	none
Hit area	100 %
Is a mask	
Masking depth	
Ease in/out	linear tween...
Bitmap caching	no

Objects converted to buttons are placed in the Up state. Click on the States button in the Property data view to open the button edit environment (discussed in the previous section). In the edit environment you can edit the Up state and create the Over and Down states.

Once an object becomes a button, it cannot be changed back; the source object is removed from the stage and placed in the new Up State. To recover an item that has been converted to a button, open the button edit environment, copy the Up State contents, and then paste back to the stage.

Motion & Effects in Buttons

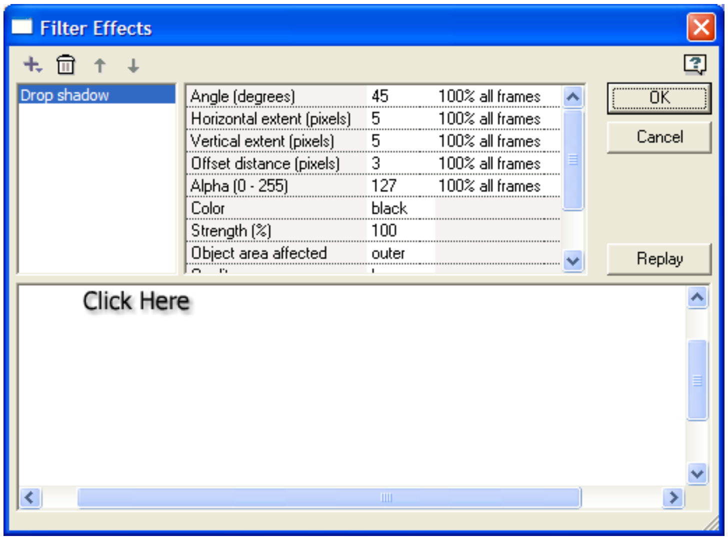
Effects and motion scripts cannot be applied within a button state, as the button state is just a static holding area. For this reason, the Button Edit toolbox, menus and Property data view do not allow you to apply effects or motion scripts to shapes in a button state. However, all these things can be done within a movie clip, and it is very easy to make buttons out of movie clips in KoolMoves.

Before we get into converting visual elements into buttons, let's make a couple movie clips. Open a new project file and create the following:

1) Text with Drop Shadow Filter

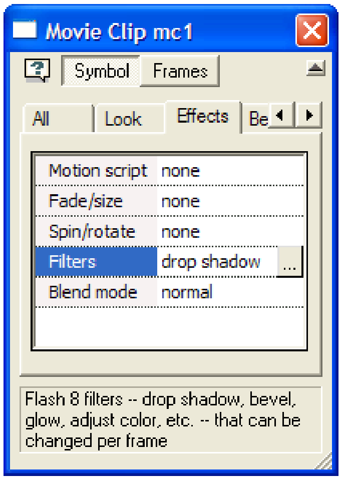
Use the Static Text tool in the toolbox to create a text object containing "Click Here".

Select the text object and use the Convert to Movie Clip tool.



Angle (degrees)	45	100% all frames
Horizontal extent (pixels)	5	100% all frames
Vertical extent (pixels)	5	100% all frames
Offset distance (pixels)	3	100% all frames
Alpha (0 - 255)	127	100% all frames
Color	black	
Strength (%)	100	
Object area affected	outer	

Select the Movie Clip and in the Properties data view, under Effects, open the Filters GUI and apply Drop Shadow.

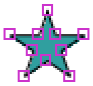


Motion script	none
Fade/size	none
Spin/rotate	none
Filters	drop shadow
Blend mode	normal

Flash 8 filters -- drop shadow, bevel, glow, adjust color, etc. -- that can be changed per frame

2) Shape with 3D Effect

Use the Star Draw tool to draw a five pointed star (should be slightly higher than your "Click Here" text object).



In applying effects to a shape, we can either apply the effect to the shape and then place it in a movie clip, or place the shape in a movie clip and apply an effect to the movie clip. Let’s see what effects are available for a shape versus a movie clip.

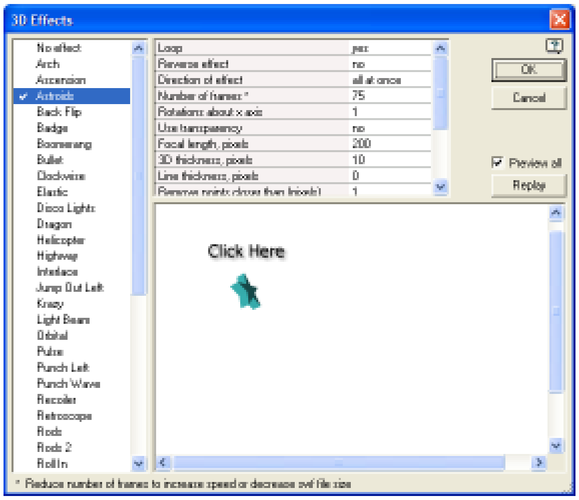
Shape Effects

Motion Script
Fade/Size
Spin/Rotate
3D
Drop Shadow

Movie Clip Effects

Motion Script
Fade/Size
Spin/Rotate
Filters
Blend Mode

We have already applied an effect to a movie clip with the Drop Shadow Filter, so let’s go the other way this time – we will apply a 3D motion to our star, and place the star in a movie clip.

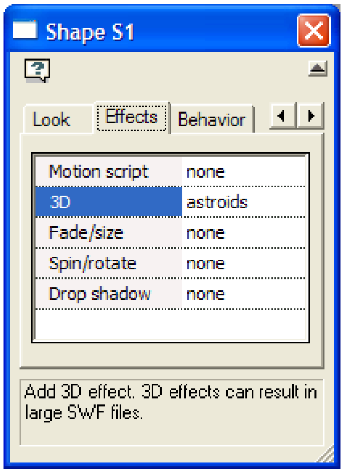

3D Effects

Loop	yes
Reverse effect	no
Direction of effect	all at once
Number of frames *	75
Rotations about x axis	1
Use transparency	no
Focal length, pixels	200
3D thickness, pixels	10
Line thickness, pixels	0
Remove points closer than (pixels)	1

☑ Preview all

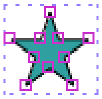
* Reduce number of frames to increase speed or decrease swf file size

Select the Shape and in the Properties data view, under Effects, open the 3D GUI and apply the Astroids effect. Set the Loop property to yes.

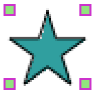

Shape S1

Motion script	none
3D	astroids
Fade/size	none
Spin/rotate	none
Drop shadow	none

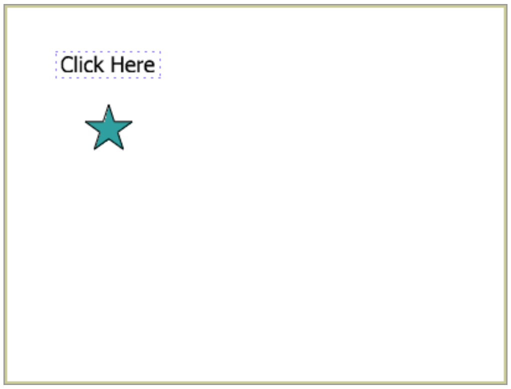
Add 3D effect. 3D effects can result in large SWF files.

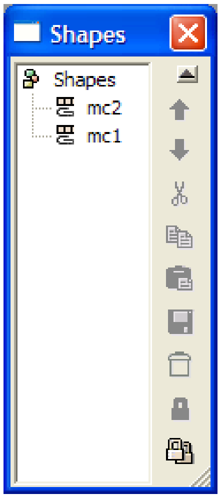


Select the Star and click on the Convert to Movie Clip tool.



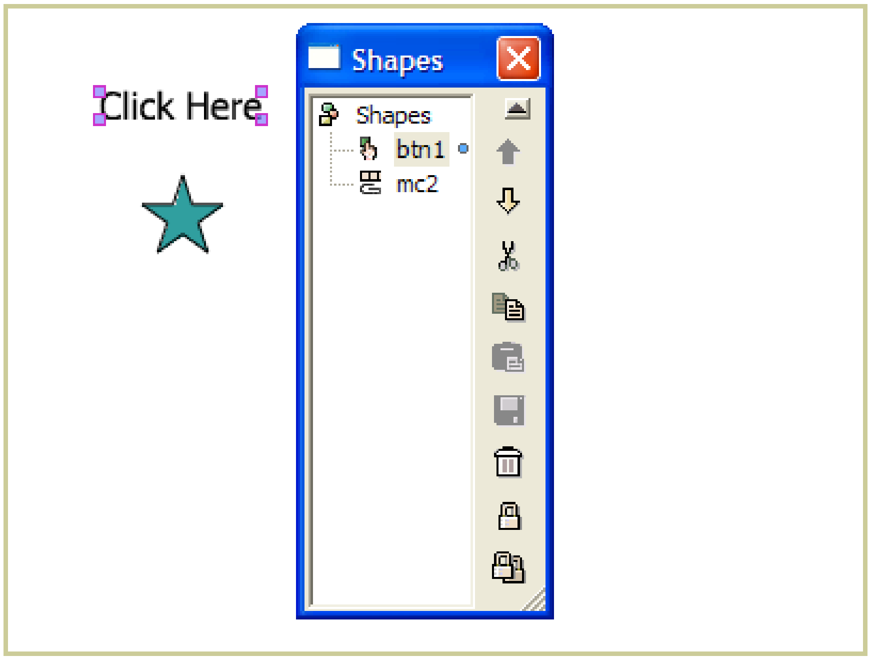
We now have two movie clips to play with. In the next section we will use them to make a button.



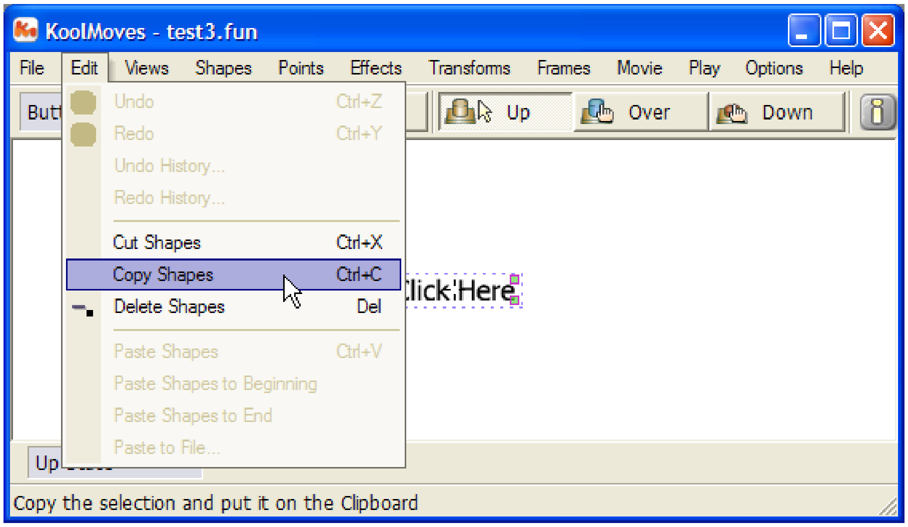


Converting a Visual Element to a Button

Select the Click Here movie clip created in the previous section, and click on the Convert to Button tool. In the List of Shapes data view below, you can see that mc1 is now btn1 (and has also been moved up in the content stack).

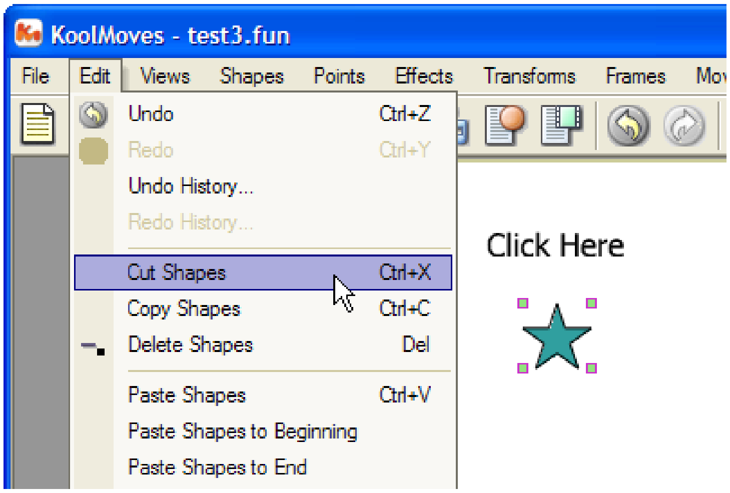


Once an object becomes a button, it cannot be changed back; the source object is removed from the stage and placed in the new Up State. To recover an item that has been converted to a button, open the button edit environment and copy the object, then paste it back to the stage.

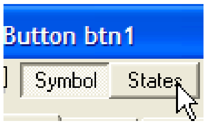


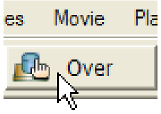
Copy the selection and put it on the Clipboard

Now we are going to move mc2 (the star movie clip) to the Over state. In the main movie, select mc2 and move it to the clipboard by either using Ctrl-X or Edit > Cut Shapes.

