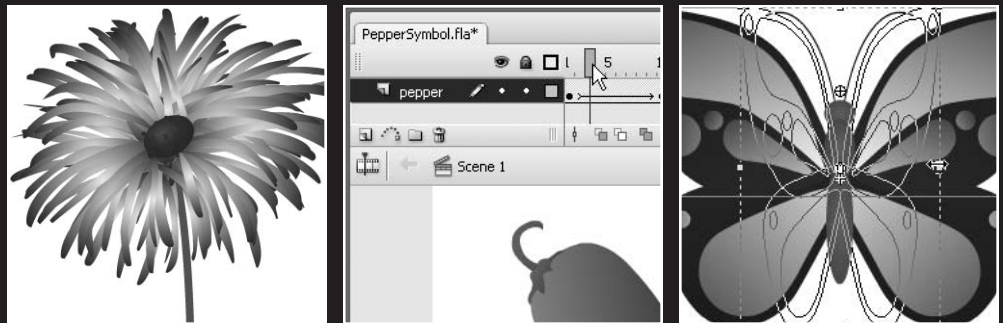


## 7 ANIMATION IN FLASH CS3



Ahh, animation! Where would we be without the likes of Disney, Warner Bros., Walter Lanz, Hanna-Barbera, and dozens more like them? For many people, animation is *the reason* to get involved with Flash as a creative outlet. This makes perfect sense, because Flash began life a full decade ago as an animation tool. Supplemental features like ActionScript, XML parsing, and video integration—every one a tremendous addition—all followed. What hasn't changed in all these years is Flash's ability to produce quality, scalable animation for the Web, and increasingly for television.

You caught the faintest whiff of tweening in Chapters 1, 2, and 3. It gets considerably more complex (read *considerably more fun!*), and because this chapter has a lot of moving parts, let's stop with the talking already and jump directly into the fray.

What we'll cover in this chapter:

- Shape tweening
- Shape hinting
- Motion tweening
- Easing
- Using the Custom Ease In/Ease Out editor
- Animating symbols
- Combining timelines
- Motion tween effects

Files used in this chapter:

- PepperShape.flc (Chapter07/ExerciseFiles\_CH07/Exercise/PepperShape.flc)
- StarStar.flc (Chapter07/ExerciseFiles\_CH07/Exercise/StarStar.flc)
- StarCircle.flc (Chapter07/ExerciseFiles\_CH07/Exercise/StarCircle.flc)
- Ant.flc (Chapter07/ExerciseFiles\_CH07/Exercise/Ant.flc)
- LogoMorphNoHints.flc (Chapter07/Exercise Files\_CH07/Exercise/LogoMorphNoHints.flc)
- LogoMorph.flc (Chapter07/ExerciseFiles\_CH07/Exercise/LogoMorph.flc)
- FlowerWeed.flc (Chapter07/ExerciseFiles\_CH07/Exercise/FlowerWeed.flc)
- GradientTween1.flc (Chapter07/ExerciseFiles\_CH07/Exercise/GradientTween1.flc)
- GradientTween2.flc (Chapter07/ExerciseFiles\_CH07/Exercise/GradientTween2.flc)
- BitmapFillTween.flc (Chapter07/ExerciseFiles\_CH07/Exercise/BitmapFillTween.flc)
- PepperSymbol.flc (Chapter07/ExerciseFiles\_CH07/Exercise/PepperSymbol.flc)
- MalletNoEasing.flc (Chapter07/ExerciseFiles\_CH07/Exercise/MalletNoEasing.flc)

