

Animating the Web

Phillip Kerman

Supplements to this presentation at: www.phillipkerman.com/wb01

Overview_____

Essentials

- Stage & Timeline

- Drawing on the stage

- Uniqueness (strokes vs. fills; "what you see is what you can select"; natural stacking)

- Helpers (snap; grids and guides; tolerances)--options

- Managing media

- Storing Symbols in the Library

- Nesting Symbols

- Animating in the Timeline

- Keyframes

- Tweening (motion vs. shape)

- Layers (to separate animations; to mask/reveal other layers)

- Guides vs. Motion Guides

Beyond the Basics

- Age-old animation techniques (anticipation; overkill; subtle touches)

- Animating text

- Incorporating external media

- Vector types

- Raster graphics (a.k.a. bitmaps)

- Audio

- Nested animations using Movie Clips

- Making buttons to click (including animated buttons)

- Using sounds in the timeline (Event vs. Stream)

- Publishing your creation

User interfaces using ActionScript

- Basic scripting (writing instructions; determining how to trigger them)

- Placing scripts (in Keyframes; on Button instances; on Movie Clip instances)

- ActionScript commands (stop(); gotoAndPlay(); loadMovie();)

- Changing properties (and "homemade properties")

- Interactivity

- Dragging

- Text input and dynamic text.

- Complete examples

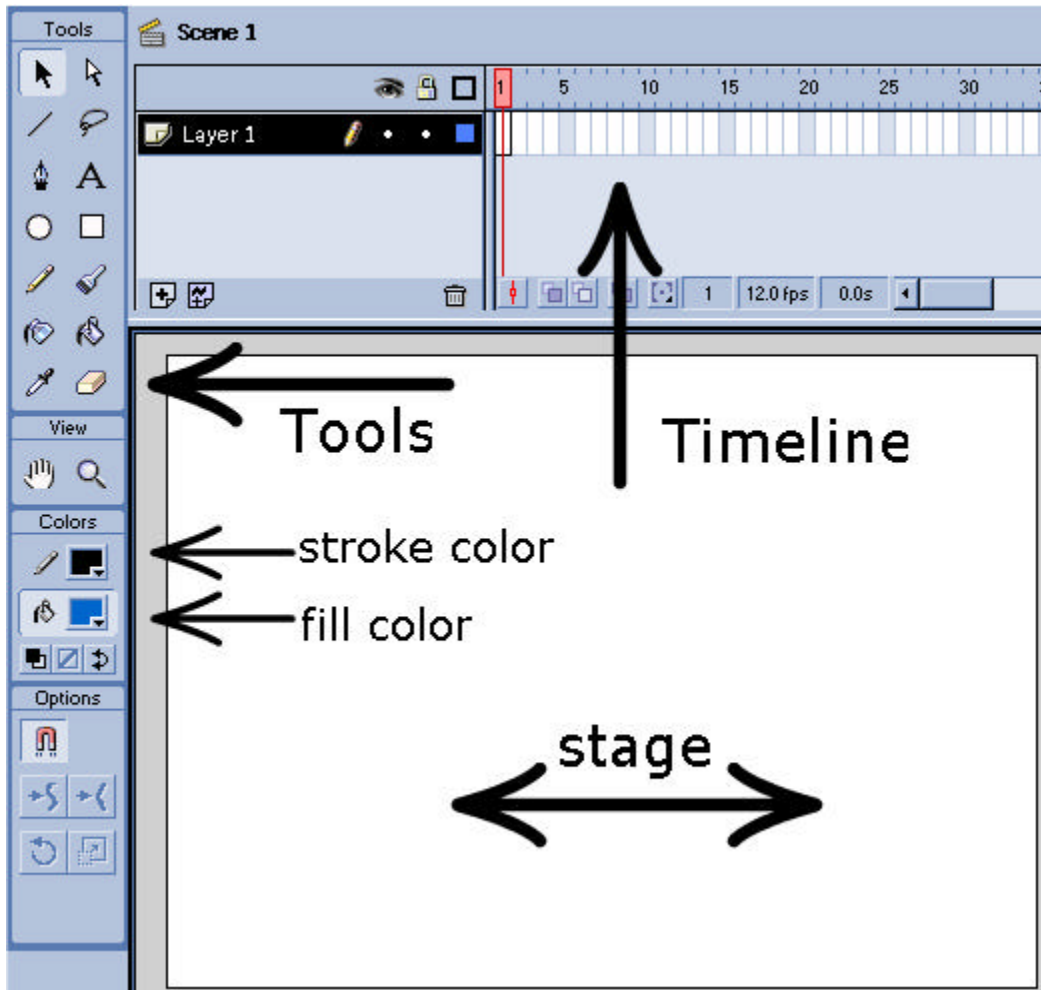
- Menus

- Scripted "frameless" animation

- Preloader

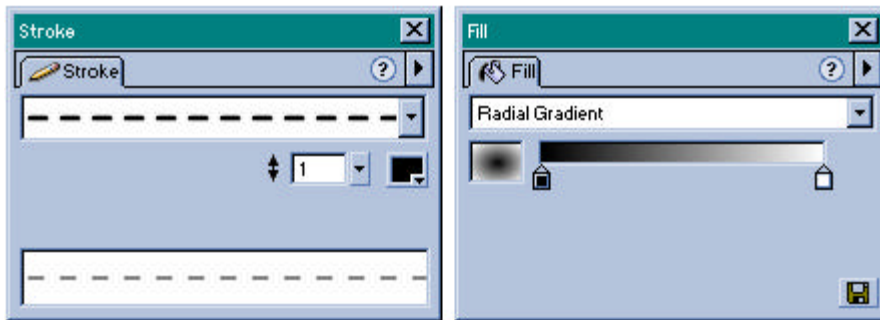
Hour 1: Essentials

Stage & Timeline. The stage is your work area where all visual elements are placed for your user to see. Tip: select the menu "View>Work Area" so you can move things off stage (and, later, animate them onto the stage). We'll use the timeline to orchestrate the animation and other visual effects. Since the timeline is dockable and can be closed, remember the menu "View>Timeline" will restore it. Next we'll look at drawing on the stage. Just realize you can give focus to either the stage or the timeline—they are distinct edit areas.



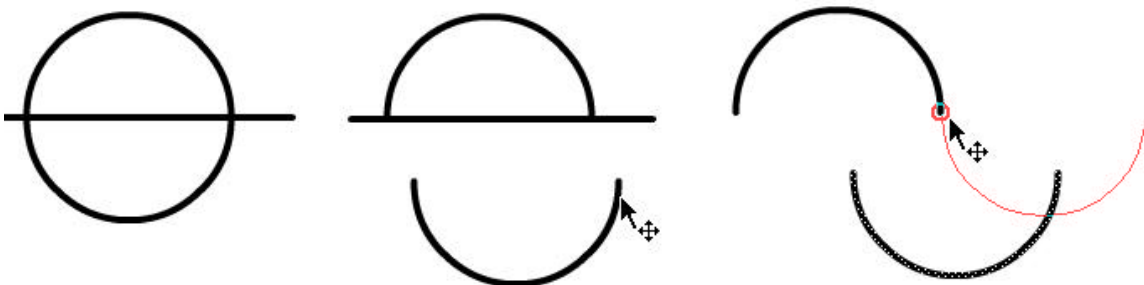
Drawing on the stage. While some standard tools exist many have unique characteristics that you'll likely hate at first—but surely love when you fully understand the "Flash way". With the exception of text and Library Symbols (discussed below), everything you draw on stage is called a shape. Shapes can have both a stroke and fill—think of m&m candies... the stroke is like the candy shell and they're filled with chocolate. Fills are created with the brush tool, or (with enclosed shapes) filled-in using the paint bucket tool. Strokes are created with the line tool and the pencil tool. Additionally, strokes can be added (or changed) on any fill with the ink bottle tool (notice the ink bottle is under the pencil). The oval tool and rectangle tool create shapes with both a stroke and fill. Notice the two color swatches (one for fill and one for stroke).

Strokes are just like the definition of a theoretical line (the distance between two points) and as such they have no width. While you can modify a stroke's current thickness (and line pattern for that matter) using the stroke panel, strokes simply don't have a left side and right side—just two endpoints. Fills are like another material type. Fills and strokes are just different... gradient colors can only be added to fills and only strokes get line patterns—to name two differences.