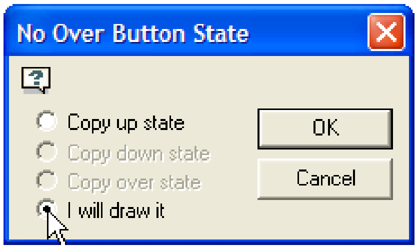


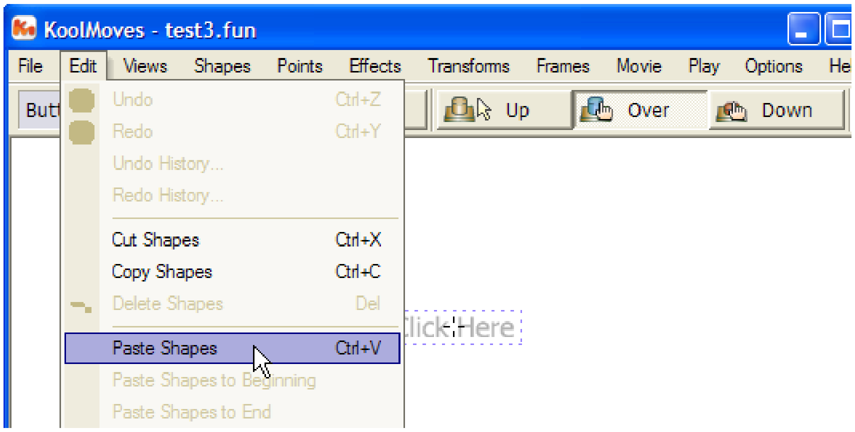
Next, select btn1 (the Click Here button), and in the Property data view click on the States button.





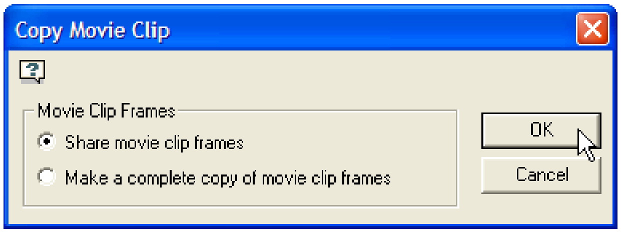
Click on the Over state button below the menu bar, and this time choose the "I will draw it" option in the No Over State window.



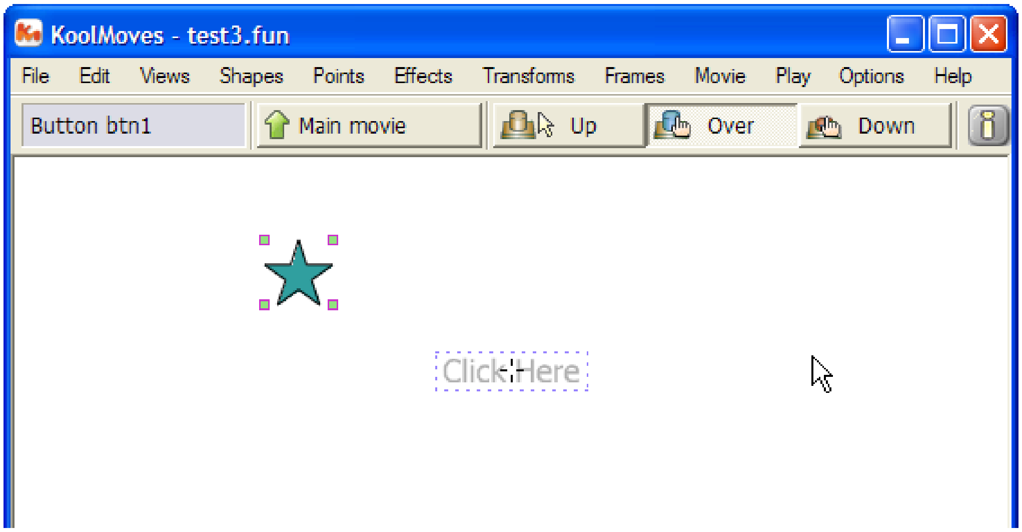


Next, use Ctrl-V or Edit > Paste to put mc2 in the Over state.

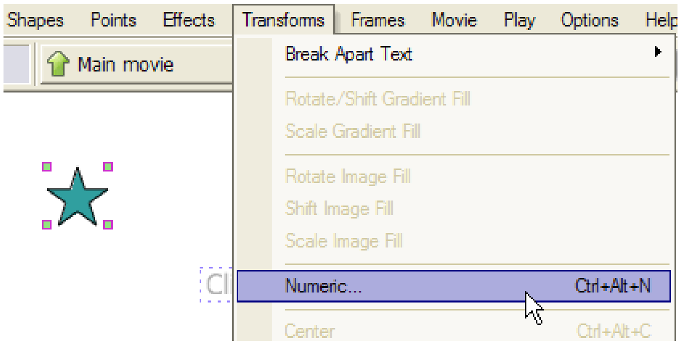
You will be asked if the clip frames should be shared, or if a complete copy of the frames should be made. Since the original has been removed from the movie at this point, it doesn't really matter. However, since we are technically moving the clip, let's select Share movie clip frames to maintain continuity.



Movie Clip Frames
● Share movie clip frames
○ Make a complete copy of movie clip frames



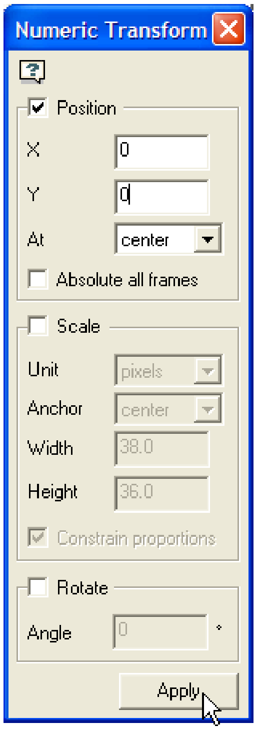
mc2 did not line up over mc1. No biggie, we can fix that quite easily.



You can drag mc2 over mc1, but in this example we will center it exactly using the Transform window (Ctrl-Alt-N or Transform > Numeric).

Note: The origin (0, 0) point in the main movie is in the upper left hand corner of the stage. However, the 0, 0 point in movie clips and button states is the center of the work area. Not a big deal when you drag shapes around with the mouse, but something to remember when you start controlling objects dynamically with ActionScript.

First set the orientation point to Center, then set the coordinates to 0, 0 and click on Apply. ***Voila!***



Numeric Transform

☑ Position

X: 0

Y: 0

At: center

☐ Absolute all frames

☐ Scale

Unit: pixels

Anchor: center

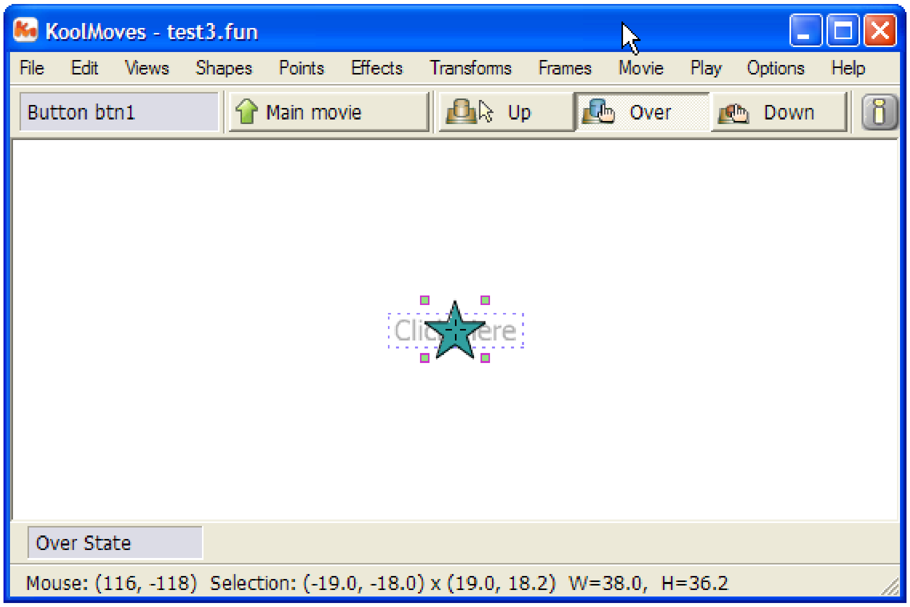
Width: 38.0

Height: 36.0

☑ Constrain proportions

☐ Rotate

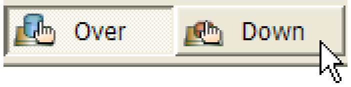
Angle: 0 °



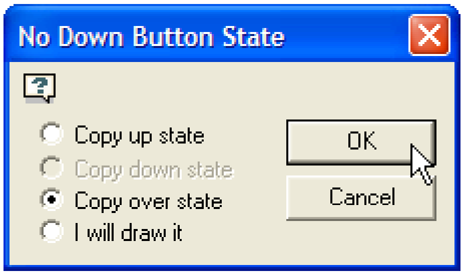
KoolMoves - test3.fun

Mouse: (116, -118) Selection: (-19.0, -18.0) x (19.0, 18.2) W=38.0, H=36.2

Now we need to create a Down State. Click on the Down state button.



Next we copy the Over state to the Down State. The Over state contains a movie clip, so this amounts to copying a movie clip (again!).



No Down Button State

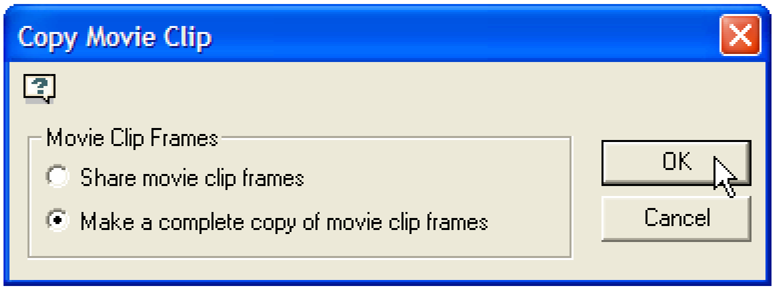
○ Copy up state

○ Copy down state

● Copy over state

○ I will draw it

But this time we do not want to share frames, because if the two clips are linked together, changes to one will be reflected in the other. After we are through here, we will edit the Down state.



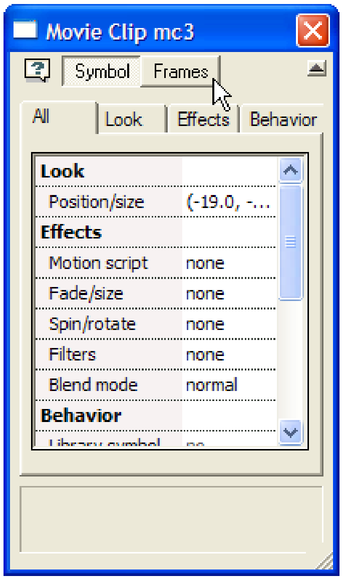
Copy Movie Clip

Movie Clip Frames

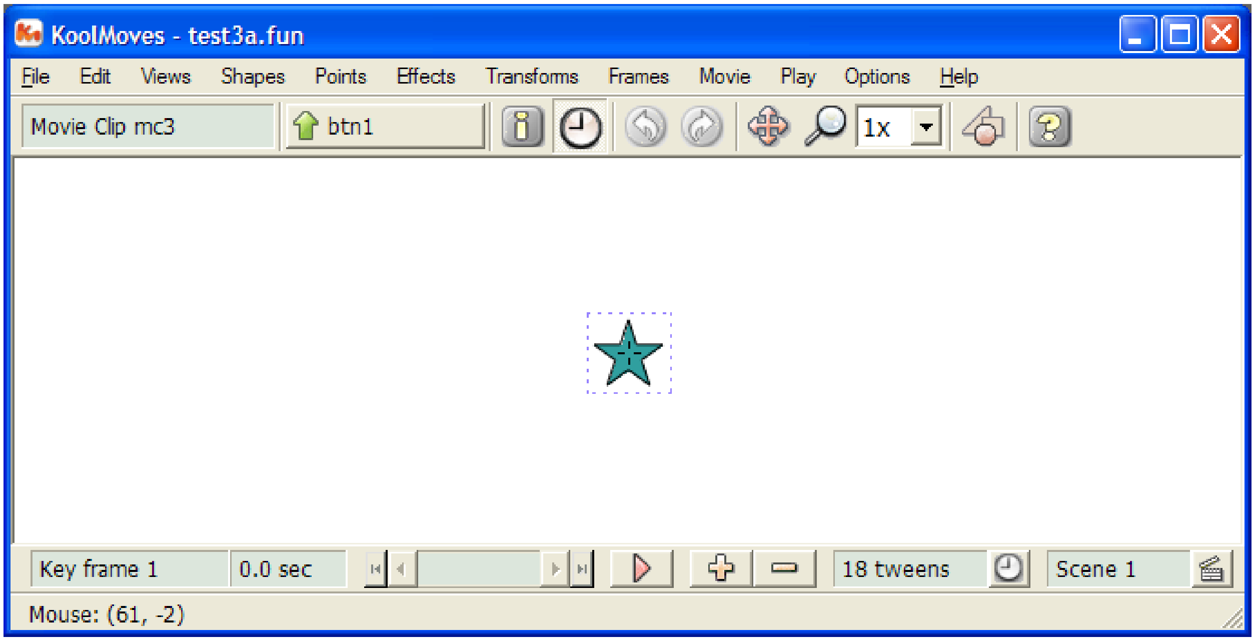
○ Share movie clip frames

● Make a complete copy of movie clip frames

At this point we have a button with all three states, but two of them are identical - we need to alter the appearance of the Down state. Double click on mc3 to open the Property data view, and click on the Frames button.