- MalletCustomEasing.flc (Chapter07/ExerciseFiles\_CH07/Exercise/MalletCustomEasing.flc)
- CustomEasingComparison.flc (Chapter07/ExerciseFiles\_CH07/Exercise/CustomEasingComparison.flc)
- CustomEasingMultiple.flc (Chapter07/ExerciseFiles\_CH07/Exercise/CustomEasingMultiple.flc)
- YawningParrot.flc (Chapter07/ExerciseFiles\_CH07/Exercise/YawningParrot.flc)
- SyncPropertyGraphic.flc (Chapter07/ExerciseFiles\_CH07/Exercise/SyncPropertyGraphic.flc)
- EditMultipleFrames.flc (Chapter07/ExerciseFiles\_CH07/Exercise/EditMultipleFrames.flc)
- TimelineCombine.flc (Chapter07/ExerciseFiles\_CH07/Exercise/TimelineCombine.flc)
- Grotto.flc (Chapter07/ExerciseFiles\_CH07/Exercise/Grotto.flc)
- tronguy.png (Chapter07/ExerciseFiles\_CH07/Exercise/tronguy.png)
- TronGuyGlow.flc (Chapter07/ExerciseFiles\_CH07/Exercise/TronGuyGlow.flc)
- FadingParrot.flc (Chapter07/ExerciseFiles\_CH07/Exercise/FadingParrot.flc)
- MotionGuide.flc (Chapter07/ExerciseFiles\_CH07/Exercise/MotionGuide.flc)
- TweenMask.flc (Chapter07/ExerciseFiles\_CH07/Exercise/TweenMask.flc)
- TweenMaskMotionGuide.flc (Chapter07/ExerciseFiles\_CH07/Exercise/TweenMaskMotionGuide.flc)
- AnimatedButton.flc (Chapter07/ExerciseFiles\_CH07/Exercise/AnimatedButton.flc)
- Zap.mp3 (Chapter07/ExerciseFiles\_CH07/Exercise/Zap.mp3)
- CreateMotionAS3.flc (Chapter07/ExerciseFiles\_CH07/Exercise/CreateMotionAS3.flc)

## Shape tweening

As useful as symbols are, both in organizing artwork and reducing SWF file size, they shouldn't overshadow the importance of shapes. After all, unless a symbol is the result of text or an imported image file, chances are good it was constructed from one or more of Flash's most basic of visual entities: the shape.

Shapes differ significantly from symbols, though many of their features overlap. Like symbols, shapes are tweened on keyframes. Tweening may be finessed by something called easing, and can affect things like position, scale, distortion, color, and transparency. The difference comes in how these changes are achieved. In addition, shapes can do something symbols can't: they can actually morph from one set of contours to another!

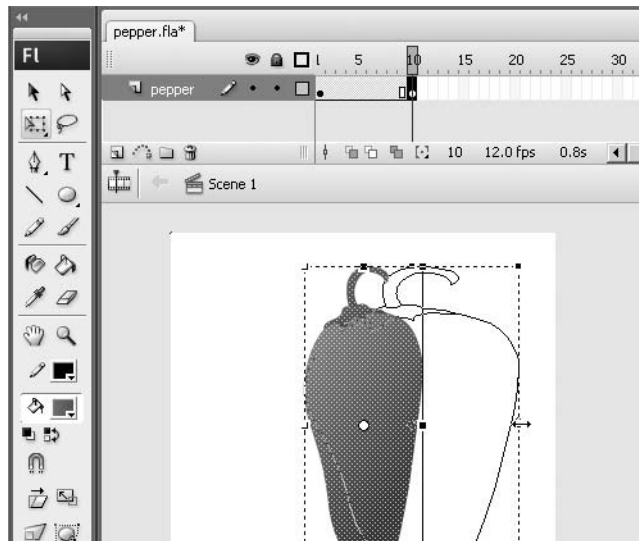
## Scaling and stretching

Let's start with the basics:

1. Open the PepperShape.fla file found in the Chapter 7 Exercise folder. You'll notice there's nothing in the library—this is because the hot pepper on the stage is composed entirely of shapes. Select **Insert** > **Timeline** > **Keyframe** to insert a keyframe at frame 10. This effectively produces a copy of the artwork from frame 1 in frame 10, and makes the copy available for manipulation. Any changes you make to frame 10 will not affect the shapes in frame 1, so you can always remove that second keyframe (**Modify** > **Timeline** > **Clear Keyframe**) and start again from scratch if you need to.

*If you prefer, you can insert a blank keyframe at frame 10 (Insert > Timelines > Blank Keyframe), and then copy and paste the artwork from frame 1. It makes no practical difference, but clearly the first approach requires less effort. You may even draw completely new shapes into frame 10, and Flash will do its best to accommodate—but that's skipping ahead. More on that in the "Altering shapes" section.*

2. With frame 10 selected, choose the **Free Transform** tool and drag the right side of the pepper's bounding box to the right. As you do this, you'll see an outline preview of the shapes in their new stretched size, as shown in Figure 7-1.