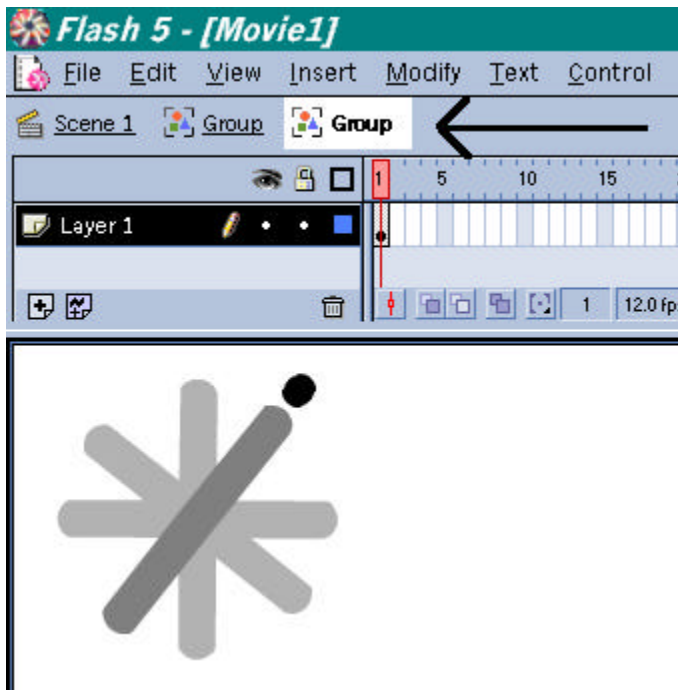
The stroke panel (left) and fill panel.

A key competency in Photoshop is learning how to select. While Flash has a magic wand tool (an option under the lasso tool) I can safely say I've never used it. Flash has a unique way of selecting shapes summed up by the phrase "if you can see it, you can select it". First, realize Flash has only one drawing surface (forget layers for now)—everything you draw is on at the same ("canvas") level. If you draw a circle on top of a line, the line has been eaten away. While this is frustrating at first, it gives you the power to make sophisticated selections. For example, you can draw a diagonal line bisecting a square to make a triangle. Again, anything you can see you can select meaning you simply need to separate the intended selection visually and then just click with the arrow tool.



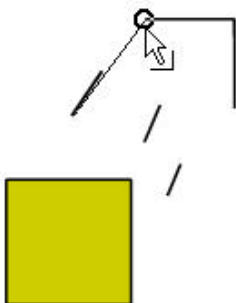
A sequence of selecting visually identifiable elements.

By the way, you can prevent shapes from "eating away" other objects by grouping the shape you want isolated. Grouped shapes can be stacked (and arranged using options like "send to back" from the Modify menu). To edit a grouped shape you first double-click on it. This technique is better than temporarily un-grouping a shape because you won't risk the "eating away" aspect (but don't forget the "eating away" issue can be used to your advantage too). The most important detail to notice when you edit a grouped shape is that the address bar shows you the hierarchy of nested shapes. (Incidentally when you double-click a grouped shape all the other shapes dim out, but the address bar is the best way to really know where you are editing.)



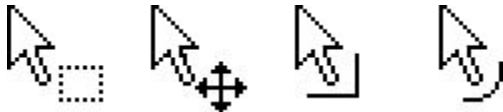
The hierarchy of nested groups are shown in the address bar.

One of Flash's greatest powers is the way it can help you draw. You'll find most of these helpers in the options section of the tool palette. Realize that as there are a ton of such helpers just the options that apply to the currently selected tool will appear. When the arrow tool is selected you can turn on the Snap to Objects option (actually, I'd say leave it on always unless you need to temporarily turn it off). If you then select a shape from a "logical" location (like the corner, center, or middle edge of a square for example) a dark circle will appear indicating you can snap this logical location to another shape or to an invisible guide. This method is far superior to connecting shapes visually.



The dark ring at the point of the cursor indicates something will snap (the line's endpoint in this figure).

Before you go wild snapping objects you should notice how your cursor is constantly changing to indicate what will happen if you click. That is, when you get close to the end point of a line, the cursor changes to indicate that *if you click* you'll be extending the end point. If you approach the mid-section of a line the cursor indicates that *if you click* you'll be bending the line. If you first click to select a line, then notice the cursor indicates that *if you click* you'll move the entire line. There are so many different cursor indications I can't list them all here (I don't even think I got to them all in my book!). Anyway, it's worth paying attention to the cursor change because it's trying to tell you something.

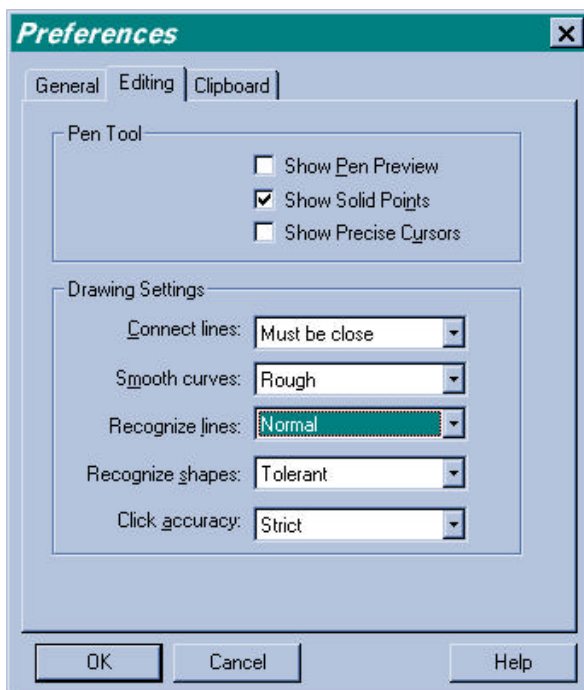


Cursor changes that indicate what will happen when you click (select, move, extend endpoint, bend midpoint).



The eyedropper cursor also changes to indicate what you're about to sample (a stroke, a fill, a text color).

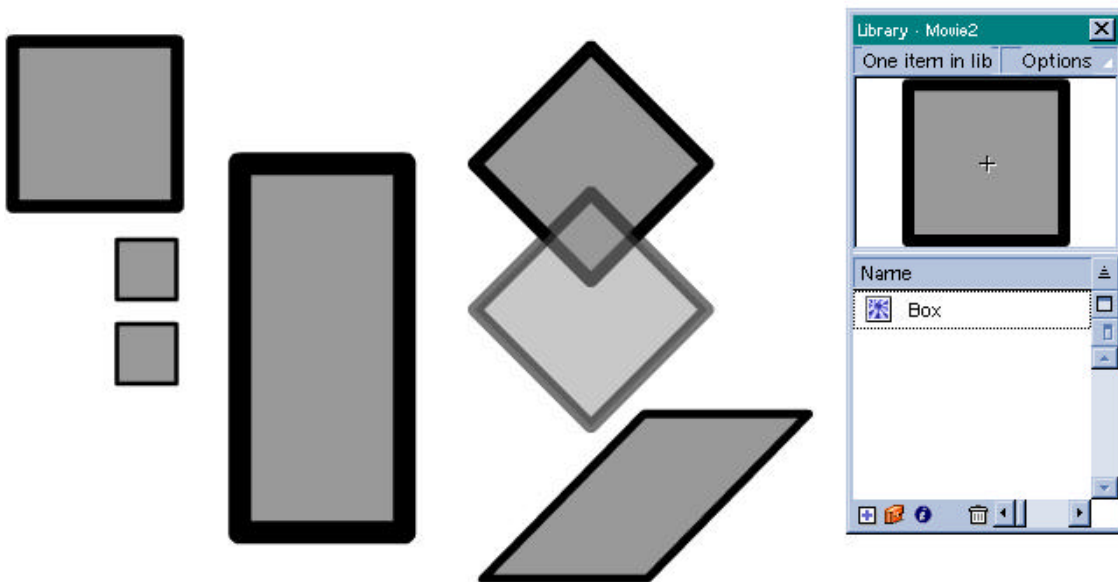
Although it's nice that Flash is trying to help you draw, it can be too much sometimes. Luckily, the sensitivity settings can be modified. Most settings appear in the options section under each tool. For example, under the paint bucket fill tool you have options like "don't fill gaps", "fill small gaps", "fill large gaps" etc. In addition a large set of tolerances can be adjusted under "Edit>Preferences>Editing".



Additional tolerances are set via Edit>Preferences.

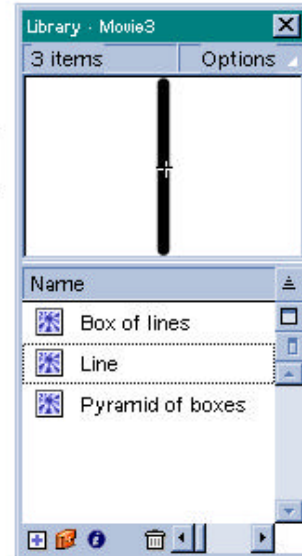
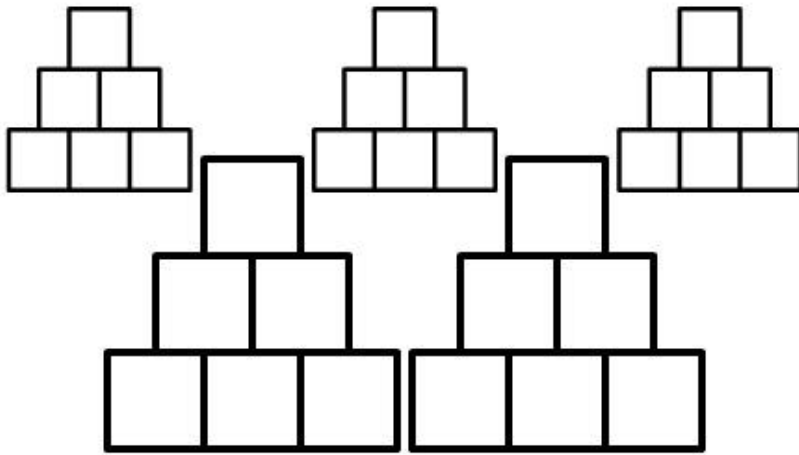
This presentation moves past the drawing features here, but please note: Flash drawing is very powerful. It's not as elaborate as Illustrator or Freehand—but that's good for several reasons. For one, it means I don't have to learn a million features I'll never use... plus, it tends to keep your Flash movies smaller because the graphics are less complex. You really should check out how you can draw in Flash—for example the interface for modifying gradients is super intuitive. Personally, I consider Flash my primary drawing tool.