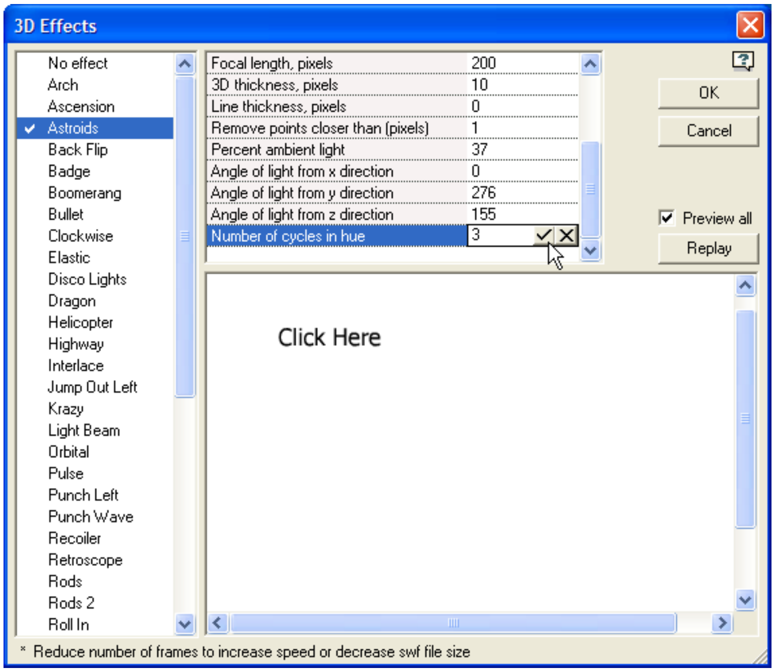


Look	
Position/size	(-19.0, -…
Effects	
Motion script	none
Fade/size	none
Spin/rotate	none
Filters	none
Blend mode	normal
Behavior	
Library symbol	no



Mouse: (61, -2)

Double click on the star shape to open the Property Data View, open the Effects tab, and change the Hue cycle property to 3.



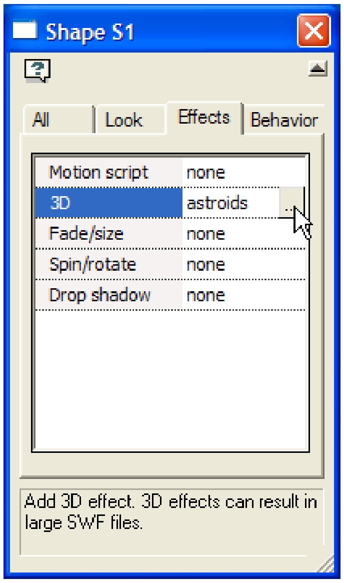
3D Effects

- No effect
- Arch
- Ascension
- ✓ Astroids
- Back Flip
- Badge
- Boomerang
- Bullet
- Clockwise
- Elastic
- Disco Lights
- Dragon
- Helicopter
- Highway
- Interlace
- Jump Out Left
- Krazy
- Light Beam
- Orbital
- Pulse
- Punch Left
- Punch Wave
- Recoiler
- Retroscope
- Rods
- Rods 2
- Roll In

Focal length, pixels	200
3D thickness, pixels	10
Line thickness, pixels	0
Remove points closer than (pixels)	1
Percent ambient light	37
Angle of light from x direction	0
Angle of light from y direction	276
Angle of light from z direction	155
Number of cycles in hue	3

☑ Preview all

* Reduce number of frames to increase speed or decrease swf file size

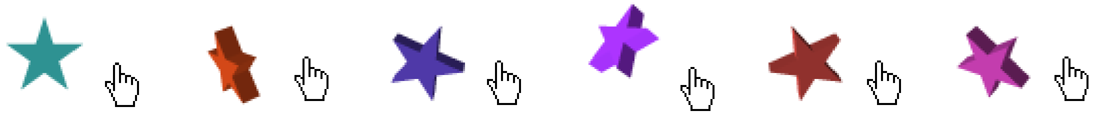


Shape S1

Motion script	none
3D	astroids
Fade/size	none
Spin/rotate	none
Drop shadow	none

Add 3D effect. 3D effects can result in large SWF files.

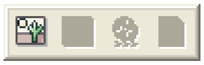
After you've changed the property, you need to move the mouse over "Click Here" in the preview area to see the effect.



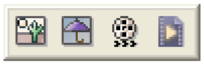
Over State Down State Down State Down State Down State Down State

Loading Images into Button States

When you open the Import image palette in a button state, you only have the Image option.



However, when open the Import image palette from within a movie clip in the button state, the entire palette is active. You could also have loaded these objects to the stage and converted them to a button's Up state, or copied/cut and pasted them to a button state.



Buttons containing raster images (e.g., bmp, gif, jpg, and png) get blurry when resized. Try to initially design the image to the final button size so minimal resizing will be required. Shapes drawn in KoolMoves are vector graphics, and will not get blurry when resized.

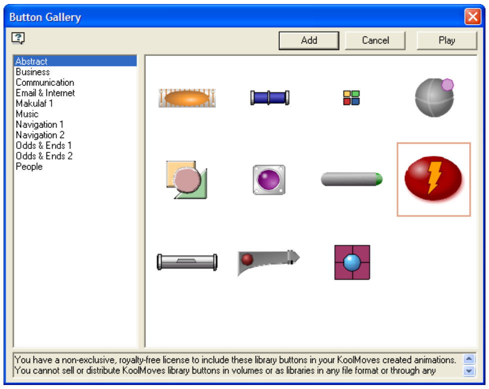
Adding Buttons to Movies

The Button Gallery

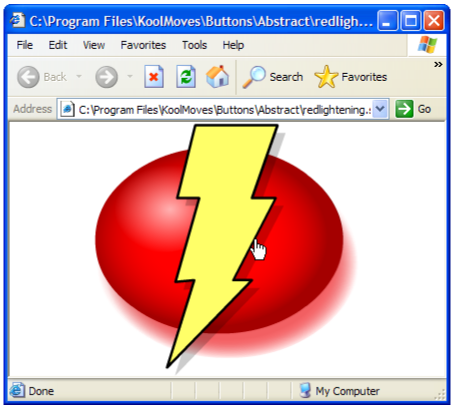
As we have seen, KoolMoves makes it very easy to design unique buttons. The Registered version of KoolMoves comes with about 70 professionally designed buttons already made.

Click on the Button Gallery tool in the Toolbox. Click on a folder category to display the contents.

These buttons were created in KoolMoves and were designed specifically for use in Flash movies. Images made with the KoolMoves drawing tools are vector graphics - they can be scaled without getting blurry, and consume little computer memory.



You have a non-exclusive, royalty-free license to include these library buttons in your KoolMoves created animations. You cannot sell or distribute KoolMoves library buttons in volumes or as libraries in any file format or through any

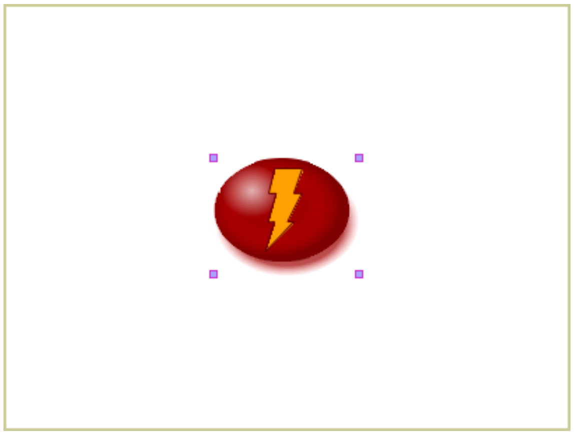

C:\Program Files\KoolMoves\Buttons\Abstract\redligh...
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Address
C:\Program Files\KoolMoves\Buttons\Abstract\redlightening.s
Go
Done
My Computer

Over State Preview

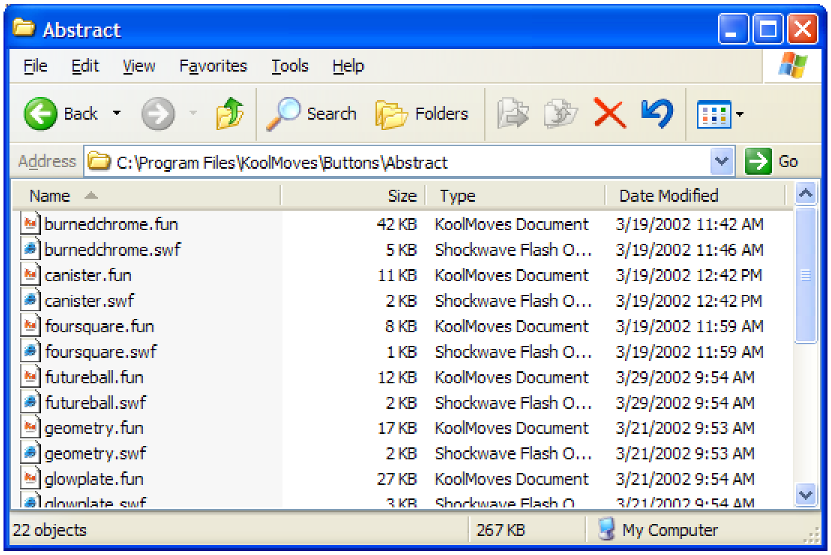
Click on a button to select it, and press Play to display it in the Flash stand-alone player or your default web browser. Move your mouse over the button, and click on the button to see the different behaviors of the button.

In the browser preview, the buttons will scale themselves to the window size.

Press Add to add the button to your movie. The button will be placed in the center of the stage.



See Button Gallery Categories for a list of buttons provided with KoolMoves.



C:\Program Files\KoolMoves\Buttons\Abstract

Name	Size	Type	Date Modified
burnedchrome.fun	42 KB	KoolMoves Document	3/19/2002 11:42 AM
burnedchrome.swf	5 KB	Shockwave Flash O...	3/19/2002 11:46 AM
canister.fun	11 KB	KoolMoves Document	3/19/2002 12:42 PM
canister.swf	2 KB	Shockwave Flash O...	3/19/2002 12:42 PM
foursquare.fun	8 KB	KoolMoves Document	3/19/2002 11:59 AM
foursquare.swf	1 KB	Shockwave Flash O...	3/19/2002 11:59 AM
futureball.fun	12 KB	KoolMoves Document	3/29/2002 9:54 AM
futureball.swf	2 KB	Shockwave Flash O...	3/29/2002 9:54 AM
geometry.fun	17 KB	KoolMoves Document	3/21/2002 9:53 AM
geometry.swf	2 KB	Shockwave Flash O...	3/21/2002 9:53 AM
glowplate.fun	27 KB	KoolMoves Document	3/21/2002 9:54 AM
glowplate.swf	3 KB	Shockwave Flash O...	3/21/2002 9:54 AM

The buttons are stored in the Buttons folder inside the KoolMoves folder. The buttons are actually stored in fun files. swf versions are included to run in the preview window.

You can create your own folders with your own button designs (up to 12 per folder), and load them as you need them.

Creating & Importing from Button Libraries

If you plan on creating and reusing large quantities of buttons, especially buttons that have common themes, store your buttons in symbol libraries instead of Button Gallery folders. A symbol library can hold a large number of buttons in a single .fun file, and the buttons are loaded all at once as opposed to being loaded individually.

Let's say you've created a cowboy themed presentation movie that uses a lot of personally designed buttons, and you would like to reuse them in future projects. No problem. Start by creating a symbols folder and a buttons sub folder (e.g., Program Files\KoolMoves\Z Symbols\Buttons\).

Next, make a copy of the project file and store it in the Buttons folder. Give it a descriptive name reflecting the buttons common theme and/or purpose (e.g., Cowboy-Navigation.fun).

Now copy all the buttons to the Symbol Library (covered in Buttons & the Symbol Library), then create a key frame that has copies of all buttons present. This key frame will be your preview frame.

Finally, delete all other key frames in the project and save the file. This leaves you with a single frame project file filled with buttons. To load the buttons into the current project, use File > Insert KoolMoves Movie…, browse to the Buttons folder, and select the file. As the GUI window previews the selected file, its Symbol Library is copied to the current project's Symbol Library. Don't preview a file more than once – multiple copies of the buttons will be loaded.

Having buttons in the symbol library that are not used will increase the project file (.fun) size, but will not impact the final movie (.swf) size.

Button Actions

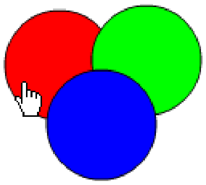
Buttons detect an event (e.g., onPress, onRelease, onRollover, etc.) and then execute commands (e.g., navigate movies, load files, launch web pages, etc.), associated with the event. The behavior of a button is defined either through the Behavior > Actions field in the Property data view, or by adding ActionScript code to the button's timeline.

The Actions field can contain ActionScript or customized predefined commands. Commands in the Actions field travel with the button; using ActionScript code is more efficient than using the predefined commands.

Timeline ActionScript is the most efficient way to define button actions, but requires a moderate knowledge of ActionScript. Additionally, the code does not travel with the button, and needs to be updated each time the button's name is changed or it is moved to a different level or clip.