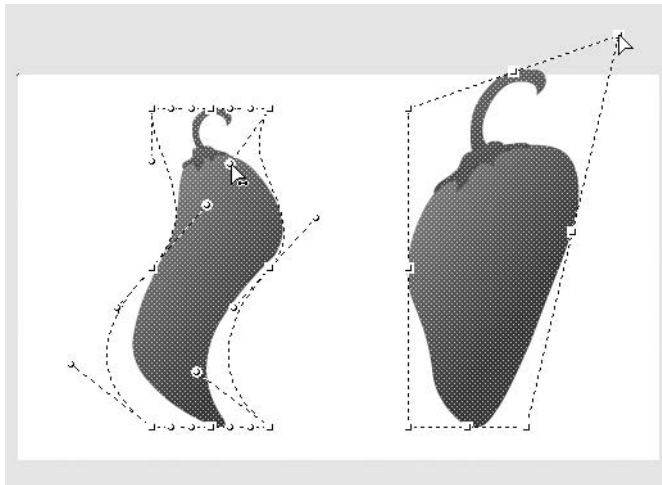


**Figure 7-1.** Changing a shape's shape in preparation for a shape tween

*You might find that you have accidentally selected either only the pepper or only its cap. The Free Transform tool's bounding box will let you know at a glance which shape you have selected, because it will either encompass the full surface area of the artwork or it won't. To ensure you've grabbed all the shapes, either use the Selection tool to first draw a marquee around the whole pepper or, even simpler, click the keyframe at frame 10, which selects everything on that layer in that keyframe.*

3. Select Edit ► Undo Scale to undo. This time, hold down the Alt (PC) or Option (Mac) key while dragging to the right. Notice how the artwork now scales out from the center. This feature often comes in handy, but it's important to understand what's really going on. When the Alt/Option key is used, it's not the center of the artwork that becomes the pivot, but rather the **transformation point**, as indicated by a small white circle. You can drag this circle where you like, even outside the confines of the shape's bounding box. With or without the Alt/Option key, the transformation point acts as the fulcrum of your modifications.

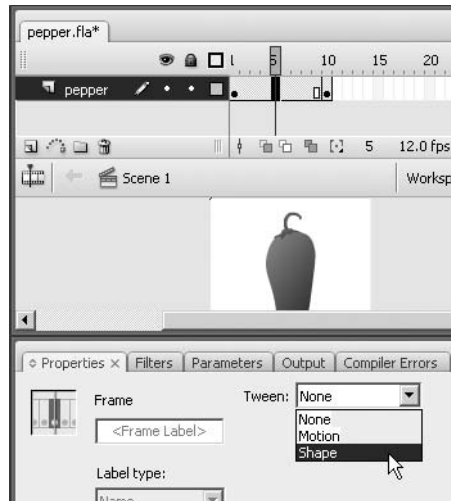
Because you're dealing with shapes, you can even use the Free Transform tool's Distort and Envelope options (shown in Figure 7-2). If you do, just be aware that things can quickly fall apart with such transformations unless you use shape hints (covered later in the chapter).



**Figure 7-2.** Shape tweens support the full gamut of shape transformations.

4. Now that you have two keyframes prepared, it's time for the magic. Click anywhere in the span of frames between both keyframes, and then select Shape from the Tween property in the Property inspector (see Figure 7-3). Two things will happen:
  - The span of frames will turn green, which indicates a shape tween. They will also gain an arrow pointing to the right, which tells you that the tween was successful.
  - The pepper will visually update to reflect a state between the artwork in either keyframe, depending on where the playhead is positioned.

Drag the playhead back and forth to watch the pepper seem to breathe.