Managing media. Just remember this, any time you're about to copy and paste *anything* stop and ask yourself if there's a better way. Any duplicates add to your filesize. However, Flash has a mechanism where you first save a shape (called a Symbol) that's stored in the Library. Then, you can drag *instances* of the symbol onstage effectively making duplicates—but they're not significantly adding to the file size as they're just aliases of the original. What's really cool is you can vary several characteristics of each instance to make them appear unique. You can vary their location, scale, rotation, and color effect (like tint and transparency). (These attributes are called properties.)



One master "Box" symbol can create lots of varied instances onstage.

Now, at the risk of causing heads to explode, realize that you can nest symbol instances inside of other symbols. For example, you could make a line symbol and use 4 instances (each rotated and positioned differently) to make a square shape. Then, you could make a symbol from those four instances of the line symbol. Then, use six instances of the square to make a pyramid... store the pyramid as a symbol... and then they'll tell two friends... and so on. The two general benefits to nested symbols are smaller file sizes (realize 100 pyramids are all based on just one line) and easier graphic management (a change to the line will propagate through all child symbols).



This file only has one shape—a line. But the "Line" symbol is used in the creation of the "Box of lines" symbol which, in turn, is used to create "Pyramid of boxes".

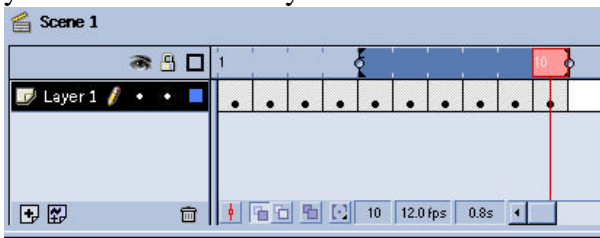
The "benefits" of nested symbols can actually work against you when you take things to an extreme. While extra instances don't add appreciatively to your filesize, each instance's properties are saved (to track how they vary). This usually isn't much, but it can add up—sometimes to larger files than just making separate shapes. Regarding the graphic management benefits, making one little change to, say, the line will cause everything based on that line to change. Maybe you want a dashed line in just one pyramid. Well, a change in one affects all. (By using "Modify>Break Apart" or the Library's "Duplicate Symbol..." options you can solve any problem.)

Animating in the Timeline. A simple definition of animation is pictures that change over time. Think of a flip book where a different drawing appears on each page. When you flip the pages you see an animation. Flash's timeline lets you draw different pictures in separate frames which get presented to the user. You'll notice Flash's default frame rate is 12 fps (frames per second). While in theory a higher frame rate will mean a faster animation—unfortunately your user's computer may not be able to display the contents of each frame that fast... and Flash simply slows down to display each frame. Luckily, you'll learn better ways to control the apparent speed of an animation—so, leave the frame rate somewhere near 12 fps (you can set it via "Modify>Movie...").

When you start, you're only given one frame in which to draw. You can make your animation last longer by adding frames. (A total of, say, 12 frames—at 12 fps—means your animation will last 1 second.) You have a choice between adding "frames" and adding "keyframes". Only keyframes let you change what appears onstage (the first frame is a keyframe by default). Say in frame 1 you draw a circle on the left side of the screen, then add a keyframe at frame 2 and draw a bigger circle. When played in sequence, you'll see the circle grow. You can continue to add keyframes with different appearing shapes. The point is, that only in keyframes are you given the chance to draw something different. You may wonder why you'd ever just want to add a frame (instead of keyframe). Simple: if you want something to appear (say in frame 1's keyframe) and then not change until after frame 12 you'd want to add only frames up through frame 12. A keyframe says "be here at this moment in time". More frames just say... "stay as you are in your most recent keyframe". Another keyframe says "now, be here at this moment in time". Quite often you won't need keyframes in every frame.

In addition to "frames" and "keyframes", there are "blank keyframes". They're nothing more than a keyframe which has no graphic content onstage. The thing is, when you select Insert>Keyframe what Flash does is it makes a new keyframe but copies the contents in the previous keyframe (making it easy to make a minor adjustment to it—maybe you just move the graphic slightly). If you delete the onscreen contents in a keyframe you're left with a blank keyframe. If you need a new keyframe but only expect to change the onscreen contents slightly, insert a keyframe. If you're going to have an entirely different graphic onscreen in the new keyframe, just insert a blank keyframe. It's really not terribly complex, just realize that "insert keyframe" not only inserts a keyframe, but also copies whatever contents are in the previous keyframe.

Frame-by-frame animation can be grueling. However such a brute-force approach can be very efficient and effective. I'll demonstrate two basic approaches. I'll make one shape and then insert keyframes, one at a time, making a minor change as I go. Also, I'll show how a series of blank keyframes can be drawn into very quickly to make a nice animation. As I demonstrate these two approaches I'll show Flash's onion-skin feature that allows you to see other frames for reference. Realize the user can only be at one moment in time at a time... but while you're authoring onion-skin lets you see where you've been or where you're headed—or both.



The Onion skin tool helps you animate by seeing where you've been.

Once you see how Flash can help you animate you may never want to animate frame-by-frame again. But don't forget this is ultimately the way to animate! I'll come back to this fact later—but I won't deprive you of Flash's tweening features.

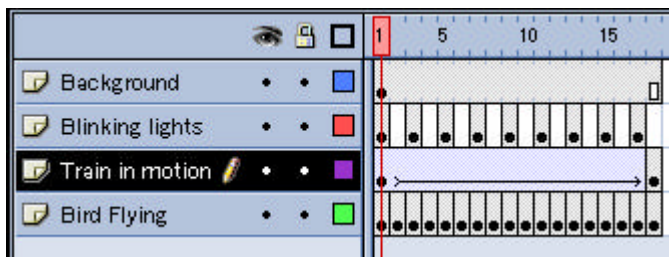
Tweening (or in-betweening) is simply a way of interpolating two keyframes. In traditional animation there are the principal artists (who only draw the keyframes representing coarse movements) and the "tweeners" (who fill in the frames between keyframes and who constantly suffer the humiliation of telling people what they do). We get to draw the keyframes and then make Flash tween the difference. The basic idea is simple: draw one keyframe, then, later in the timeline (not the very next frame) draw another keyframe... then, tell Flash how to get *from* the first frame *to* the end frame. (Remember, you tell Flash how to get from the first frame to the second—you don't go to the end frame and tell Flash how it got there.)

In the timeline, select the first keyframe and use the Frame panel to set tweening to either Motion or Shape. The main difference is that Shape tweens create a morph-like effect where Motion tweens only tween properties (location, scale, rotation, and color effect). Realize these rules for the two tween types:

--Motion allows you to only tween one object per layer—and that object must be a symbol instance.

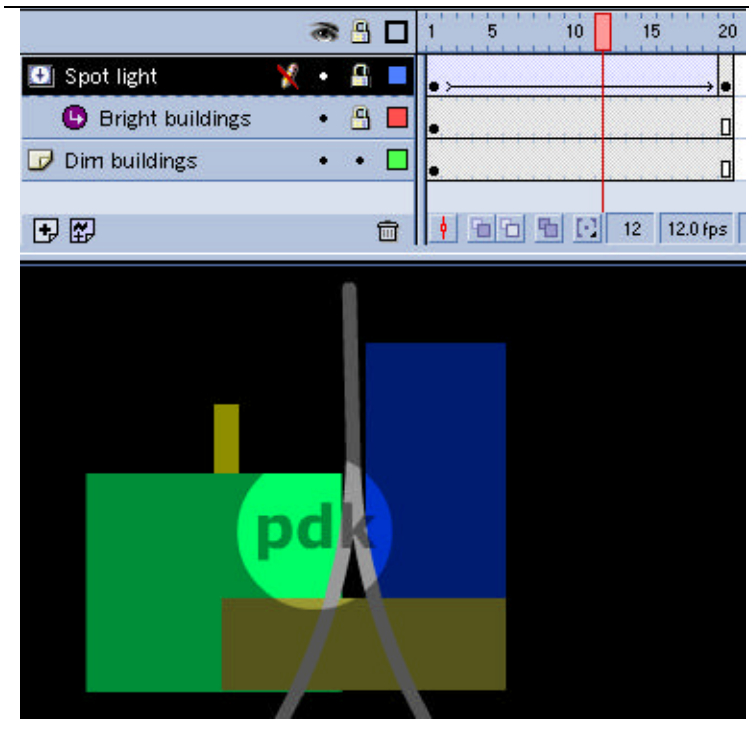
--Shape tweens requires that only un-grouped non-symbol instances appear in the layer. (Tip: keep it simple or consider separate layers for more complex effects as complex shape tweens can be uncontrollable.)