Remember:

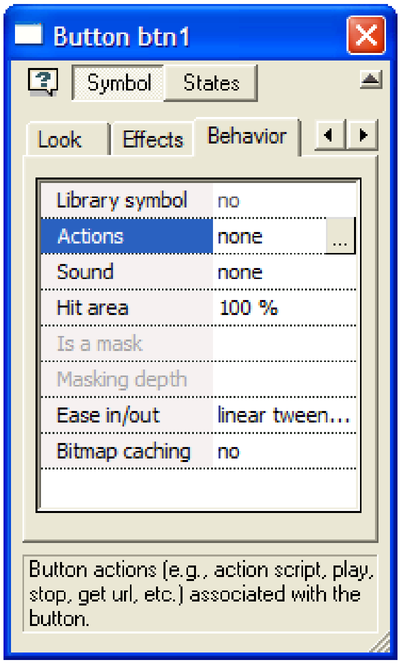
- The Flash Player tends to skip the first key frame. Unless your movie only has one frame, buttons and ActionScript code should not be placed on the first key frame.
- ActionScript code is evaluated as it enters the Flash Player. If the actions for a button are defined on key frame 15 but the button first appears on key frame 3, that button won't work until key frame 15 has been "interpreted". Button actions should be coded at the beginning of the movie (i.e., key frame two) or placed in the Actions property.
- When testing a movie using Play in Browser (Ctrl-Spacebar), any HTML and SWF files referenced by the Get URL or Load Movie actions, ***must*** be in the same directory as the fun file for these button actions to work. The same applies to text files referenced by the Load Dynamic Text command. <u>However</u>, even if these files are in different directories, they can be made accessible to the internal player by selecting their folder in the "Play animation from this" field in File > Preferences > Play.
- When working with Netscape web browsers, avoid spaces in the file name as this will cause problems. Using Http:// in the path may also cause problems with the Netscape web browser.
- A button underneath a shape will continue to work, even if you can't see it.
- When buttons or movie clips with button behaviors overlap, the unobstructed portions will generate the mouse events. Assuming that each button's hit area is at 100%, the mouse pointer is over the red button's hit area. However, if the blue button's hit area is 200%, the pointer is over the blue button's hit area.



- If you are going to stack movies containing buttons in different levels, you need to avoid the "button bleed through" effect. In the higher level movie, place a movie clip the size of the movie at the bottom of the movie's content stack, add an onRollover event handler with no actions, and set the clip's useHandCursor property to false. This clip will block all buttons in lower level movies.

Actions Property > Dialogs

To place an action within the button itself, so that the action travels with the button, select a button and open the Properties Data View.

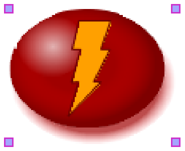


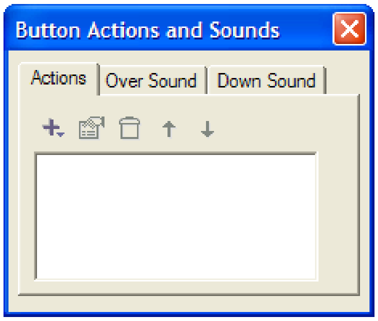
Library symbol	no
Actions	none
Sound	none
Hit area	100 %
Is a mask	
Masking depth	
Ease in/out	linear tween...
Bitmap caching	no

Button actions (e.g., action script, play, stop, get url, etc.) associated with the button.

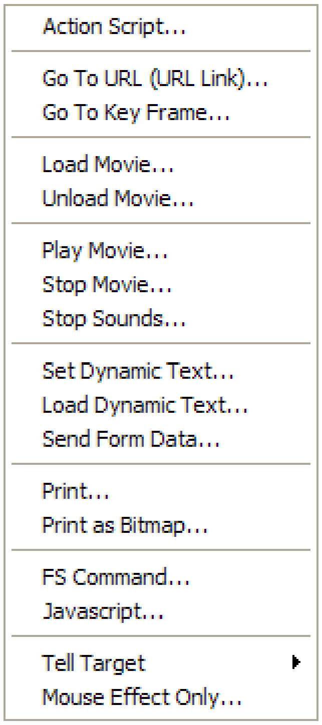
In the Behavior properties, click on Actions to open the Button Actions and Sounds window.

The Actions tab is used to create, edit, delete and organize actions stored in a button.





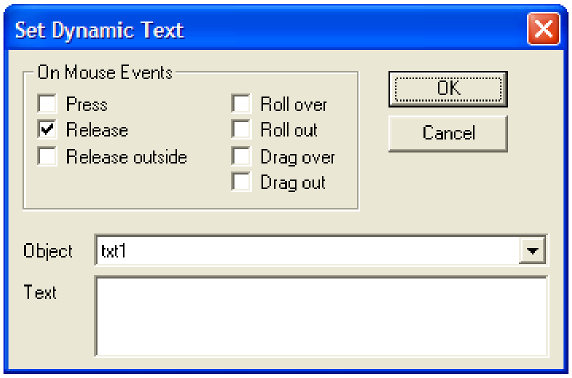
Click on the plus button with the drop down menu arrow to list available actions.



The Action Script… option is used to add ActionScript code to a button.

The Mouse Effect Only option is obsolete and can be disregarded.

The remaining options allow you to add commands to a button without knowing ActionScript. Selecting one of these options opens a task specific dialog window where you will be asked to decide what event will trigger the action. You may also need to set task specific parameters.



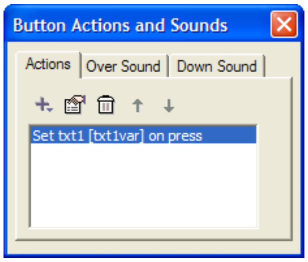
On Mouse Events

□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

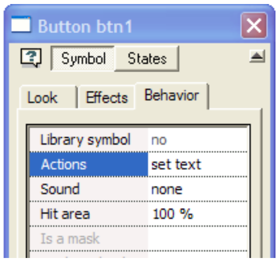
Object txt1

Text

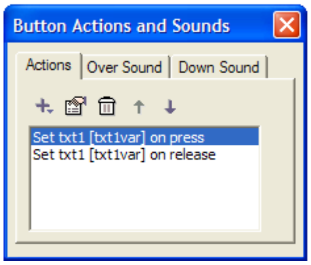
The Button Actions and Sounds window's display area lists the actions that are stored in a button. A button's Property data view will reflect the action on the button, unless multiple actions have been defined, in which case it will display "multiple".



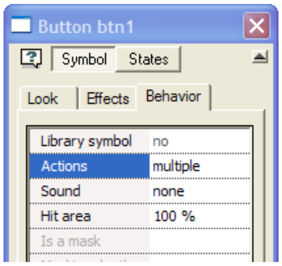
Set txt1 [txt1var] on press



Library symbol	no
Actions	set text
Sound	none
Hit area	100 %
Is a mask	



Set txt1 [txt1var] on press
Set txt1 [txt1var] on release



Library symbol	no
Actions	multiple
Sound	none
Hit area	100 %
Is a mask	

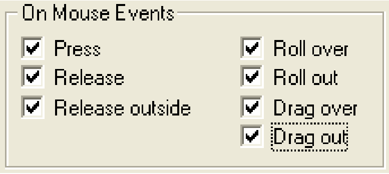
Use the Edit button to change the selected action.

Use the Delete button to remove the selected action.

Use the Up and Down arrow buttons to move the selected action.

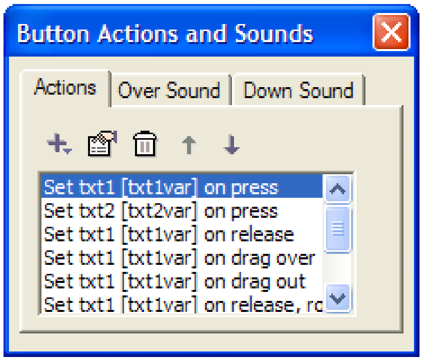
There are seven mouse **Events** in the action dialog window:

- **Press** – The mouse button is clicked on while over the button hit area.
- **Release** – The mouse button is released after clicking on the button hit area.
- **Release Outside** - The mouse button is clicked on the button hit area and the released outside the hit area.
- **Roll Over** – The mouse pointer moves over (enters) a button hit area.
- **Roll Out** – The mouse pointer moves outside (exits) a button hit area.
- **Drag Over** – The mouse button is clicked on the button, and without releasing the mouse button, the mouse pointer is dragged away from the button hit area and then mouse pointer is dragged over the button hit area.
- **Drag Out** – The mouse pointer is dragged outside a button after clicking on the button hit area.


On Mouse Events

☑ Press
☑ Release
☑ Release outside
☑ Roll over
☑ Roll out
☑ Drag over
☑ Drag out

More than one action can be defined per event, and/or unique actions can be defined for every event.



Set txt1 [txt1var] on press
Set txt2 [txt2var] on press
Set txt1 [txt1var] on release
Set txt1 [txt1var] on drag over
Set txt1 [txt1var] on drag out
Set txt1 [txt1var] on release, rc

Note: Certain events are linked by their sequence; e.g., a Press event ***always*** occurs before a Release, Release Outside, Drag Over or Drag Out event.

Go To URL

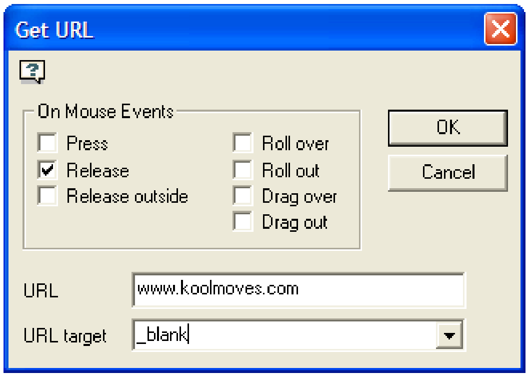
The Go To URL action is equivalent to the ActionScript getURL (URL, URL target) global function, where.

- URL specifies the location (absolute or relative) of the document to load, external script to run, or command to execute.
- URL target identifies the browser window or frame where the document will be loaded. Enter the frame name, or select one of the predefined values:
    - _blank – opens file in new browser window
    - _parent – opens file in current browser frame
    - _self – loads the results of a script execution into the current frame
    - _top – replaces all framesets in browser with file

In the On Mouse Events section, select which event(s) will trigger the getURL() command.

In the URL field, enter the document name or command, including its path where applicable.

In the URL target field, type in the target frame name or select a frame/window from the drop down list



Get URL

On Mouse Events

□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

URL: www.koolmoves.com

URL target: _blank

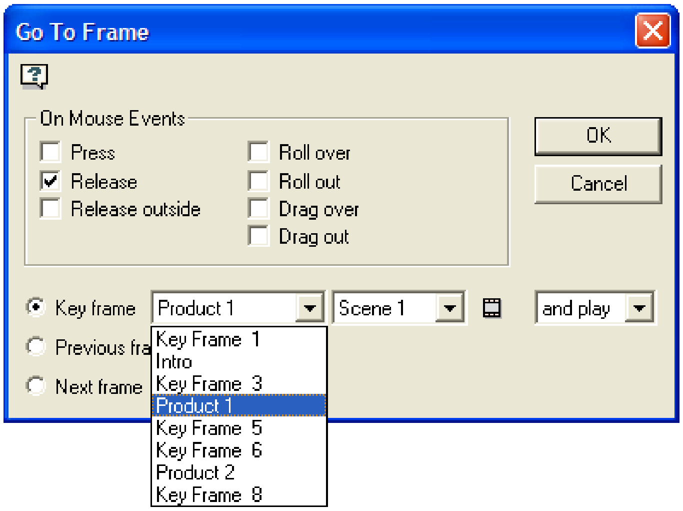
The KoolMoves Go To URL button action supports the following protocols:

Protocol	Format	Purpose
File	file:///path/filename	Access a local file
FTP	ftp://path/filename	FTP a remote file
HTTP	http://path/filename	Open a web page
Javascript	javascript: command	Execute a JS command in a browser
VBscript	vbscript: command	Execute a VB command in a browser
Mailto	mailto:user@domain.com	Send email via the default email app
Telnet	telnet://domain.com:port#/	Open a Telnet connection

If variables need to be sent to an external script or command, use the Send Form Data command, which has the same general format and supports the same protocols.

Go To Frame

Use this action to navigate within a movie – the playhead of the current timeline jumps to the target frame. Click on the Key frame drop down list to display the key frames available in the current scene (the Scene field appears when the movie has more than one scene.



On Mouse Events

□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

● Key frame Product 1 Scene 1 and play
○ Previous fra
○ Next frame

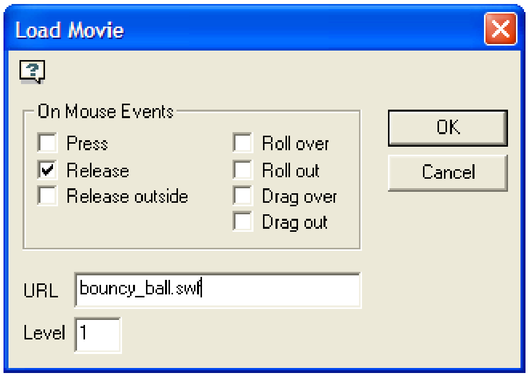
Key Frame 1
Intro
Key Frame 3
Product 1
Key Frame 5
Key Frame 6
Product 2
Key Frame 8

KoolMoves automatically updates a button's target frame when key frames are added, deleted and renamed. However, the internal pointer can get dislocated when a movie's first key frame is removed or shuffled in the frames list.

Use the Previous and Next frame actions to restart the movie either before of after the current frame, respectively (works best when tweens = 0).

Load Movie

Load movie loads a swf without closing the current movie. All of the higher levels have transparent backgrounds which allow the layers underneath to be seen. Unless the highest loaded movie is opaque and equals or exceeds the size of the movie frame, you will see the swfs in lower levels. The movie background color, size, and speed are set by the movie at level 0 (_level0).