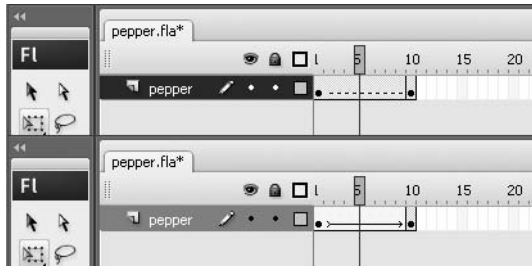


**Figure 7-3.** Applying a shape tween

*If you applied the tween while in frame 1—a perfectly legal choice, by the way—you wouldn't immediately see the pepper change. Why? Because the tweening is applied between the two keyframes, and frame 1 still represents the artwork as it was before tweening was applied. Drag the playhead back and forth, and you'll see the tween.*

*What if the tweened frames don't turn green? By default, they will, but you may have experimented with your Timeline panel settings. Click in the panel's upper-right corner, just below the x, and ensure that Tinted Frames is selected in the context menu.*

5. Select anywhere between the two keyframes and choose None as the Tween property setting. The tween goes away.
6. Let's purposefully make a mistake. This time, choose Motion as the Tween property. Motion tweening is not supported for shapes, and Flash gives you an unmistakable sign that you've gone wrong if you try to use it. Instead of green, the span of frames will become purple—the indication of motion tweens. More importantly, the arrow is now the broken line shown in Figure 7-4. Time to either undo or change the Tween property in the Property inspector.



**Figure 7-4.** Erroneous tweens (top) are indicated by a broken line, while successful tweens (bottom) are indicated by a solid arrow.

7. Change the Tween property back to Shape. Select frame 10 and choose the Free Transform tool once again. Drag one of the bounding box corners to change both the horizontal and vertical scale. If you like, hold down Shift to constrain the aspect ratio, and Alt/Option to apply changes from the center of the transformation point. Make the pepper a good bit bigger than the original size. This shows that it's possible to adjust keyframes even after they're already part of a tween.

*If you right-click (PC) or Ctrl-click (Mac) between any two keyframes, you'll see a Create Shape Tween choice in the context menu. This is new to Flash CS3 and is an alternate approach to applying a shape tween. This mechanism happens to be smart enough to avoid mistakes, so if you try to apply a shape tween to something that doesn't support shape tweens, it will simply ignore your attempt. If you open this context menu on an already shape-tweened span, you'll see a choice of Remove Tween.*

7

## Shape tween modifiers

There are a couple ways to refine a shape tween once it's applied. These are shown in the Property inspector when you click in a tweened span of frames: Ease and Blend. We'll cover easing in greater detail in the "Motion tweening" section, but for now, here's the punch line: easing tends to make tweens look more lifelike because it gradually varies the amount of distance traveled between each frame.

If an astronaut throws a golf ball in outer space, the ball flies at a constant rate until . . . well, until it hits something. That's not how it works on a planet with gravity. The ball flies faster at first, and then gradually slows down. This deceleration is called **easing out**. A ball dropped from a tall building begins its descent slowly, and then gradually increases speed. This acceleration is called **easing in**. Adjust the Ease slider in the Property inspector to see how easing affects the shape tween applied to the pepper in the previous exercise. Supported values range from 100 (strong ease out), through 0 (no easing), to -100 (strong ease in). As shown in Figure 7-5, easing can have a profound effect upon an object in motion.



**Figure 7-5.** Examples of easing; from top to bottom: easing in, no easing, and easing out

Blend is a much subtler matter.

There are two blend settings: Distributive and Angular. According to Adobe, Distributive “creates an animation in which the intermediate shapes are smoother and more irregular,” while Angular “creates an animation that preserves apparent corners and straight lines in the intermediate shapes.” In actual practice, the authors find this distinction negligible, at best. In short, don’t worry yourself over this setting. Use it or not, but we’re willing to bet our hats that you won’t be able to tell one from the other.

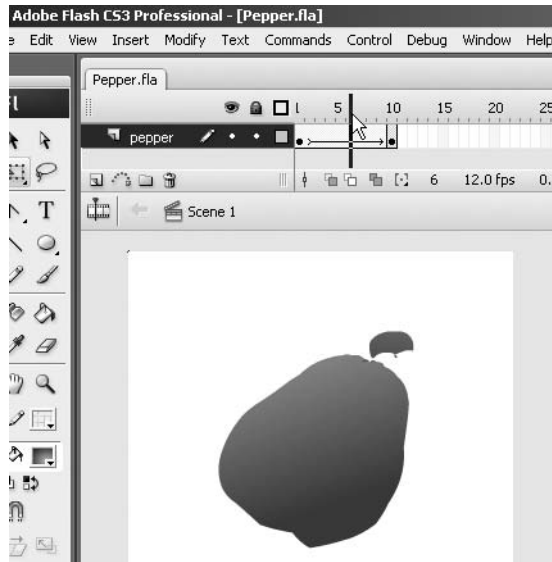
OK, so far, so good. These tweens have been pretty straightforward. In fact, as you’ll find later in the chapter, everything you’ve seen to this point can be accomplished just as easily with motion tweens. This raises a good question: What makes shape tweens so special? Why not just use motion tweens?