People often (mistakenly) believe Layers in Flash's timeline are for visual stacking. Of course they work this way—but they're really needed to separate animations and for special effects. Consider the fact Motion tweens require only one symbol per layer. You'll often end up with separate layers for each tween. Also, something like a background graphic may never move... or move very slowly. This needs to be in an independent layer for such an independent behavior. Tip: when you start working with layers pay special attention to the pencil icon that indicates the current layer—this is the layer into which any drawing (or pasting) will occur. (Similarly, you always want to be clear where the red current frame marker is placed as it indicates which moment in time you're editing.)



Pay attention to the pencil icon indicating in which layer you're currently editing.

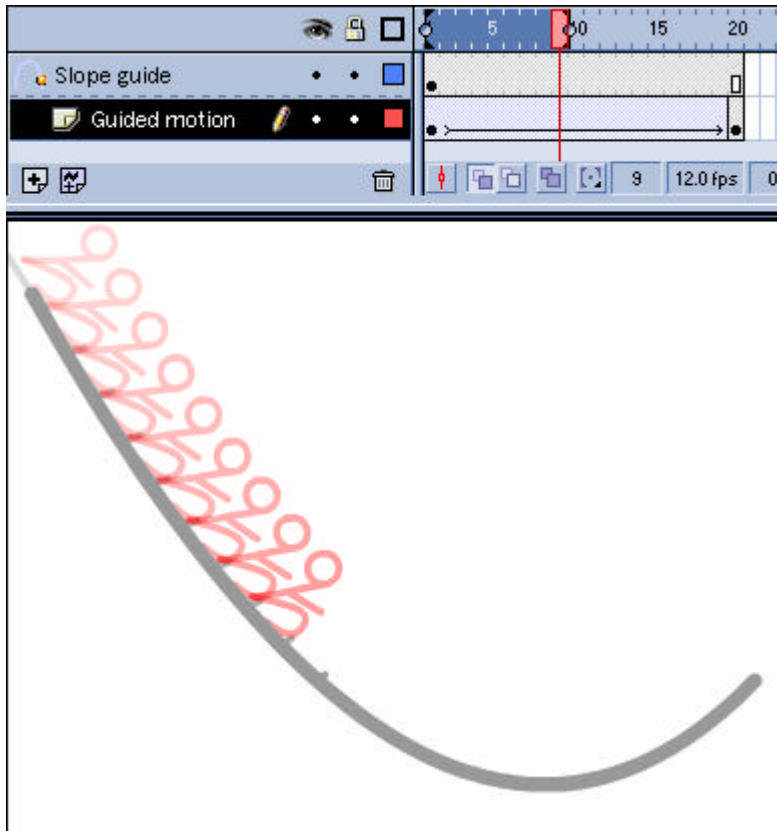
Layers also allow you to make masking effects—where the shapes in one layer defines what part of another layer will be visible. Think of a mask on your face. The shape you draw in a "mask" layer defines the holes (like eye and mouth holes in a mask) that will reveal the contents of the "masked" layer (that is your face behind the mask). The point is you have two layers—"mask" and "masked". They must be adjacent. One mask can have several "masked" layers below. Because every object in a mask layer *behaves* like a plain shape (even if it's a symbol) often you'll want to animate your masked layer instead of the mask layer. However, it just depends what effect you're trying to achieve.



*Just one example of masking—
a spotlight effect.*

Finally, another layer type is guide. All guides become invisible when you export the movie—so the user never sees it. Changing a layer to "guide" is a good way to keep things you plan to remove... but they're safe in case you change your mind and decide to change the layer back. Into guides you can draw anything you want, it won't add to the final filesize. Guides are a great place to save production notes or for guidelines to which you snap objects from other layers for alignment.

The funny thing about guide layers is that you can set the next layer below to "guided" (sort of like mask and masked—there's guide and guided). But this arrangement changes the plain guide to a "motion guide". It's still a guide (it doesn't export with the movie) but when used in combination with a motion tween, the motion guide serves to define the path your symbol instance will follow. This way you can draw a squiggly line and make an instance follow the line.

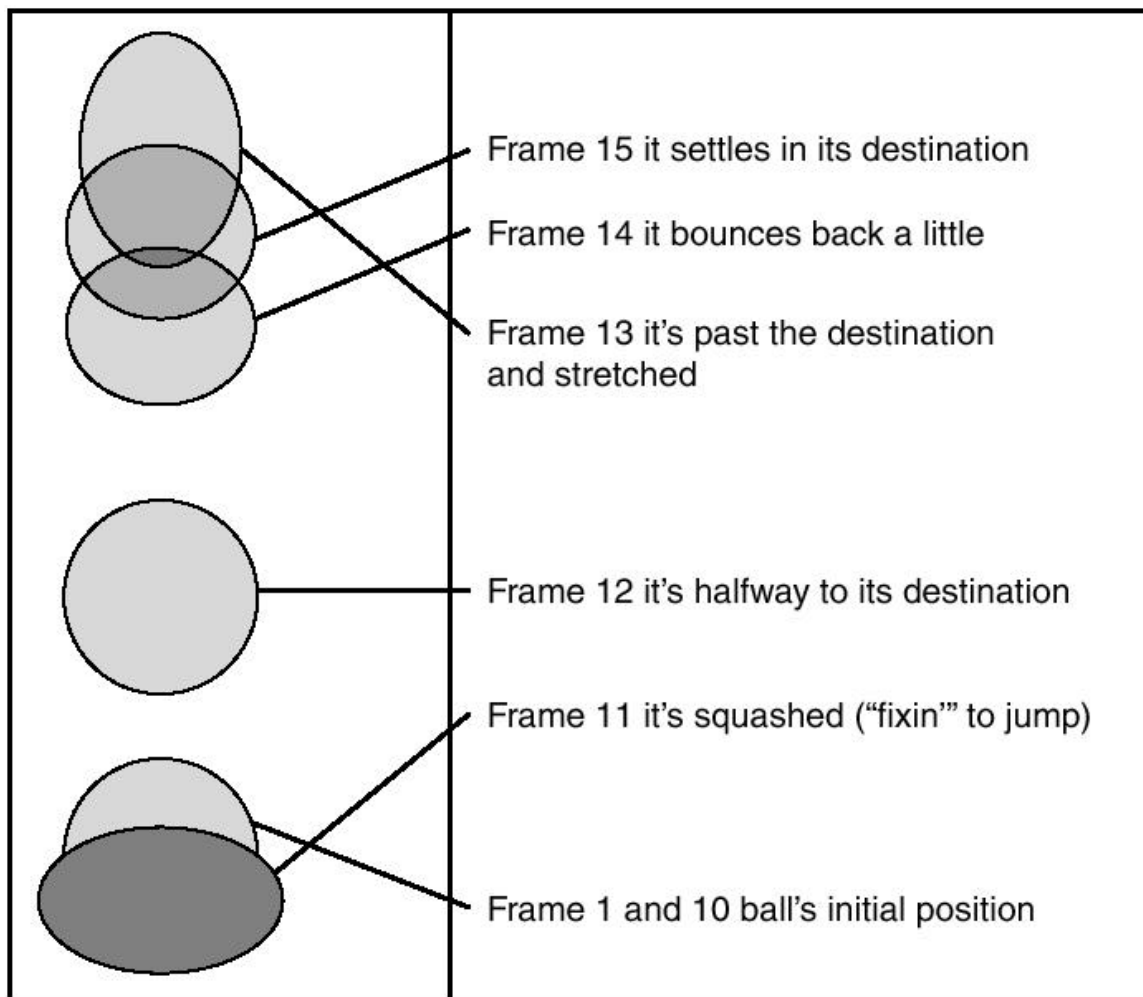


A curved path as a motion guide can make a skier follow a curved line (onion skin is turned on for this image).

Hour 2: Beyond the Basics

Before you go wild with so-called advanced features, it's wise to step back and consider age-old animation techniques. If they worked well before Flash surely they'll work well now. Two techniques involve sort of over-doing it—but that's good. Remember it's rare that you want to make an animation that really behaves the *same* as some physical movement. Rather the goal is to make something look the same, or just look the way you think it looks. There's a difference between what your mind sees and what your eyes focus on. Anyway, two techniques are anticipation and overkill.