On Mouse Events

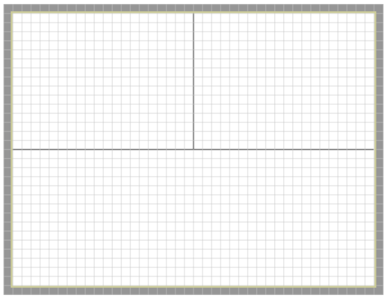
□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

URL bouncy_ball.swf

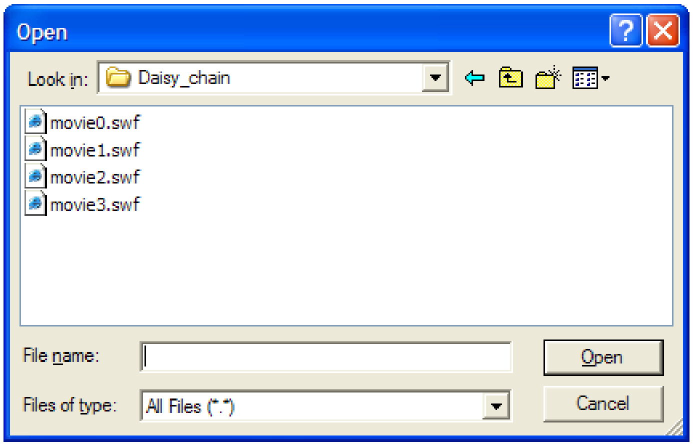
Level 1

The main movie and any swfs loaded into it should be in the same folder. Once a movie is loaded, use the Tell Target commands to control the movies within the different levels. To minimize interruptions, have a control movie on _level0 regulate the loading and playing of the other movie segments; one segment can load while another segment is playing.

Loaded movies are origin point (0, 0) justified, that is, they are lined up based on their top-left corner. In situations where different movies will fill different portions of the browser, it is a good practice to create a template with reference areas marked out as a starting point for the contributing movies. (Grid preferences are set under File > Preferences, Grid options are located under Options > Grid.)



Daisy chaining: A large movie that exceeds the Flash Player's buffer size can be broken into multiple swf files. As each swf ends, its last action is to load the next segment into _level0. Daisy chaining is not typically controlled through buttons, but buttons are often used to test pieces of code and validate the logic of complex actions, either with the Go To Frame or the Load Movie commands. Say you have broken your project into four movies:



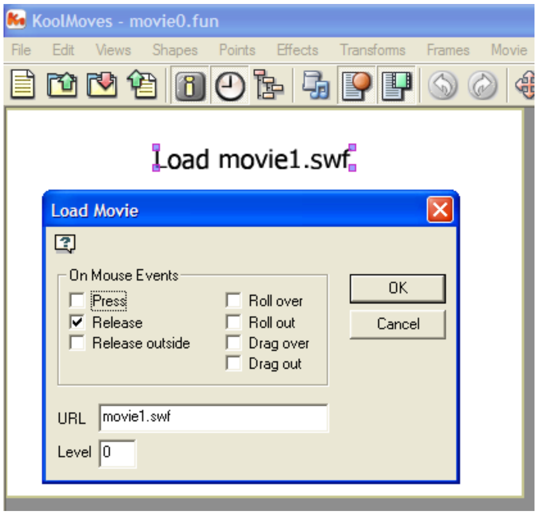
Look in: Daisy_chain

movie0.swf
movie1.swf
movie2.swf
movie3.swf

File name:

Files of type: All Files (*.*)

Each movie has a button on the last frame that opens the next movie (don't forget the stop command on the last frame to keep the movie from looping past the button). The button in movie0.swf loads and launches movie1.swf, and so on. movie3.swf loads and launches movie0.swf and the sequence repeats.

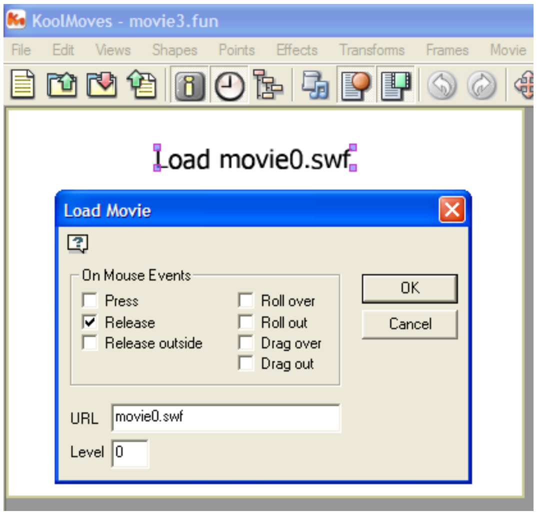


On Mouse Events

☐ Press
☑ Release
☐ Release outside
☐ Roll over
☐ Roll out
☐ Drag over
☐ Drag out

URL movie1.swf

Level 0



On Mouse Events

☐ Press
☑ Release
☐ Release outside
☐ Roll over
☐ Roll out
☐ Drag over
☐ Drag out

URL movie0.swf

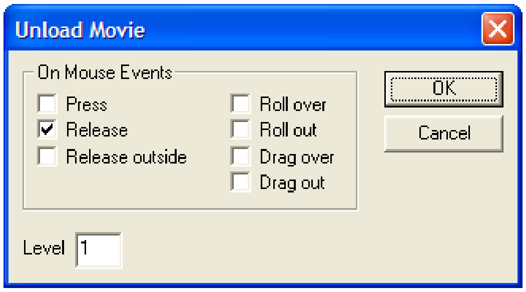
Level 0

After validating each segment of the movie, you can remove the button and stop action on the last frame, and enter the following action on the last frame:

```
this.onEnterFrame = function () {
   this.loadMovie("movieN.swf", "level0");
}
```

Unload Movie

Use Unload movie to remove a movie from a level. Do not unload level 0 (_level0) – this will cause the movie to crash.



Unload Movie

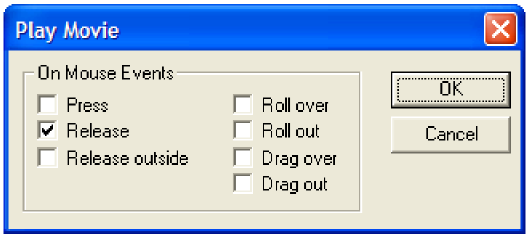
On Mouse Events

□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

Level 1

Play Movie

Use this action to restart the current timeline. If the timeline is already playing this action has no effect.



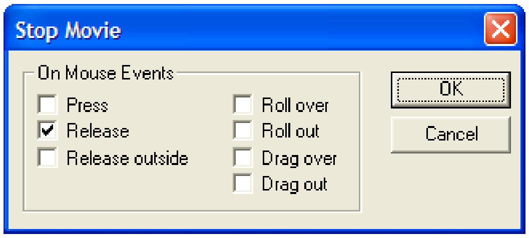
Play Movie

On Mouse Events

□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

Stop Movie

This action stops the playhead at the current frame in the current timeline. The stop command has no impact on sounds that are playing or actions occurring in other timelines (i.e., movie clips in the current level and movies in other levels).



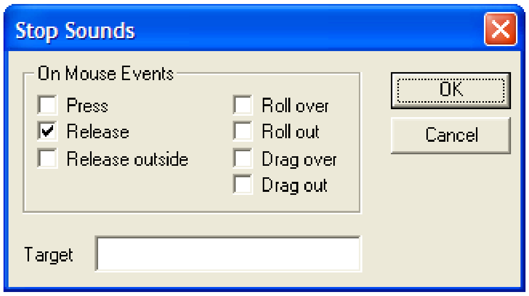
Stop Movie

On Mouse Events

□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

Stop Sounds

This action will stop all sounds currently playing in the target timeline. Leaving the target blank will stop all sounds in the movie. Enter the path to a movie clip or level to stop specific sounds.



Stop Sounds

On Mouse Events

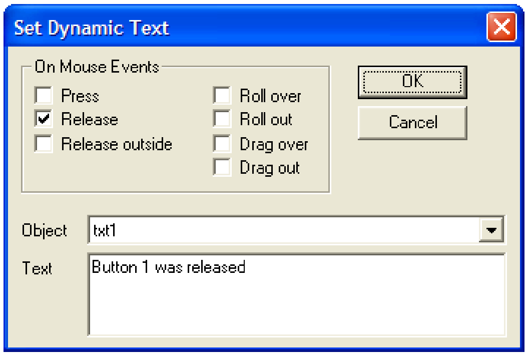
□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

Target

Set Dynamic Text

Use this action to have a button load a string of text into a dynamic text object. If no dynamic text objects exist at author time, the Set Dynamic Text command is grayed out in the list of button actions. Click on the drop down arrow in the Object box to display a list of all the dynamic text objects in the movie.

Enter the message you want displayed in the dynamic text object.



□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

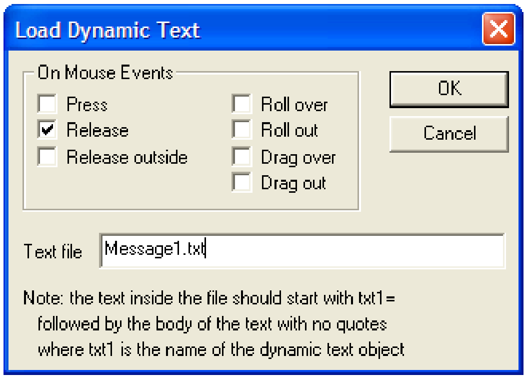
Object: txt1

Text: Button 1 was released

Load Dynamic Text

Use this action to load text from a file into a dynamic text object. If no dynamic text objects exist at author time, the Set Dynamic Text command is grayed out in the list of button actions. Be sure to include the path to the text file.

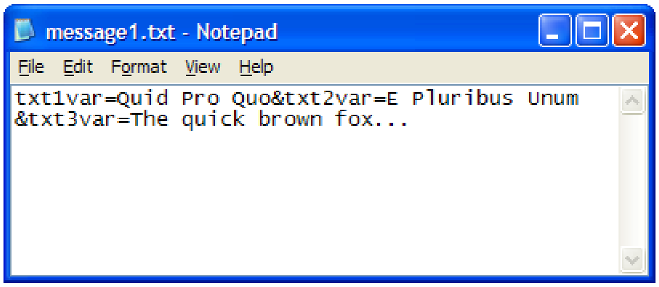
Below is a text file where three strings are assigned to three variables. Variables are used instead of the text object name.



□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

Text file: Message1.txt

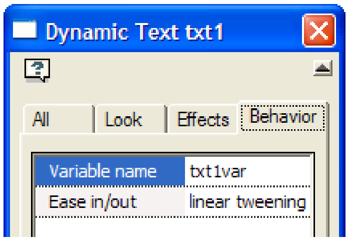
Note: the text inside the file should start with txt1=
followed by the body of the text with no quotes
where txt1 is the name of the dynamic text object



```
txt1var=Quid Pro Quo&txt2var=E Pluribus Unum
&txt3var=The quick brown fox...
```

Note that A) a text object's default variable name is the object's name followed by "var", and B) except for the first one, the variables start with "&".

A dynamic text object's variable name is a property of the object – it can be changed to any valid variable name by opening the Properties data view and editing the value in the Variable name field. You can even change it to the name of the text object (in this case txt1).



Variable name	txt1var
Ease in/out	linear tweening

Send Form Data

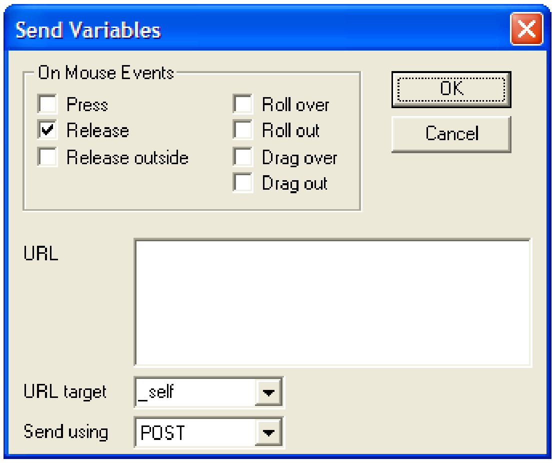
The Send Form Data action is equivalent to the ActionScript getURL(URL, URL target, Method) global function:

- URL specifies the location (absolute or relative) of the external script to run or command to execute.
- URL target identifies the browser window or frame where the command will be executed. Enter the frame name, or select one of the predefined values:
    - _blank – opens file in new browser window
    - _parent – opens file in current browser frame
    - _self – loads the results of a script execution into the current frame
    - _top – replaces all framesets in browser with file
- Method is a literal string (either "GET" or "POST") specifying how the current timeline's variables are sent to an external script.

Compared to the Go To URL command, the URL field is much larger (to accommodate script commands) and the Send Using field has been added.

In the URL target field, type in the target frame name or select a frame/window from the drop down list.

If your form's data field(s) will have chunks of data greater than 255 characters, use the Post method, otherwise either Get or Post can be used.



Send Variables

On Mouse Events

☐ Press
☑ Release
☐ Release outside
☐ Roll over
☐ Roll out
☐ Drag over
☐ Drag out

URL

URL target: _self

Send using: POST

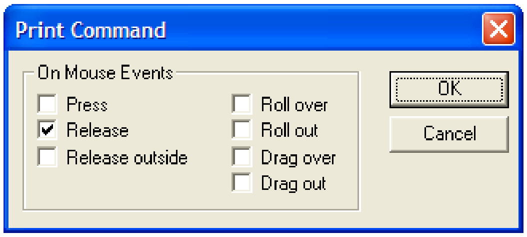
This command sends all variables on the current timeline. In order to send only specific variables, place these variables and the corresponding button action within a movie clip.