The answer comes in two parts: gradients and shape. Let’s tackle shape first, because it has the potential to set your teeth on edge if you aren’t prepared for it.

## Altering shapes

The compelling reason to use shape tweens is for their ability to manipulate the actual form of the artwork itself, beyond scaling and stretching. Let’s keep playing:

1. Continuing with `PepperShape.fla`, use the Free Transform tool at frame 10 to rotate the pepper about 90 degrees in either direction.
2. You should still have a shape tween applied—if not, add one—and then drag the playhead back and forth to see a result that may surprise you. Rather than rotating, the pepper temporarily deforms itself as it changes from one keyframe to another (see Figure 7-6).



**Figure 7-6.** Sometimes shape tweens perform unexpected transformations.



What on earth is going on here? Though it may look like an absolute mess, what you are seeing is the key distinction between shape tweening and motion tweening. Believe it or not, this behavior can be a very useful thing. We'll see an example in just a moment. First, let's take a quick field trip to frame 10 in order to illustrate a point.

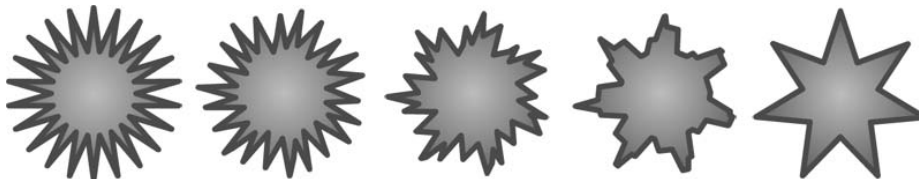
*In case you're worried, we'll put your mind at ease without further ado: it is entirely possible to rotate artwork with tweens in Flash. In fact, it's easy. In contrast to shape tweens, motion tweens always maintain a strict marriage between one keyframe's anchor points and the next. We'll show you why later in the chapter. When you understand what each approach does best, you'll know which one to use for the task at hand.*

3. Choose the Subselection tool and click in frame 10 of the PepperShape.fla file. You'll see dozens of tiny squares that act as anchor points among the various lines and curves that make up the pepper's shape. All those points exist in frame 1 as well, of course, but they're in different positions relative to one another.

With shape tweens, Flash does not think of artwork in terms of a whole; instead, it manipulates each anchor point separately. What seems like a rotation to you is, to a shape tween, nothing more than a rearrangement of anchor points—sometimes a chaotic one, at that!

Think of it like a square dance. If a particular point happens to be in the upper-left corner on frame 1, it has no idea that its corresponding point may be in the upper-right corner on frame 10. It simply changes a partner—do-si-do!—and moves to a new spot during the tween. Like square dancing, there are sophisticated rules at play, and movement across the dance floor may appear unpredictable. It's possible, for example, that two keyframes may even present a completely different number of anchor points. Let's look at that next.

1. Open the StarStar.fla file from the exercise files for this chapter, and take note of the 22-point star in frame 1. Use the Subselection tool, if you like, to see the individual anchor points (there are 44). Click in frame 20 to see a 7-point star (14 anchor points). Note that a shape tween has already been applied between these two keyframes. Drag the playhead back and forth to watch the promenade (shown in Figure 7-7). Flash handles the reduction in anchor points in a neat, organized way. In this case, by the way, the star in the second keyframe was drawn as new artwork into frame 20.