Anticipation means to clue the viewer into a motion before it happens. Say a ball is going to move down. Before it moves, nudge it up a little bit. When the user sees it creep up they're effectively told "hey, something's going to happen here...look!". Overkill is simply adding a recoil effect. Not unlike the way passengers in a car feel like the bounce backwards slightly after stop... you can add such a touch for added realism.



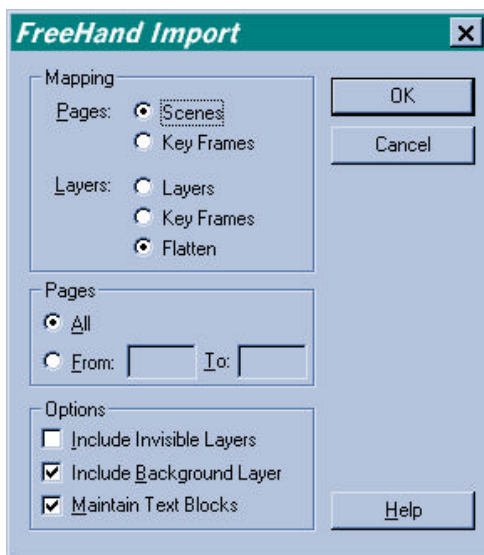
A bouncing ball sequence using anticipation and overkill.

Finally, try to find ways to make your animation more believable. Add intentional "mistakes". For example, you can make an animation of a car moving more believable by adding a couple bumps here and there. The point is, you want your animation to look like real-life not like a computer built it. There's certainly more on this subject. Consider watching an old cartoon or animated movie frame by frame and you'll see a lot of tricks. (You'll also see how they often "cheated" by not animating things which moved fast or things that could be covered up—perhaps with a sound effect—by occurring off screen.)

Animating text is less of a feature and more just technique. Hopefully, you'll be able to first convert the text into a symbol (or symbols if you want to animate each letter individually) so that you can use Motion tween. Of course, if you want to morph one letter into another, you'll need un-grouped non-symbols. Actually, you need plain shapes to do a shape tween which requires the text is first broken apart (Modify>Break Apart). It's not that you'll find the "animate text" button in Flash... but you will notice one common theme in the effects I demonstrate—they always "start with the end in mind". That is, if you want the text to settle in place so you can read it, you should define the end keyframes first... then step back and move the text to where it will be coming *from*.

At some point you'll likely want to import media from outside sources. Always first consider whether it's possible to draw it in Flash. Often it's worth the effort to re-draw something in Flash (even if you have to trace it) because you'll see significant filesize savings. However, Flash only creates vector graphics. If you want to include a raster graphic (say, something photo realistic) you'll need another program. Similarly, Flash can't create audio (just import it).

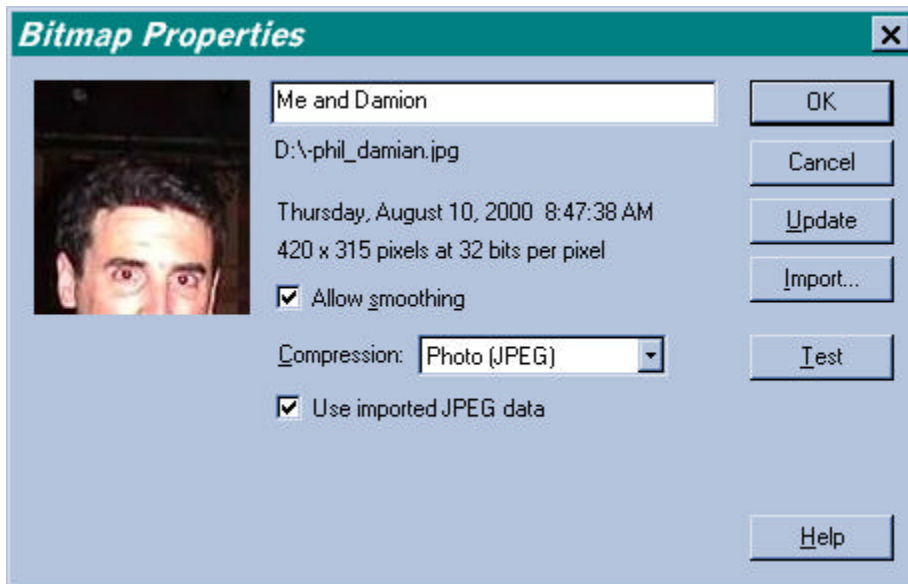
Even despite my persuasive argument that Flash is a great vector drawing tool, you'll likely want to import vector artwork from Freehand or Illustrator. Since Flash 5, Freehand files are very well supported. A dialog appears when you import native Freehand documents to help you decide how to handle details of the import. For example, Freehand files can have several pages. Well, in Flash you must decide what to do with those pages—do you want separate keyframes created... or separate scenes? Do you want text to remain editable?... etc. All these details are answered in the import dialog that appears.



When importing from Freehand you're given this detailed dialog.

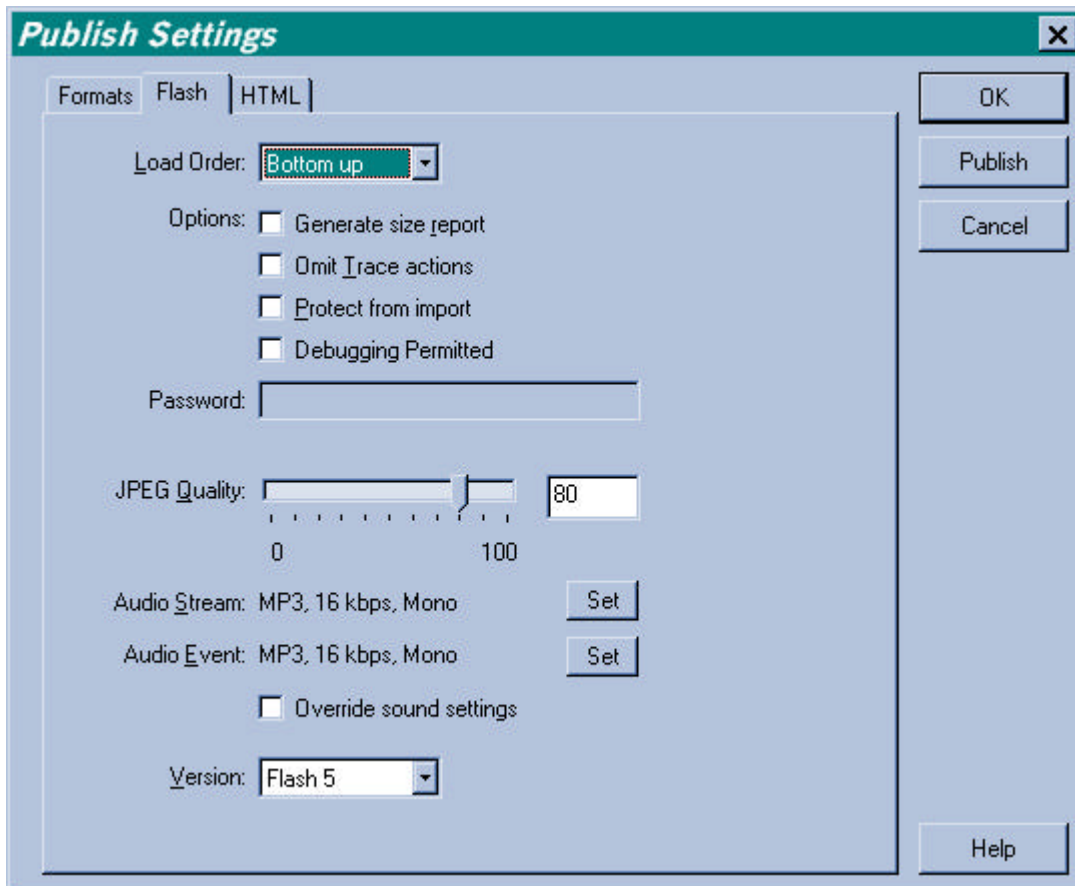
You can also import Illustrator files. There are probably other variations to my suggestion here, but basically you want to either save your Illustrator documents in version 6.0 or use the free Macromedia plugin for Illustrator that allows you to export Flash .swf files (I think this feature is built into the latest version of Illustrator). (By the way, I would suggest not even bothering with trying to import EPS files as there are so many variations that EPS is nearly not a standard format.)

Importing raster graphics is simple—just select File>Import. There's certainly a lot you can say about different file formats and their respective advantages and disadvantages—but I'll assume you know this (or can research it as needed). Anyway, once the raster graphic is in Flash you need to decide how it will be treated. That is, Flash can compress images using JPG compression... or, if you want, you can leave images uncompressed. Plus, if you import an image that's already compressed (say, a JPG) you'll need to decide whether you want Flash to re-compress (usually a bad idea). Anyway, all of these settings are found individually for each imported image through the Library item's properties dialog.