The KoolMoves Send Form Data button action supports the following protocols:

Protocol	Format	Purpose
File	file:///path/filename	Access a local file
FTP	ftp://path/filename	FTP a remote file
HTTP	http://path/filename	Open a web page
Javascript	javascript: command	Execute a JS command in a browser
VBscript	vbscript: command	Execute a VB command in a browser
Mailto	mailto:user@domain.com	Send email via the default email app
Telnet	telnet://domain.com:port#/	Open a Telnet connection

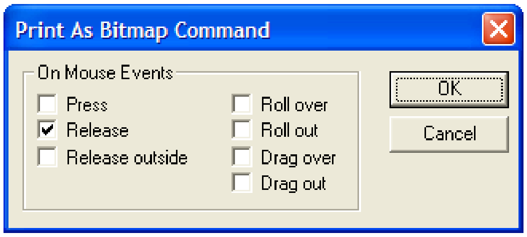
For more information on using forms to submit data, please see the Adobe Technical note How to create and submit a form in Flash 4.

Printing from Within a Movie


Print Command

On Mouse Events

□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out


Print As Bitmap Command

On Mouse Events

□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

The **print()** and **printAsBitmap()** ActionScript commands provide the capability to print frames from within a movie. The print() function will send vector graphics to PostScript printers and raster graphics to non-PostScript printers. printAsBitmap() only sends raster information but supports alpha channels and color transformations. printAsBitmap() is functionally identical to print().

```
print (level)
print (level, "Bounding box")
print ("target")
print ("target", "Bounding box")
```

Options for the Bounding box:
- bmovie (the bounding box of a specific frame)
- bmax (all of the bounding boxes of all the printable frames)
- bframe (the bounding box of each printable frame)

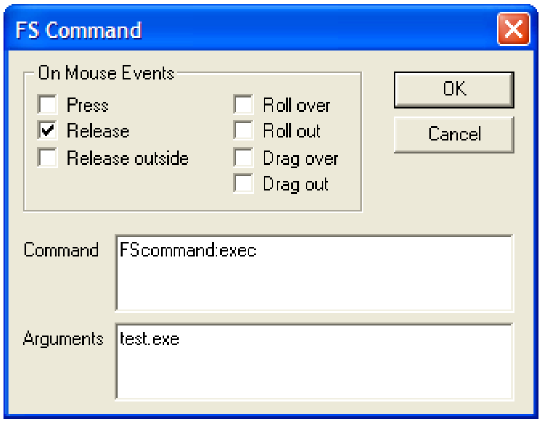
To limit the printing to specific Key Frames, label the key frames to be printed as #P; otherwise all frames (including tweens) will be printed.

FS Command

The FS Command button action is used to send commands to the Flash Player's host application (usually a web browser) and is comparable to the fscommand global function:

fscommand(command, parameters)

Command	Argument	Description
allowscale	true/false	"false" prevents a movie from scaling with changes in the player's window size.
exec	application_name	Launches an external application (include path in the application_name argument).
fullscreen	true/false	"true" causes the player to fill the entire screen.
quit		Stops the movie and closes the player.
showmenu	true/false	"false" suppresses the display of controls in the right-click menu.
trapallkeys	true/false	"true" causes all keystrokes to be sent to the movie – this disables keyboard control of the player window.



□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

Command: FScommand:exec

Arguments: test.exe

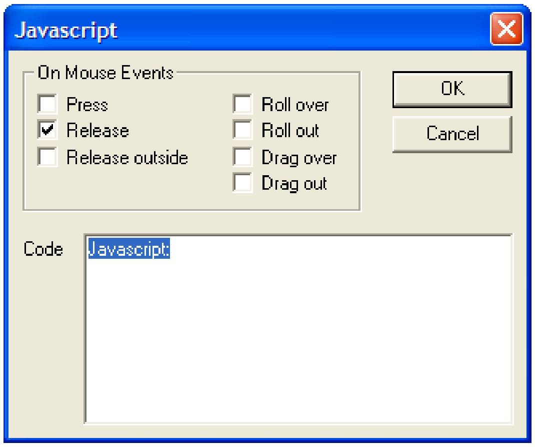
In this example, the movie is actually launching a separate application in the movie's folder.

Additional information can be found in the Adobe ActionScript fscommand dictionary entry, in the KoolMoves manual (Chapter 5 – Data Views > Score / Timeline > Actions & Sounds > FS Command).

JavaScript

Use the JavaScript button action to pass JavaScript commands directly to the host application / browser. Similar to the FS Command window, except any arguments are included in the command line.

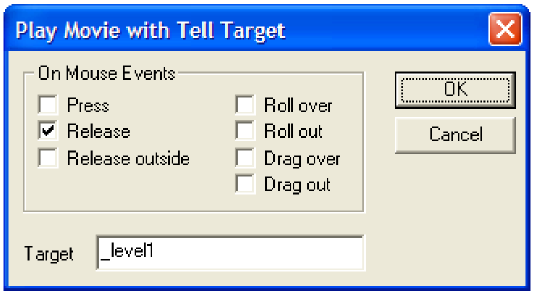
Javascript:***command***



□ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

Code: Javascript:

Tell Target Actions

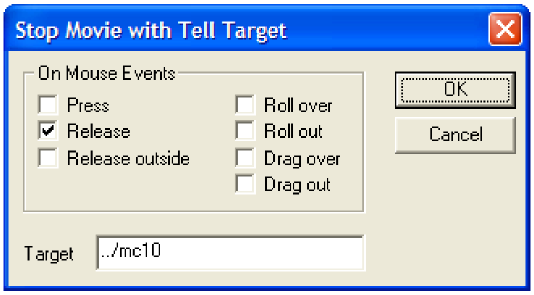


Play Movie with Tell Target

On Mouse Events

☐ Press ☐ Roll over
☑ Release ☐ Roll out
☐ Release outside ☐ Drag over
☐ Drag out

Target: _level1



Stop Movie with Tell Target

On Mouse Events

☐ Press ☐ Roll over
☑ Release ☐ Roll out
☐ Release outside ☐ Drag over
☐ Drag out

Target: ../mc10

Tell Target is a means to communicate timelines without using ActionScript: any movie can control another movie. Your movie has multiple timelines when you…

1. have movie clips,
2. create a SWF Object, or
3. load a movie into a level other than _level0.

There are two basic Tell Target functions built into KoolMoves:

1. Stop (the target timeline)
2. Play (the target timeline)

The tell target actions work just like regular actions with one difference - you must define a target in the target box. Inside a SWF each target is laid out much like a directory. To access the current main movie's timeline leave the Target field blank. Use the "../" syntax to control a higher timeline, or "/path/" to control a lower timeline. In the screenshot to the right above (Stop Movie…), a button in _root/mc1 is controlling _root/mc10.

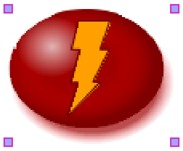
For loaded movies the syntax is a little different. To use Tell Target on a loaded movie you use _leveln/ where n is the level of the loaded movie. If, for example, you want to control a loaded movie in level 1 the target would be: _level1/. To control a SWF Object from the main movie you can use either /Object name or just Object Name.

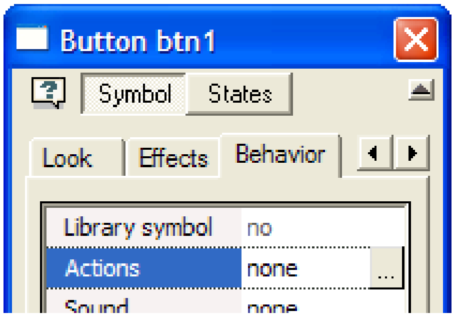
Actions Property > Action Script…

All the actions covered in Actions Property > Dialogs could have been done with ActionScript. However, most new computer animators are in such a rush that they don't have time to learn ActionScript before they make their first Flash movie. That's why the dialog windows in the preceding section were created (KoolMoves was created by a professional animator – he understands the overwhelming desire to create).

As powerful as the dialog commands are, they are limited in scope and flexibility, and should only be used until you have had time to learn ActionScript. In this section you will see how easy it is to make those commands using ActionScript. The first few steps are the same as in Actions Property > Dialogs:

To place code within the button itself, so that the code travels with the button, select a button and open the Properties Data View.

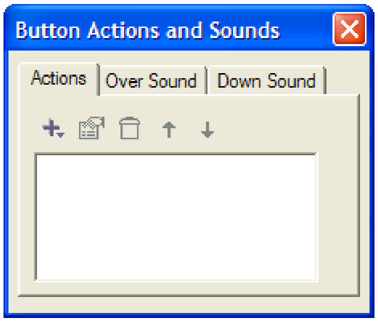




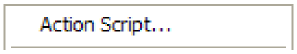
Library symbol	no
Actions	none
Sound	none

In the Behavior properties, click on Actions to open the Button Actions and Sounds window.

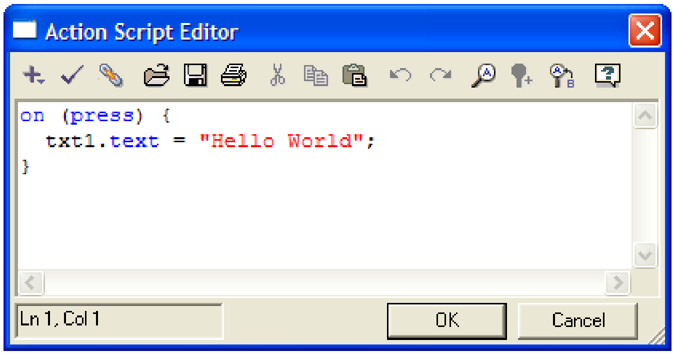
The Actions tab is used to create, edit, delete and organize actions stored in a button.



Click on the plus button with the drop down menu, and select Action Script… from the menu.



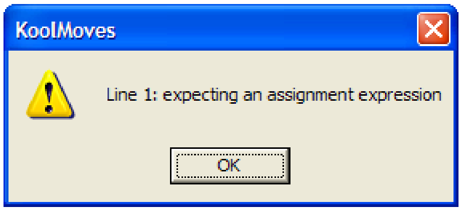
The ActionScript Editor is covered in Chapter 13 of the KoolMoves User Manual. However, for the purposes of this user guide, you don't really need to look up anything. Just remember to click on OK to save your changes, click on Cancel to start over.



```
on (press) {
  txt1.text = "Hello World";
}
```

(The example to the left will work nicely provided you have previously created a dynamic text object named, "txt1". Note that this example, is analogous to the Set Dynamic Text dialog command, and has all the same information: the event to look for, the target dynamic text object's name, and the string to send. Everything else is just a matter of formatting the information for the computer's sake.)

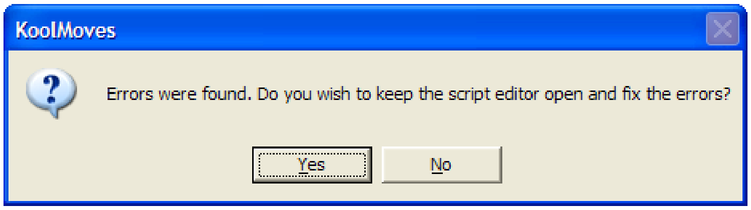
When you click on OK, KoolMoves will check the code for syntax errors. In the following pages there are examples that you can copy and paste into KoolMoves that should work 100% of the time (hey, typographical errors happen). If you get an error message…



KoolMoves

Line 1: expecting an assignment expression

…make a note of the line number and error message. Then click on OK to go back and either fix the problem or save your latest changes.



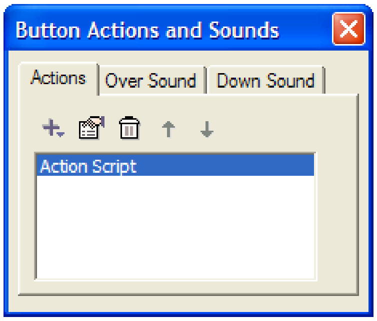
KoolMoves

Errors were found. Do you wish to keep the script editor open and fix the errors?

Clicking on No will save your latest changes, including any error you might have added to your work in progress. To abandon your latest changes click on Yes and then on Cancel in the ActionScript editor (ASE). Okay, enough about the ASE, back to programming buttons. The general format for code placed on a button is:

```
on (event) {
   commands;
}
```

After your code is saved, it will have an entry in the Button Actions and Sounds display area.