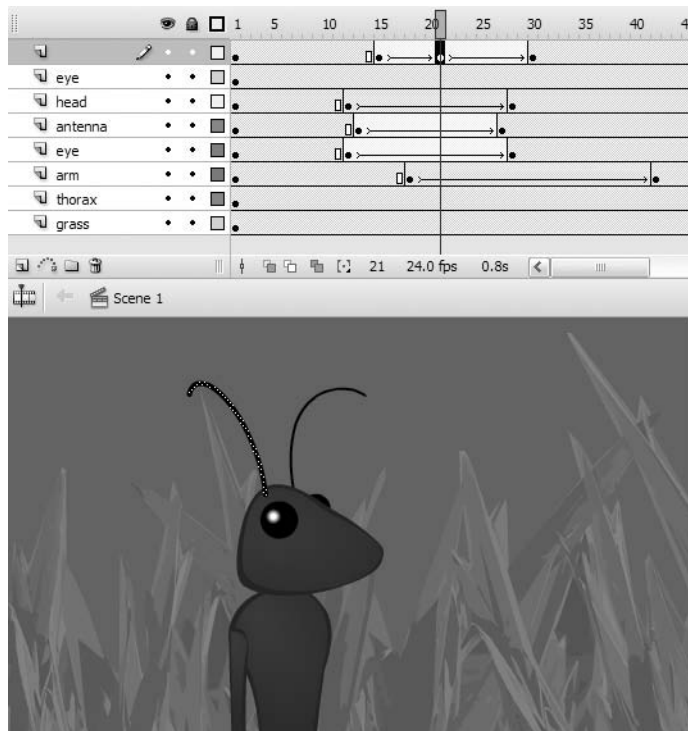


**Figure 7-7.** The 44 anchor points artfully become 14.

2. Open the StarCircle.fla file and run through the same steps to see a 22-point star become an 8-point circle. These are some nifty transformations that are simply not possible with motion tweens.

This opens up a whole avenue of vector-morphing possibilities, from sunshine gleams to water ripples to waving hair and the antennae shown in Figure 7-8. For anything where you need the actual *shape* of an item to change—where anchor points themselves need to be rearranged—shape tweens are the way to go. Keep in mind that tweens happen on a keyframe basis, and timeline layers are distinct. If you have a complex set of shapes and you only wish to tween some of them, move those shapes to a separate layer. In fact, you may want to put every to-be-tweened shape on its own layer because that reduces the number of anchor points under consideration for each keyframe. Let's try it by setting some antennae in motion:

1. Open Ant.fla and insert a keyframe in frame 21 of the antenna2 layer.
2. Select the Subselection tool and change the shape of the antenna in the layer.
3. Add a shape tween between the keyframes and scrub through the timeline. The antennae move around (see Figure 7-8).



**Figure 7-8.** Need to change the shape of those antennae? Shape tweens to the rescue!

As we've seen, Flash can make some fairly stylish choices of its own in regard to the repositioning of anchor points. Well, most of the time. The earlier pepper rotation demonstrates that Flash's choices aren't always what you might expect. Fortunately, Flash provides a way to let you take control of shape tweens gone awry. The solution is something called shape hints.

## Shape hints

What are shape hints? Though often overlooked and misunderstood, these useful contraptions allow you to specify a partnership between a region of your choosing from one keyframe to the next. They are a means by which you can guide an anchor point, curve, or line toward the destination you've determined is the right one. Let's take a look.

1. Open the LogoMorphNoHints.fla file from the samples for this chapter. Take a look at frame 1 to see a lowercase *i* that has been broken apart from a text field into two shapes. In frame 55, you'll see an abstract shape that represents a hypothetical logo. The aim here is to morph between the shapes in an appealing way, but something has gone horribly wrong (see Figure 7-9). Drag the playhead along the timeline and note the atrocities committed between frames 20 and 35.



Figure 7-9. Something has gone horribly wrong.

This looks as bad as (if not worse than) the hot pepper rotation . . . but why? On the face of it, this should be a basic shape tween. Seemingly, the letter and logo shapes aren't especially intricate, and yet . . . the timeline doesn't lie.

*At this point, the authors look deftly side to side, and with a sly, "Hey, pssst," invite you to step with them into a small, dimly lit alley. (Don't worry, we're here to help.) "The thing is," begins the first, "honestly, there's often a bit of voodoo involved with shape tweens, and that's the truth." "That's right," chimes in the other, lowering his voice. "To be frank, if I may"—you nod—"we don't know why these anchor points sometimes go kablooney. It's just a thing, and you have to roll with it." There is a slight pause, and suddenly a cappuccino machine splooshes in the distance. The first author draws a finger across his nose. "Keep that in mind as we continue," he says. Another pause. "You wanna see the shape hints?" You nod again.*

2. Click in frame 20, and select **Modify** ► **Shape** ► **Add Shape Hint** (see Figure 7-10). This puts a small red circle with the letter *a* in the center of your artwork. Meet your first shape hint.