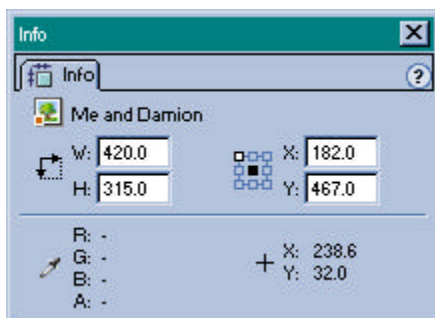
An individual image's properties dialog contains several options regarding compression.

Generally, you decide between compression or no compression. If the image is already compressed you'll find the option "Use Imported JPG Data" which means "don't recompress". In addition to the individual properties per image, there's a global setting for JPG compression in the Publish Settings. This will affect only images imported that have their individual properties set to use "default" JPG compression. Basically, this is a way to increase or decrease all compression globally for the whole movie. (Again, this publish setting affects only certain images in the movie—and has no effect on non-raster graphics such as those you draw inside Flash.)



The Publish Settings dialog provides global compression settings.

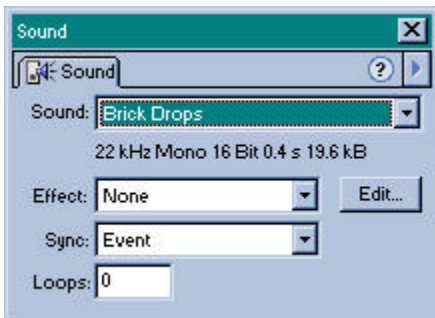
One last thing about raster graphics. You want to always position them in whole pixel locations. Using the info panel, you'll want to make sure the x and y coordinates of a raster graphic are whole number (103.0 not 103.81 for example). This can help prevent an annoying shift that is common when using raster graphics in Flash.



You should position raster graphics in whole number coordinates (see X Y top right)

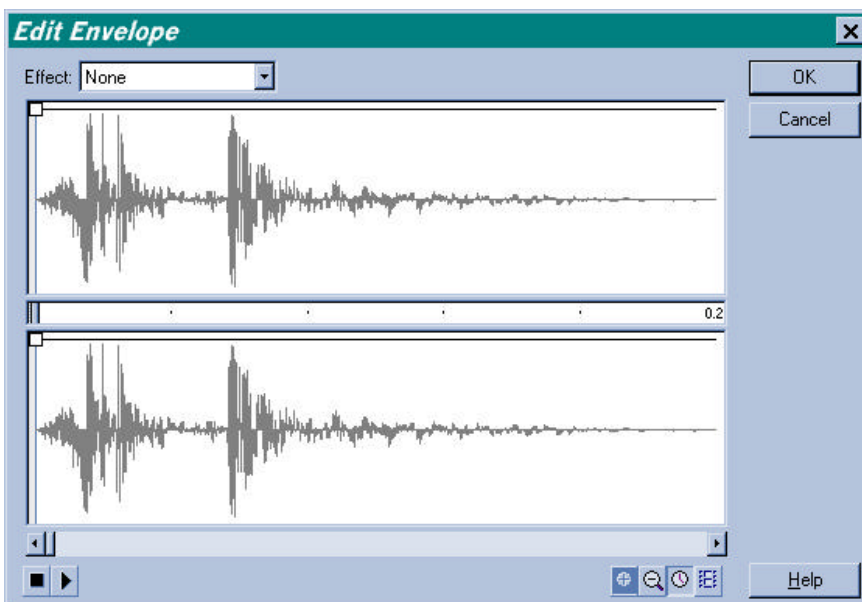
Importing audio is similar to importing raster graphics—you'll find compression settings individually (in the respective library item's properties) and globally (in the publish settings). Naturally, there's much more than can be said regarding the compression alternatives. For a detailed discussion everything I know about Sound in Flash, view the handout and presentation "The Sound of Flash" that I gave at another Thunder Lizard conference: www.phillipkerman.com/wdw2001

When it comes to *using* audio in your movie, much more can be discussed. Sounds are placed in keyframes. Simply select a keyframe and open the sound panel. From there you can select any sound that's already been imported into your movie.



When you select a keyframe the sound panel lets you select any previously imported sound.

The sound panel has more options to modify precisely how a sound plays. Pressing the Edit... button reveals the "Edit Envelop" dialog where you can adjust when a sound starts, stops, increases, or decreases volume—in relation to the entire imported sound. That is, the sound will start when the playback head reaches the keyframe but you can make the sound pan from left to right or start playing from a point other than the beginning of the sound. (One thing to remember with Edit Envelop is settings only affect this instance of the sound—not the original item imported into the Library.)

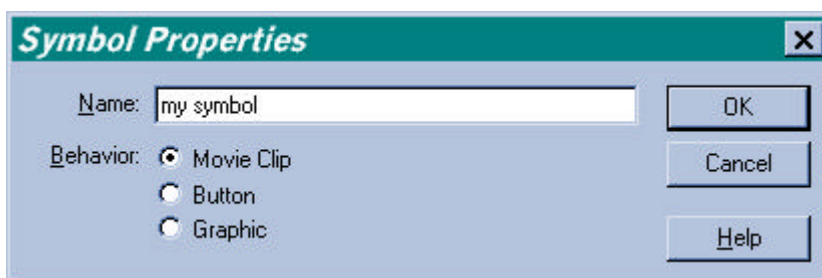


Edit Envelop lets you make changes to a particular sound instance.

The most significant setting in the sound panel is the Sync setting. From here you control how the sound plays in relation to other sounds and in relation to the timeline. There are four settings possible—Event, Stream, Start, Stop. Event and Stream are the most difficult to understand. Event sounds start playing when the keyframe is reached. Other than the keyframe that triggers the sound, Event sounds have no relation to the timeline. In the case of Stream sounds, they too start when the keyframe is reached but they continue to play in perfect synchronization with the timeline. Stream sounds require that you provide enough frames for them to finish—that is, a 2 second sound set to stream (in a movie playing at 12 fps) needs 24 frames of timeline to finish—otherwise the sound gets cut off early. "Perfect synchronization" with the timeline may sound like a great benefit of Stream sounds—and it can be. In the ideal situation you'll be able to place animated graphics in other layers and, while authoring, hear and see the animation play as the user will. However Stream sounds cause problems on machines that can't display animated graphics fast enough. Realize some computers may not be able to keep up with your animation. Therefore, instead of making the sound slow down (it'd sound funny if it did), Flash simply skips graphic frames as needed to keep up with the timing. Conversely, Event sounds start playing and then may finish before or after the graphics finish displaying (depending on how fast the user's computer can display graphics). Basically, some machines cause Flash to slow down... but since Stream sounds are tied to the timeline, Flash drops frames to keep up. (Tip: compare the difference between Event and Stream to how a Movie Clip compares to a Graphic clip—below.)

The other two settings are easy. Start is the same as Event but will only play the specified sound provided it isn't already playing. Start will prevent the same sound from overlapping. (I'll have examples when you want this and when you want the opposite.) Stop is a weird one. You specify the sound (that's been imported) just like the other options—but when a keyframe with Stop selected will cause that particular sound to stop (provided it's already playing).

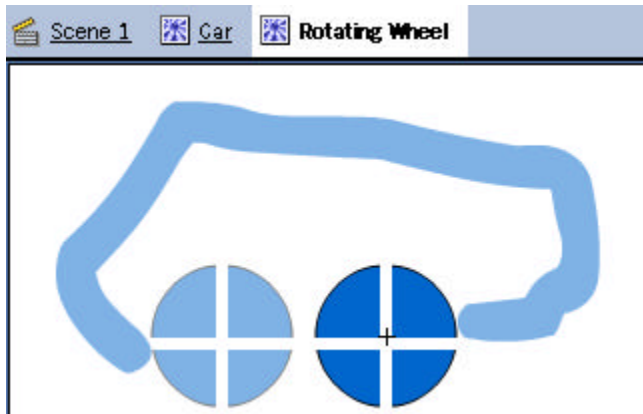
Nesting animations using Movie Clips. We've gotten this far without even defining the three symbol types! Movie Clip, Button, and Graphic. Buttons will be used later when you want to tie a script (or instruction) to a button.



When you create a symbol you can choose between the three Behavior types.

Before comparing Clips to Graphics be sure you understand how each symbol contains it's own timeline. In this way, we can make a symbol that contains an animation. Say you have just a wheel shape that's converted to a symbol. You can then animate that wheel using Motion tween—since it's a symbol. However, if you make another symbol (call it "rotating wheel") you can drag an instance of the plain wheel symbol into the

rotating wheel's timeline. Do a little animation (say, of the wheel making a single rotation). Then create a third symbol (called "car") that contains two rotating wheels plus a car body. Finally, you can drag the car to the main timeline and animate it across the screen. But you'll see more than just a car moving... you'll see a car moving that contains wheels that are rotating. While this brief explanation might be hard to follow, you'll always want to be clear "where you are" when working in Flash. Pay attention to the address bar and you'll do alright.