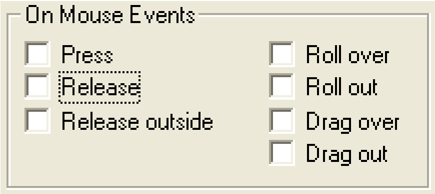


Button Actions and Sounds

Buttons work with user events – actions caused by the user. User events allow you to make your movies interactive. The ability to code actions in buttons was introduced in Flash 5. The available events are the same as in the window dialogs, just capitalized differently and put in parentheses:

(press)
(release)
(releaseOutside)
(rollOver)
(rollOut)
(dragOver)
(dragOut)

equals



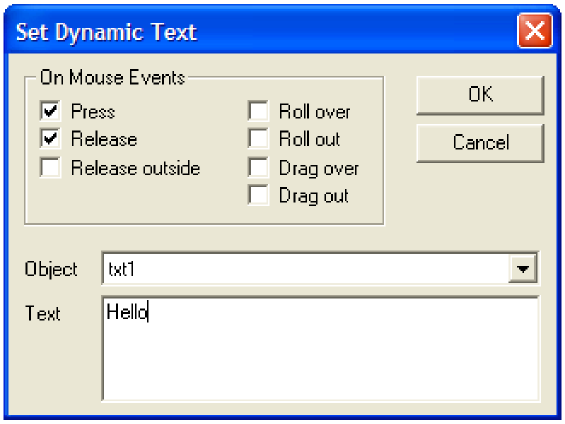
On Mouse Events

☐ Press
☐ Release
☐ Release outside
☐ Roll over
☐ Roll out
☐ Drag over
☐ Drag out

Just like in the dialog actions, we can combine events. In the example below the action is executed when the button is clicked on and when it is released:

```
on (press, release)
   txt1.text = “Hello”;
}
```

equals



☑ Press
☑ Release
□ Release outside
□ Roll over
□ Roll out
□ Drag over
□ Drag out

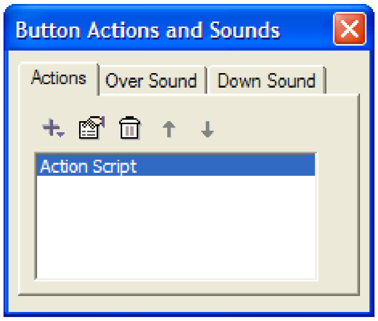
Object: txt1

Text: Hello

To have multiple actions tied to different events on the same button, we can write additional event handlers in the same block of code:

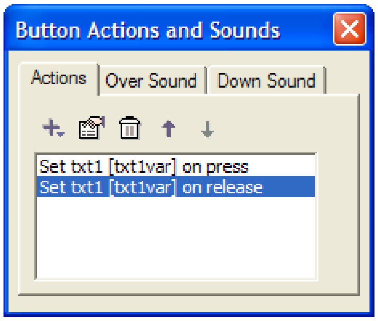
```
on (press) {
   txt1.text = “Hello”;
}

on (release) {
   txt1.text = “Goodbye”;
}
```



Action Script

versus



Set txt1 [txt1var] on press
Set txt1 [txt1var] on release

Button Code Examples

Let’s look at some typical button actions done with ActionScript on the button. We are assuming that the buttons with the following actions are on the main timeline (_level0 of _root), and that the action should occur on (release). In many of the example below there is more than one way to achieve the desired goal, in which case the simplest example was selected.

Opening the KoolMoves website in the current browser window:

```
on (release) {
   getURL(“http://www.koolmoves.com/”);
}
```

Opening the KoolMoves support page in a new browser window:

```
on (release) {
   getURL(“http://www.koolmoves.com/support.html”, “_blank”);
}
```

Jumping the playhead to key frame [number ***n*** / labeled ***exitSequence***] and playing:

```
on (release) {                                   on (release) {
   gotoAndPlay(n);                                  gotoAndPlay(“exitSequence”);
}                                                }
```

Loading ***test1.swf*** into level 1 (can also use this to load image files):

```
on (release) {
   loadMovieNum(“test1.swf”, 1);
}
```

Unloading the movie currently in level 1:

```
on (release) {
   loadMovieNum(“test1.swf”, 1);
}
```

Restarting the current timeline:

```
on (release) {
   play();
}
```

Stopping the current timeline:

```
on (release) {
   stop();
}
```

Stopping all sounds currently playing in the movie:

```
on (release) {
   stopAllSounds();
}
```

Setting / replacing the text in dynamic text object txt1:

```
on (release) {
   txt1.text = “Hello World”;
}
```

Adding text to the text currently in a dynamic text object txt1:

```
on (release) {
   txt1.text += “Hello World”;
}
```

Loading text from message1.txt into dynamic text object txt1:

```
on (release) {
   loadVariables(“message1.txt”, “_root”);
}
```

Printing selected key frames of the movie (where selected frames are named “#p”):

```
on (release) {
   print(“_root”, “bframe”);
}
```

Printing a movieclip with printAsBitmap:

```
var pj = new PrintJob();

// open printer dialog
if (pj['start']()){

   // fit mc1 to page (not required)
   var scale = Math.min(pj.pageWidth / mc1._width, pj.pageHeight / mc1._height);
   mc1._xscale = mc1._yscale = scale * 100;

   // add page
   if (pj.addPage(mc1, null, {printAsBitmap:true})){

      // process all pages sent to the print spooler
      pj.send();

   }
}
```

Setting the movie to scale with the host application using fscommand():

```
on (release) {
fscommand(“allowscale”, “true”);
}
```

Sending a JavaScript command to the host application:

```
on (release) {
   getURL(“javascript:command”);
}
```

Stopping the movie in level 1:

```
on (release) {
   _level1.stop();
}
```

Stopping movie clip mc3 in mc1:

```
on (release) {
   mc1.mc3.stop();
}
```

Timeline Button ActionScript

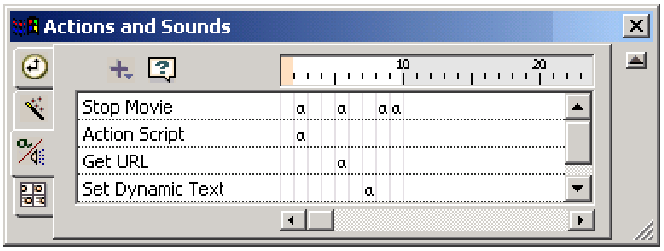
As putting ActionScript on buttons is not much more complicated than working with the window dialogs, putting button actions on the timeline is only slightly more complicated than putting the actions directly on the button, and has its own advantages. While Flash allows you to place code on your buttons and other objects, as your movies become more complicated, and your knowledge of ActionScript increases, it makes sense to place all you code in one place as opposed to searching the length of your movie for the piece of code that needs to be debugged.

Whether you use code directly on the button, or place the button code within the timeline is up to you. In situations where the name of the button is being changed a lot, or the button is copied between project files, it may make sense for the intermediate level ActionScript programmer to embed the button code in the button itself.

We are interested in adding ActionScript that will define the actions for a button. The big difference between this and placing the code directly on the button (accomplished in the preceding section) is that we will need to identify the button in the action code.

To place code on the timeline, open the Score/Timeline data view and click on the Actions and Sounds Tab.

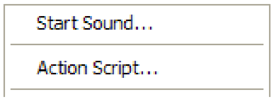
Many actions have been placed directly on the timeline in this movie, and will execute when the specific Key Frame loads in the Flash Player. Note that there are no actions on Key Frame 1.

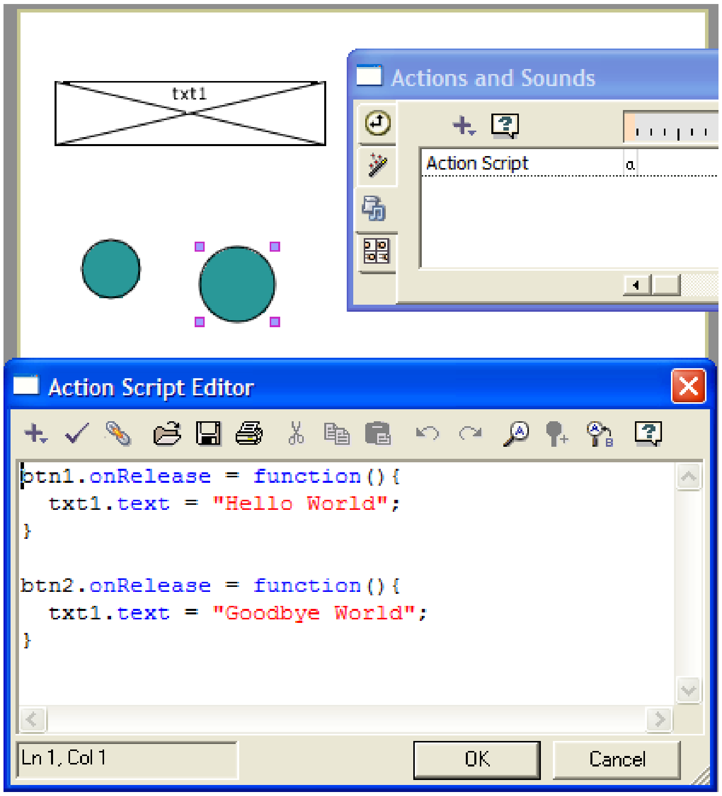


It is a good practice when writing code for a multi frame movie to leave key frame 1 blank with 0 tweens, and to place all your code on key frame 2. In the Score/Timeline data view displayed above, click on the key frame 2 in the frame navigation bar before adding the button action to the timeline.

However, our first example is a single frame movie, so the above caveat is not applicable.

Next, click on the plus button with the dropdown arrow, and select Action Script… This opens the ActionScript Editor (covered in Chapter 13 – Action Scripting, of the KoolMoves User Manual). In the example below, the code assigns actions to two buttons to change the message in a dynamic text object.





```
btn1.onRelease = function(){
   txt1.text = "Hello World";
}

btn2.onRelease = function(){
   txt1.text = "Goodbye World";
}
```

One of the buttons in the example to the left will cause "Hello World" to fill dynamic text object txt1 when the mouse click is released. The other button will fill the same box with "Goodbye World".

(Low on originality, but this example illustrates the general programming environment, relevant objects on the stage and applicable data view. This is a single frame movie, so the code is on key frame 1.)

The following is the general format for creating a button event handler:

> *buttonObject*.on***Event*** = function() {
>   commands;
> }

Buttons work with user events – actions caused by the user. User events allow you to make your movies interactive. The ability to code button actions on a timeline was introduced in Flash 6. The available event handlers are:

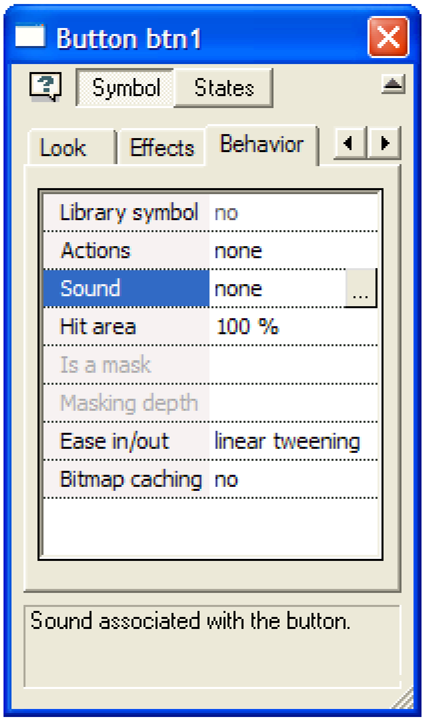
onPress()
onRelease()
onDragOver
onDragOut()

onReleaseOutside()
onRollOver()
onRollOut()
onSetFocus(***oldFocus***)
onKillFocus()

There are two new events – onSetFocus and onKillFocus. A button has "focus" when it has either been navigated to with the Tab button, or focus has been assigned programmatically with the Selection.setFocus() command. Actions tied to onSetFocus will occur when the button receives focus, while onKillFocus actions occur when the button loses focus.

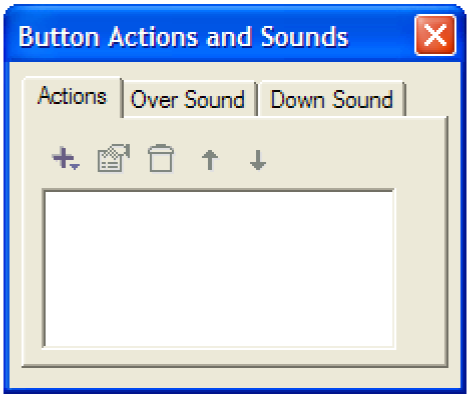
Adding Sound to Button States

Use a button's Sound property to open the Button Actions and Sounds window.



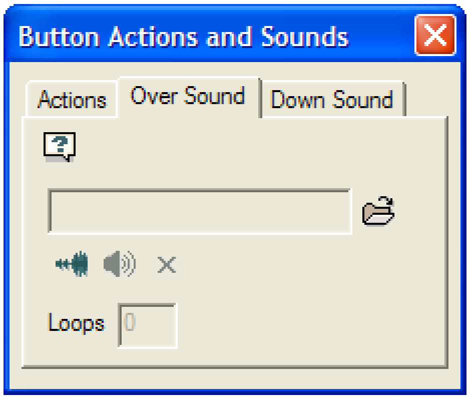
Library symbol	no
Actions	none
Sound	none
Hit area	100 %
Is a mask	
Masking depth	
Ease in/out	linear tweening
Bitmap caching	no

Sound associated with the button.



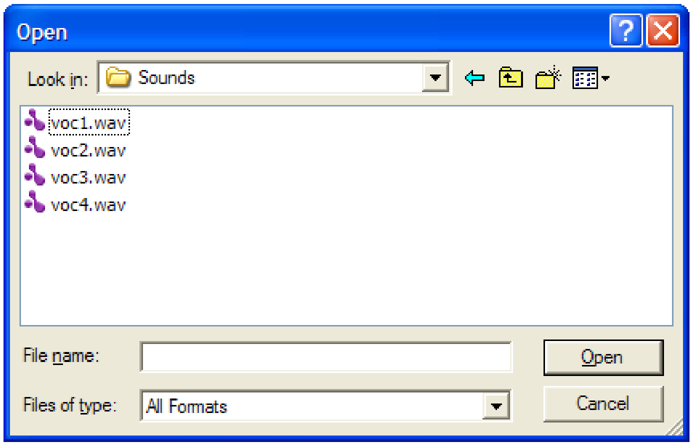
Click on the Over Sound / Down Sound tab to set the sound file for the respective state.

Let's look at adding a sound to a button's Over state.



Loops 0

Click on the folder icon to open an explorer window, and select the desired sound file.



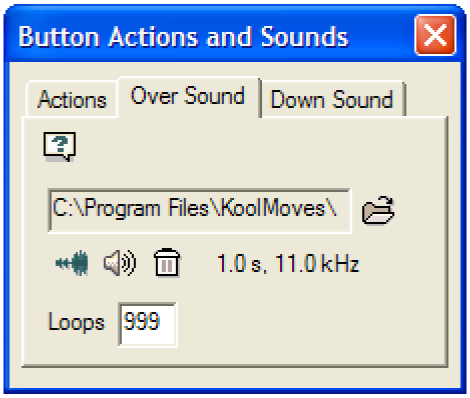
Look in: Sounds

voc1.wav
voc2.wav
voc3.wav
voc4.wav

File name:

Files of type: All Formats

The sound files duration and sampling rate are displayed. Set Loops to 999 for continuous play.