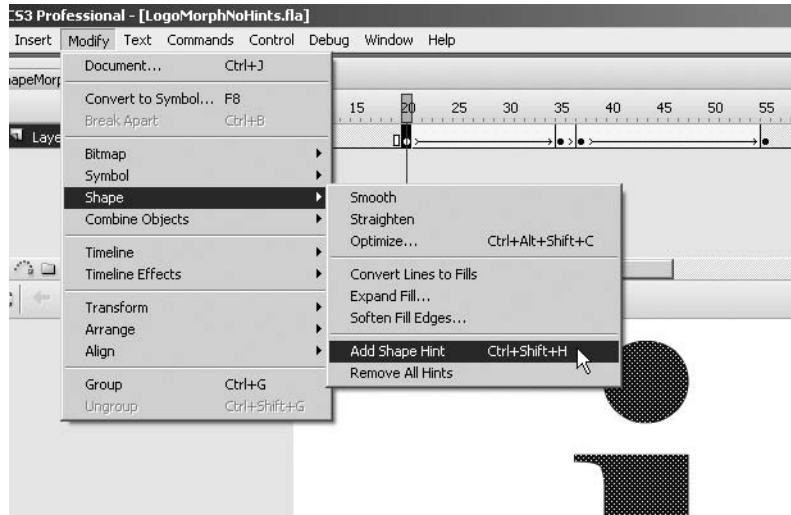


Figure 7-10. Inserting a shape hint

3. Make sure object snapping is on, either by selecting Snap to Objects in the Tools panel or ensuring that a check mark is present under View > Snapping > Snap to Objects. Snapping significantly helps the placement of shape hints. Drag and snap the a circle to the lower-left corner of the letter's upper serif, as shown in Figure 7-11.

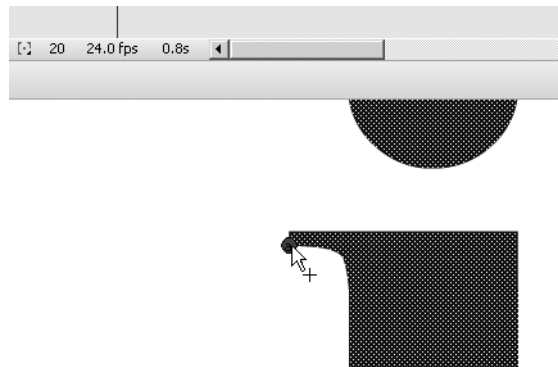
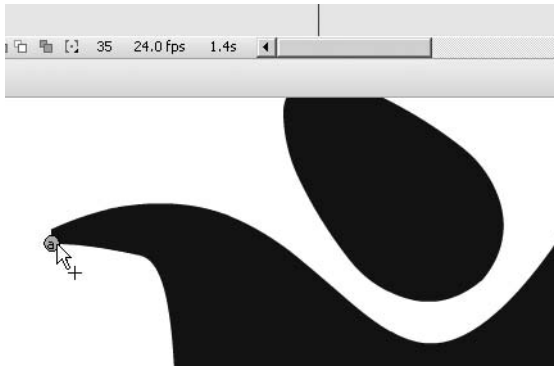


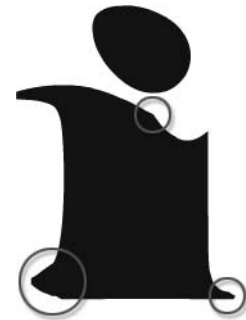
Figure 7-11. Positioning a shape hint

4. This next point is important: what you've done is placed *one half* of a shape hint *pair*. The other half—the partner—is on the next keyframe, frame 35. Drag the playhead to this frame and position the second a circle on the corresponding serif on this keyframe's shape, as shown in Figure 7-12.