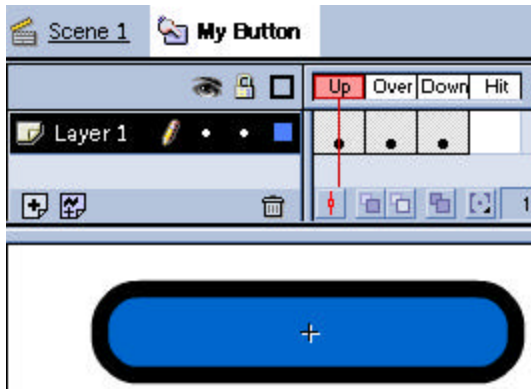


Always pay attention to the address bar when editing nested symbols.

The easiest way to make the car with rotating wheels example is to use the default symbol type Movie Clip. The main thing you'll notice about instances of Movie Clips that contain multiple frames (that is, an animation) is that you can drag such Movie Clips anywhere in your movie, and when you test the movie, you'll see them animate. You might say, duh... of course I want to see them animate. Well, what if you have a rotating wheel in a symbol that takes 10 frames to make one rotation... and the you place this rotating wheel symbol in a timeline that has 5 frames and then stops? (We'll discuss making the timeline stop later.) Well, do you want to see the wheel rotate for the 5 frames and then stop? ... or do you want it to keep spinning? It's up to you. There are cases for both. If you use Movie Clips they can be placed in timelines of any length and they'll continue to animate—even if they have only one frame. If you use Graphic symbols they'll animate only provided they have enough frames to do so. Placing a 20-frame Graphic animation in a 10-frame timeline gives you only half the animation. You might think Graphic symbols are useless—but there's one other detail of Graphics that make them nice. When you are authoring you'll get a preview of the animation contained in the nested symbol. That is, you'll see it rotate or whatever it's doing inside the symbol. Basically, this is good for situations where you want to synchronize a nested animation with another such as with lip syncing. Graphic symbols are like Steam sounds—both are "tied" to the timeline where they're used. Clip symbols are like Event sounds—they both just play on their own.

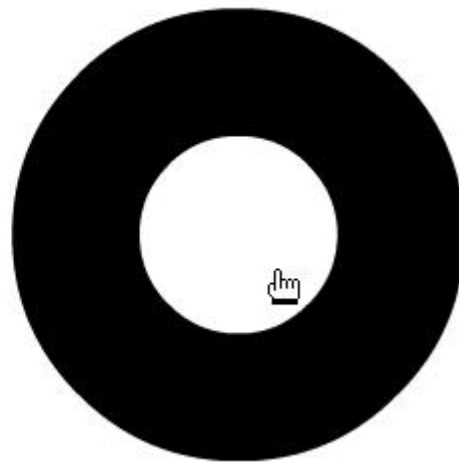
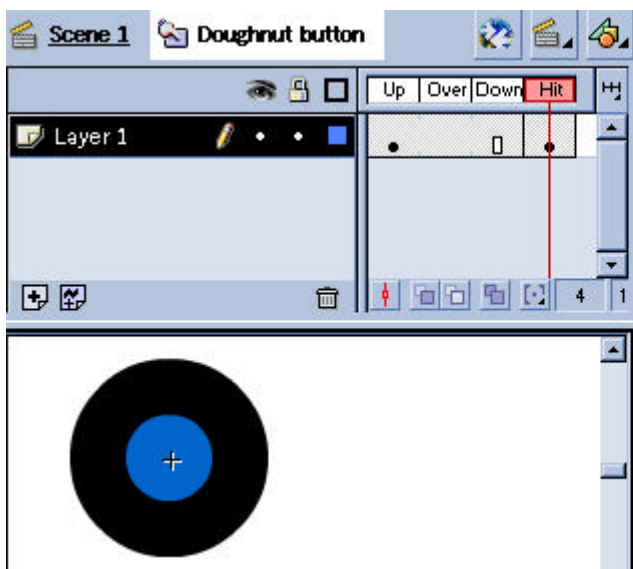
It turns out Movie Clip instances have another big advantage. Namely, using scripting you can change Clip instances' properties (like location and transparency) dynamically while the movie plays. So, generally, always use Movie Clips unless you need one of the characteristics in Graphics... or if you want a button.

Making buttons to click on. First realize there are two parts to a button: building the visual elements (how it appears when you roll over etc.); and assigning scripts so Flash *does* something when the user clicks them. First, we'll look at how you create the graphic elements in a button. When you edit the contents of a button symbol you'll see 4-frames and their named. You can use just one keyframe at the beginning (and the contents of frame 1 will appear in all frames), or you can make multiple keyframes and make something different appear in each frame.



Button symbols have 4 named frames or "states".

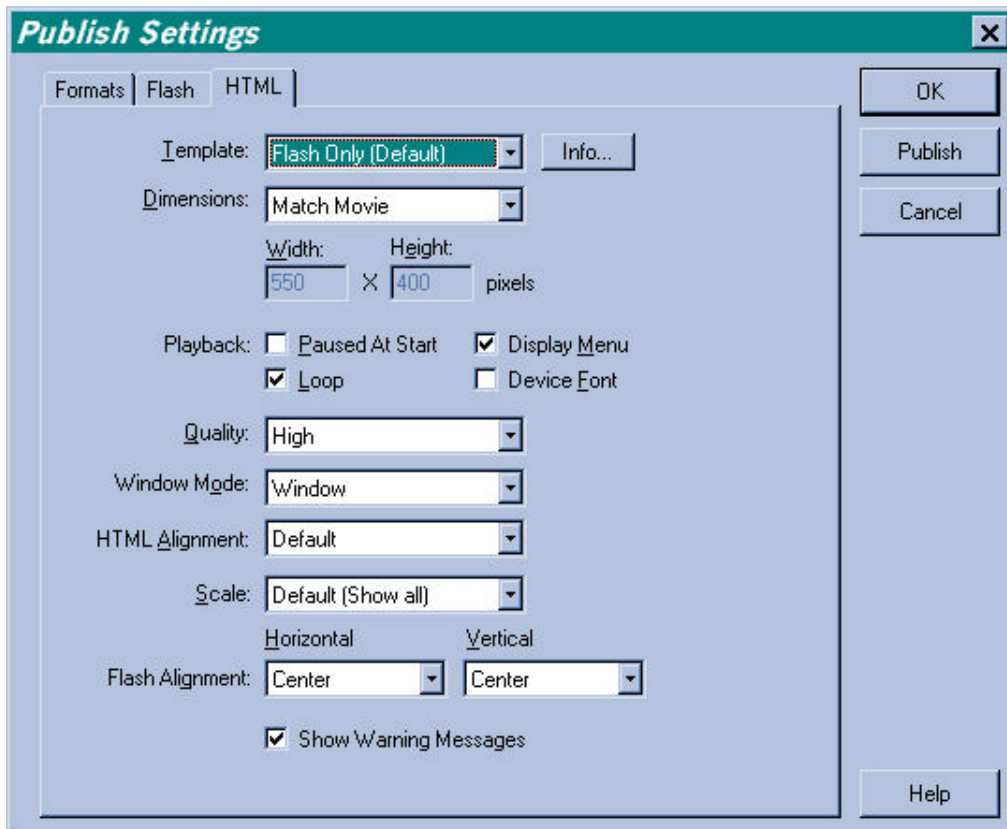
The four frames are "states" of the button. Into the Up state you draw how the button should appear normally. The Over state is where you can draw how the button should appear while the user's cursor is on top of the button. Down contains the look of the button when the user presses. Finally, Hit (if you create a keyframe here) lets you draw the invisible shape that defines the area necessary for the button to be active. Consider a button shaped like a doughnut. If you want the user to be able to click on the doughnut button, even when they're over the center doughnut hole if you will, then you need to draw a solid shape in the Hit state. No one "sees" the hit state—but it defines what's clickable. A nice large and solid Hit state makes buttons that are otherwise just text much easier to click.



The Hit state defines what's clickable on a button.

Animated buttons are very easy to produce. You can put an animation in any or all the states--Up, Over, or Down. However, since each is only one frame long you'll need to first make an animated Movie Clip symbol then place an instance in the appropriate state. That is, if you want an animation to appear when the user rolls over the button—just put an animated Movie Clip symbol instance in the Over state. It really is that simple.

Publishing is the process of taking your masterpiece Flash movie and exporting it to a form that anyone on the internet can view. Your source Flash document is a .fla file (keep it so you can make edits). Every "page" on the internet is an HTML document containing text and formatting information plus the filenames of any images that are to appear within the page. Similarly, a webpage with Flash is an HTML document containing formatting information and the filename for the Flash movie (in the exported form .swf). Basically you need to produce an HTML file and a .swf. Through the Publish Settings you can specify the details you want for both the HTML (and formatting) and the .swf (including compression settings and Flash version number required). Then, when you select File>Publish Flash will export the files as you requested.