C:\Program Files\KoolMoves\

1.0 s, 11.0 kHz

Loops 999

Edit the sound using a sound editor which has been assigned in Preferences

Play the sound

✕ Remove the sound

The Over state sound is activated for roll over, roll out, drag over, and drag out mouse events. The Down state is activated for press and release mouse events.

At this time, KoolMoves only supports the PCM wav and the mp3 file types. Button sounds do not play when testing your project in KoolMoves' internal player.

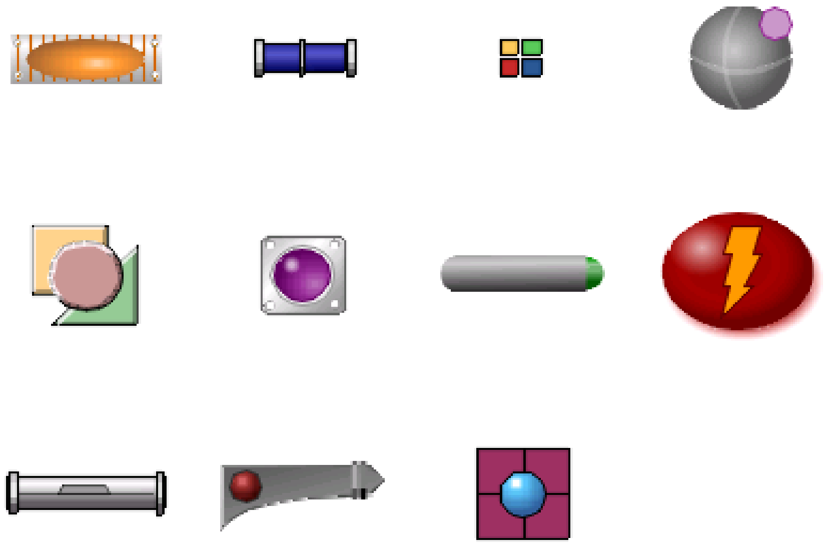
Sound files tend to be huge which can make the exported swf file huge. To conserve resources, take these steps: in a sound editing program convert from stereo to mono and to the smallest sample rate that gives you the quality you need. Shortening the sound track also helps.

If a sound plays longer than the movie plays and if the movie loops, the sound from the first loop will continue playing during the second loop until the sound from the first loop finishes. This effect worsens as the movie continues to loop. To solve this problem, add the Stop Sound action to the last frame in the movie.

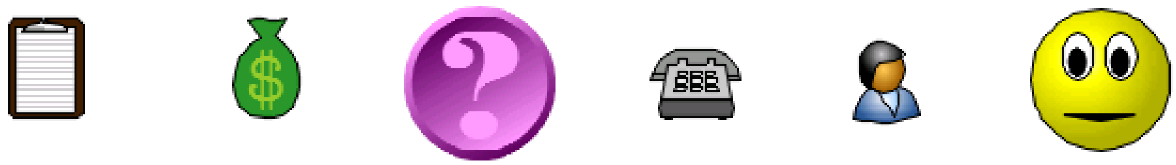
If your sound file is sampled at a rate other than 5500, 11025, 22050, or 44100, the sound will play in the Flash player at a rate nearest to one of these four rates.

Button Gallery Categories (Folders)

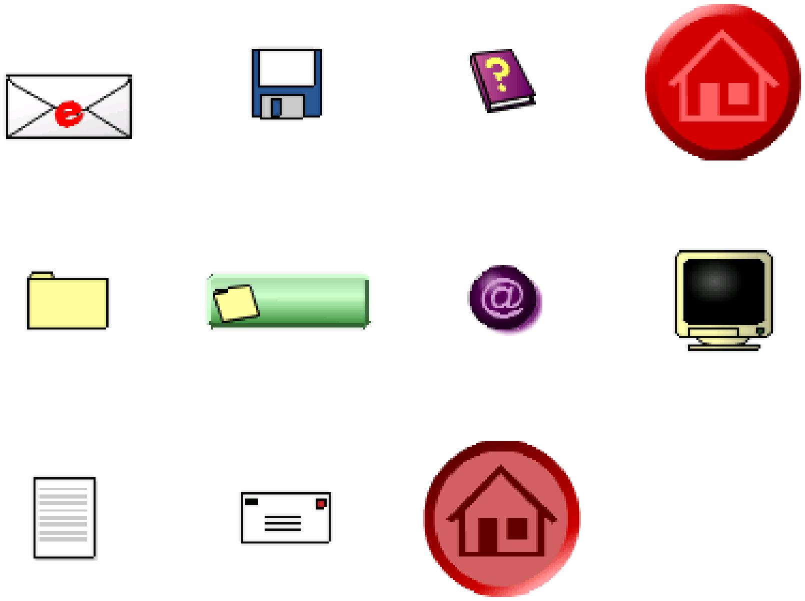
Abstract



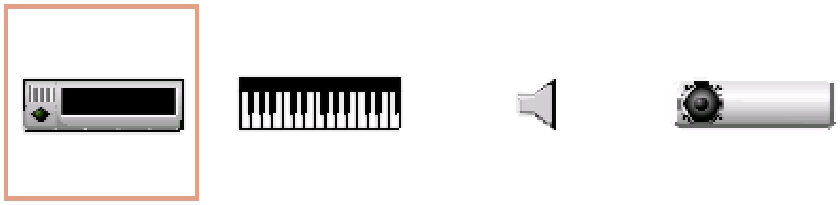
Business & Communication & People



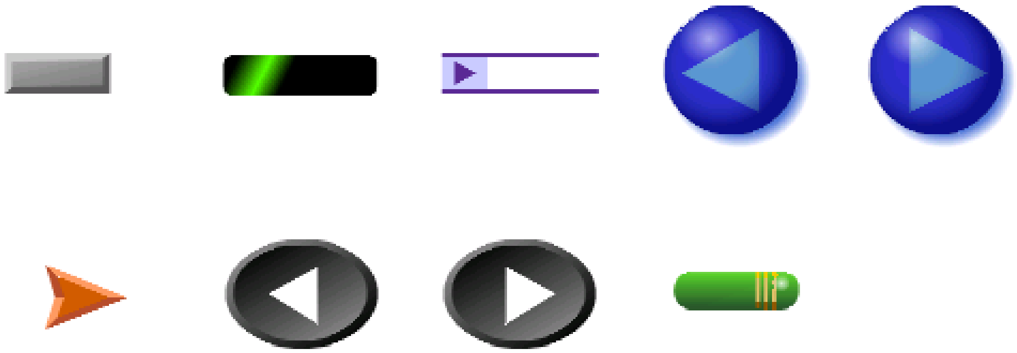
Email & Internet



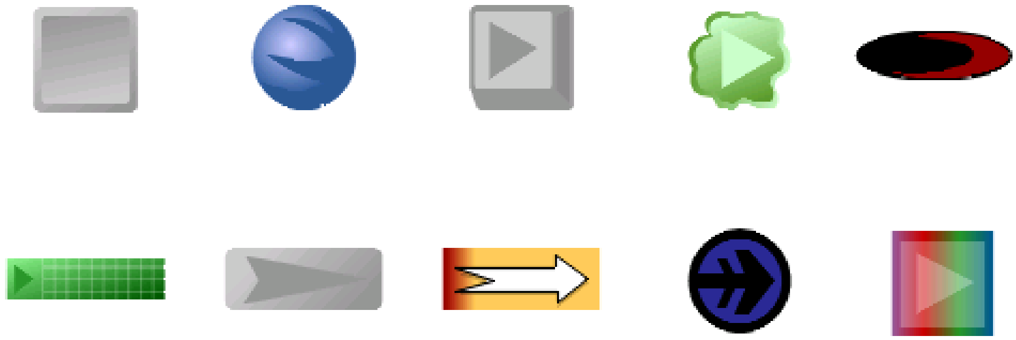
Music



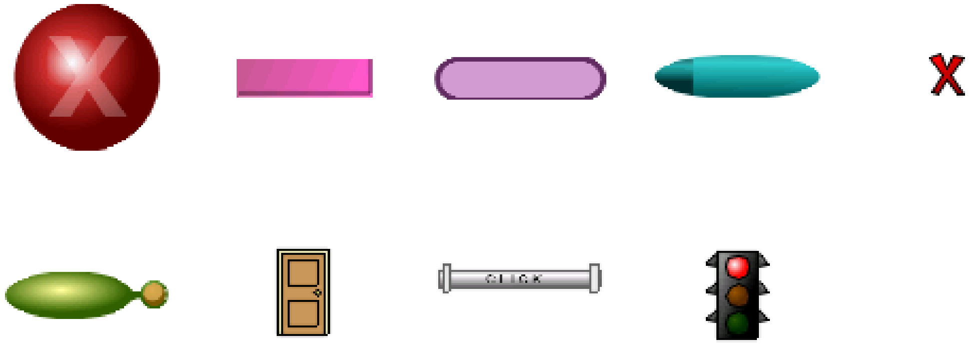
Navigation 1



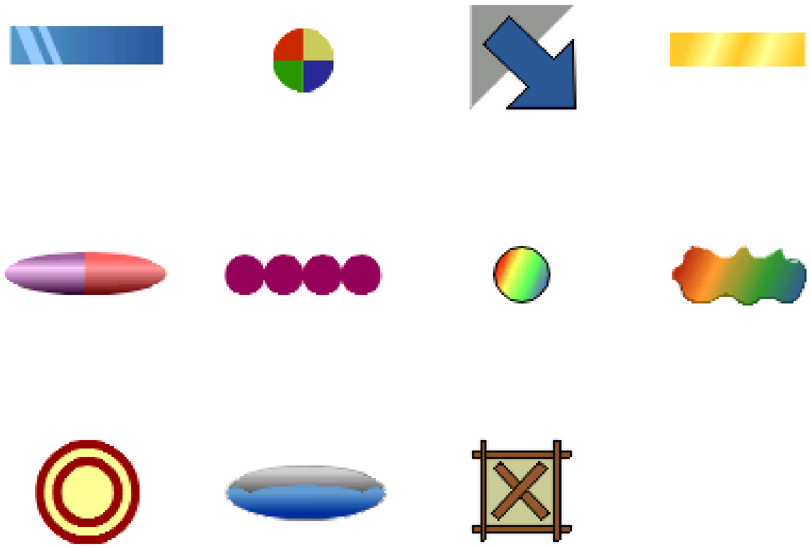
Navigation 2



Odds & Ends 1



Odds & Ends 2



Movie Clips as Buttons

As you've found out by now, buttons are easy to work with and very capable in making movies interactive. However, as you become more proficient with ActionScript, you might consider using movie clips. Movie clips have all the capabilities of buttons but a more generic object design, allowing for greater flexibility. It is beyond the scope of this User Guide to cover movie clips, but I would like to mention why advanced Flash programmers use movie clips in place of buttons.

Dynamic Interactivity
Movie clips can be both created and removed via code at runtime. This allows new levels of flexibility and interactivity as well as conserving system resources.

Additional Event Handlers
In addition to mouse actions occurring to the movie clip's presence on screen, the clip can handle additional mouse, keyboard and system events not directly tied to the clips presence on the stage.

Additional States
Because movie clips don't come with states, a clip's state consists of frames in the clip's timeline, with the playing of these frames tied to one or more events. A movie clip acting as a button can have a different state for each event – you are not limited to just Up, Over and Down.

There are many threads with examples in the KoolMoves support forum concerning the use of movie clips as buttons, and there is at least one sample at www.koolexchange.com.

Summary

Buttons are symbols. The Symbol Library can be used to store, edit, and track buttons.

Buttons can have three states: Up, Over and Down. Each state can be scaled/edited separately.

There are three types of copied buttons based on how they relate to their parent button: linked states and behaviors, linked states, and unlinked complete copies.

In KoolMoves, almost any visual element can be turned into a button. Bitmap images can also be imported into button states, but become blurry when rescaled.

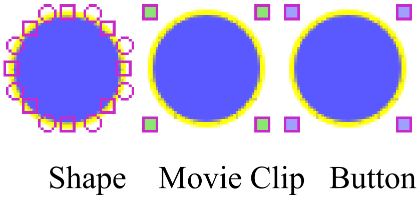
Every effect in KoolMoves can be applied to buttons or button states, one way or another.

A) Some effects can be applied directly to the button, affecting all button states;
B) Effects that can't be applied to a shape within a button state, can be applied to a shape within a movie clip within a button state; and
C) Effects applied to a movie clip which is then placed in a button state, cannot be edited within the button state. However, effects on a movie clip within a movie clip within a button state can be edited.

Movie clips inside button states can contain animations, swf files, and Flash Live Video files.

Items can be copied elsewhere and pasted into a button state, or copied in a button state and pasted elsewhere.

Different items display different border boxes when selected. If you don't know what you are working with, double click on the item to open the Properties data view.



Registered versions of KoolMoves come with buttons ready to use in the Button Gallery.

Buttons can be stored in project files, either in Button Gallery folders or in Symbol Libraries.

Button actions can be placed in a button's Actions property, or applied dynamically via ActionScript on the timeline containing the button.

Sound files can be assigned to a button's Over and Down states.

Buttons made out of movie clips can handle additional events, are more flexible/customizable, and can be created and removed dynamically.

Beware the Bleed Through effect.