These publish settings affect the HTML that gets exported.

Without stepping through each setting, realize that Flash is still creating an HTML file... Publish just automates this process. Feel free to snoop through the exported HTML to learn what Flash is doing. This is especially useful if you want to integrate the .swf inside a different HTML file that you produced elsewhere. Finally, the last step is to upload the HTML and .swf file to your webserver and then call your friends to tell them to check out your movie.

Hour 3: User Interfaces using ActionScript

Scripting is nothing more than writing instructions. It just so happens these instructions can't be written in your own words—but rather follow the strict syntax of the language called ActionScript. While learning this language can be challenging—the hard part that remains is clearly stating the instructions you want Flash to follow. Besides just writing instructions you have to figure ways for those instructions to get executed at the right time. The idea is that events will trigger your scripts. Say the instruction is "move the ball" (not actual ActionScript but my own words). You need figure *when* such a script is intended to execute. Should the event that the user presses a button trigger that script? Should the event that the movie comes to the last frame trigger it? It's up to you.

We'll look at events in a second, but try to step back and first write scripts by thinking of detailed instructions—using your own words. Saying "move the ball" is great. If you can get more detailed, like "move the ball 10 pixels to the right of where the ball is currently" then that's even better. But don't bother using grammatically correct ActionScript syntax right off the bat. If you clearly define the instructions first, it's just a matter of routine to clean up your "pseudo-code" into actual legitimate ActionScript. The technique of pseudo-code should not be discounted!

There are three places you can put scripts in Flash—keyframes, button instances, and movie clip instances. Placing a script in one of these three places also affects the kinds of events that will trigger the script. That is, a script on a button can be triggered by any of the mouse events—press, roll over, release. You don't just say "move the ball" you say "on the event button is pressed, then move the ball". Or, you can say "on the event the user presses (and then releases) the button, then move the ball". These are the types of events available to buttons. Think "button-type" things.

Placing a script in a keyframe is a bit more straightforward. The "event" that triggers the script is simply the playback head reaching that keyframe. You might put the script "stop" in the last keyframe of a movie. When that frame is reached, the movie stops. You could also put "gotoAndPlay(10)" in frame 15. (Translated back to pseudo-code that means "jump to frame 10 and play from there"). Every time frame 15 is reached the playback head jumps back to frame 10 and plays. The effect is the last 5 frames loop indefinitely.

The last place where scripts can be placed is on instances of Movie Clip symbols. The selection of "clip events" is not quite as intuitive as the mouse events that go with buttons. One clip event is EnterFrame meaning the script will execute 12 times a second (provided the frame rate is 12fps). In this way, you don't have to wait for a frame to be reached or for a user to click a button—your script can respond to the enterFrame script and effectively execute repeatedly. The clip events are slightly more difficult to understand, but they're very powerful.

One thing about events (whether they're mouse events or clip events) is they follow the same basic form. One of the two following forms:

For a button only:

```
on (press){
    //script goes here
}
```

For a clip instance only:

```
onClipEvent(enterFrame){
    //script goes here
}
```

Notice in either form the specific event (press or enterFrame shown above) goes inside the parenthesis. Also, notice the open curly brace in the first line is finally closed way at the end. Any scripts placed between the two curly braces will execute when the specified event is triggered. That is, you can have more than one result from an event. Tip: remember to close anything you open—quotes, parenthesis, curly braces, etc.

ActionScript commands are simply Flash-specific features that cause the movie to change. Both **stop()** and **play()** are commands with very simple results. Another, **gotoAndPlay()** is slightly more complex. You need to provide a parameter to specify which frame you want to "go to". Just like you can't buy a plane ticket without specifying where you want to fly to (you need to provide that information)... the **gotoAndPlay()** command requires that you specify the frame number you want to go to. Actually, the parameter goes within the parenthesis—so you can say **gotoAndPlay(1)** to jump to frame 1. Even if a command doesn't require a parameter you'll see the parenthesis—this is one way to identify them.

Besides invoking commands to change the course of a movie, you can affect any clip instances on stage by changing that clip's properties. Just like how you can manually change clips' properties (say, location or alpha) you can do so using script—but only while the movie plays. That is, moving or changing a clip's properties manually (with the mouse) when authoring permanently changes your source file. Using script to change properties only affects the movie while it plays.

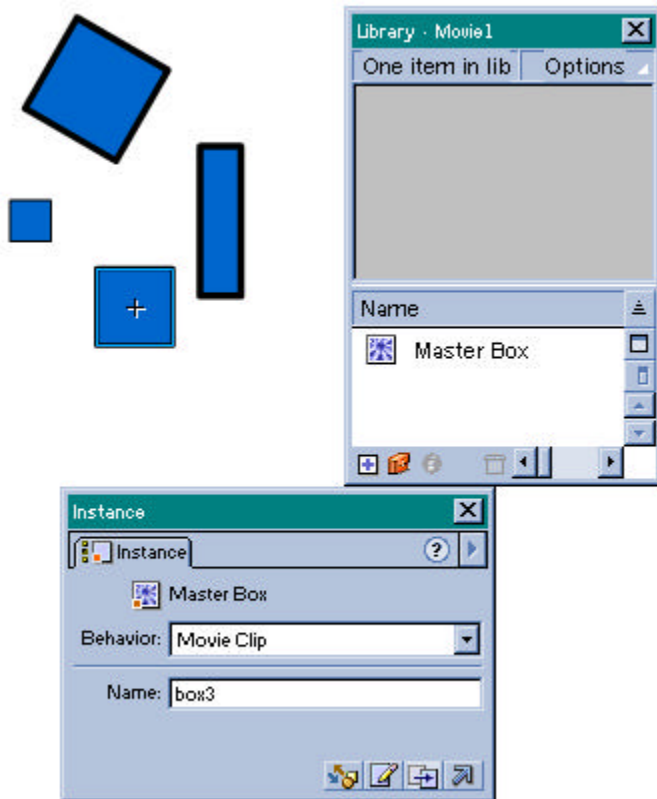
To change a clip's properties use this basic form:

```
clipInstanceName.property = newValue
```

Replace **clipInstanceName** with the actual name given to the instance through the instance panel, change **property** to one of the built-in properties (like **_x** or **_alpha**) and change **newValue** to some logical value, say 50. If you want to affect a clip that's nested within another clip just make sure all instances are named and use:

```
clip.clipInAClip.property = newValue
```

Remember to only use instance names—not the name in the library—this way you can have several unique instances onstage (all with different properties).



Always use the instance panel to set a clip's instance name for this is the one you use when changing or referring to properties.

By the way, sometimes you'll only need to *refer* to a clip's property—not necessarily change it. That is, as just part of an expression. For example, consider the pseudo-code "move the ball 10 pixels to the right of where it is". You can get close to real code by saying:

ball._x = where it is plus 10

Naturally, the part on the right is still pseudo-code. The expression "where it is" is the same as saying "ball._x". So the finished code is:

ball._x = ball._x + 10

So whether you're changing properties or just referring to properties, the form is the same: *clip.property*. It's just that if you say *clip.property=something* the equals sign is simply performing an assignment.

Properties are just a way for clips to maintain information about themselves. Just like your hair-color, height, and weight are properties. If the built-in properties are not enough, you can effectively make up your own. Keep in mind though, all the built-in properties correspond to a visual component of clip instances (width, location, alpha etc.). When you make up your own they'll have no visual result unless you decide to write a script which later uses the homemade properties. Why would you want clips to have homemade properties? Say you want each clip to move across the screen but at a rate that varied. You could make up a property called "speed". It's as simple as deciding you want this property and then start using it.

`ball.speed=5` translated says, "set the speed property of the clip called ball to 5".

So much for making a new property—that's easy. But like I said, you'll never see anything unless you use that property in conjunction with the built-in properties.

Consider if you ball instance had the following code on it (remember it's a clip so it can use clip events):

```
onClipEvent (enterFrame){
    ball._x = ball._x + ball.speed
}
```

The intent is that every 1/12 of a second the ball moves to the right a distance proportional to the ball's speed. Naturally, however, this doesn't work as advertised. The problem is the above code assumes there's a clip called ball—inside the clip... but it's on the clip itself that we're writing this code. You can instead use this code and it will work as you might expect:

```
onClipEvent (enterFrame){
    _x = _x + speed
}
```

Don't forget to assign speed a value... maybe in the first keyframe of the movie say `ball.speed=5` (or whatever). If you leave out the name of the clip (in front of a property) Flash assumes you're talking about the current clip—the one where the script appears. The reason my first example didn't work has to do with relative or absolute references—a subject that's not being covered here today. (You can find much more on this and other related subjects in the documents at www.phillipkerman.com/fk01)

Finally, we can look at some extended examples. These files can be downloaded and explored at www.phillipkerman.com/wb01

Additional links and downloads from my books (including full chapters and all the workshop files) can be found at: www.phillipkerman.com/books