**Figure 7-12.** Positioning the shape hint's partner

5. When this partner snaps into place, it will turn green. Return to frame 20 and notice that the original shape hint has turned yellow. It may be that shape hints have a thing for stoplights (not that there's anything wrong with that), but the point is that the color change indicates something. It tells you that this shape hint pair has entered into a relationship. You have now indicated to Flash your intention that these paired regions correspond.
6. Slide the playhead along the timeline again, and you'll see a remarkable improvement (as shown in Figure 7-13). So remarkable, in fact, that the authors look deftly side to side, wink, and silently mouth the word *voodoo*. To be frank, if we may, the placement of shape hints often makes a noticeable difference, but the decision on placement is something of a dark art. We encourage you to reposition your first shape hint pair at other corners to see how the remaining trouble spots ripple to other areas.
7. You should get the idea by now that shape hints are a bit like cloves (you know, the star-shaped things you poke into your ham during the holidays)—a little goes a long way. Let's add a few more, but do so sparingly. To get rid of the kink in the upper curve, add a new shape hint to the upper-right corner of the *i* on frame 20. This time, you'll see a small *b* in a red circle. Snap its *b* partner to the upper-right corner of the logo at frame 35, and drag the playhead again to see your progress.
8. Add shape hints *c* and *d* to the lower-left and right corners, and you should see a very smooth morph along this span of frames. The only thing remaining, if you're a perfectionist, is a slight wrinkle along the bottom of the "egg" between keyframes 37 and 55. Remedy this by adding a new shape at frame 37—it will start again at *a*, because this is a new pair of keyframes—and snap it in place to the corresponding curve at frame 55.

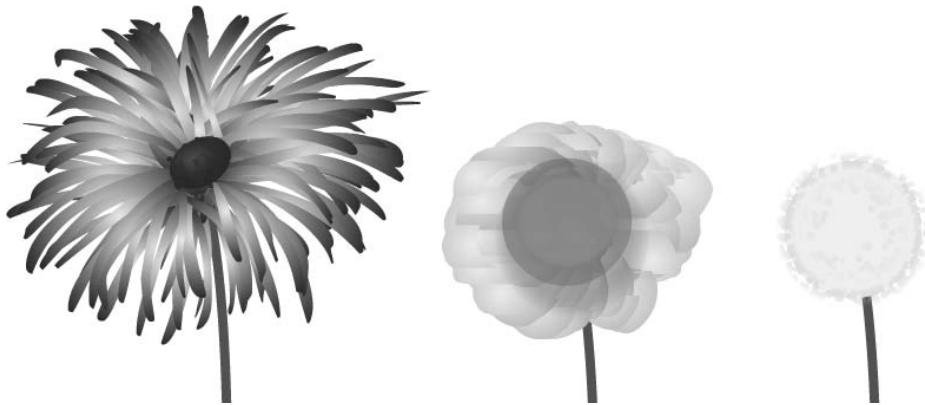


**Figure 7-13.** A dramatic improvement, but there are still a few trouble spots

Compare your work with the `LogoMorph.fla` file, if you like. When you open a file that already contains shape hints, you'll need to take one small step to make them show, as they like to hide by default. To toggle shapes hints on and off, select `View > Show Shape Hints`.

Even with the benefit of shape hints, we caution you to keep simplicity in mind. Certain collections of shapes are simply too intricate to handle gracefully. It is entirely possible to choke Flash through the use of an overwhelming number of anchor points, as shown in Figure 7-14.

1. Open the `FlowerWeed.fla` file and drag the playhead along the timeline. The morph isn't especially polished, but it certainly doesn't count as a complete eyesore.
2. Test the SWF (`Control > Test Movie`), and you'll see that playback slows nearly to a halt as the tweening progresses. No amount of shape hinting can fix that.

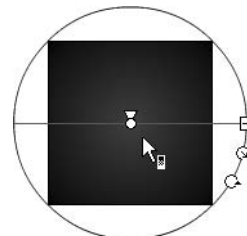


**Figure 7-14.** Moderation in all things! While this transformation doesn't look awful, it nearly chokes Flash Player.

## Altering gradients

If you want to animate gradients, shape tweens are the only way to do it. You may not immediately think of gradients as shapes, but when you select the Gradient Transform tool and click into a gradient, what do you see? You see the handles and points shown in Figure 7-15.

That center point, to Flash, is not much different from an anchor point. The resize, radius, and rotate handles are not much different from Bezier control point handles. In effect, you are manipulating a shape—just a special kind. When animating a gradient, you simply change these gradient-specific features from keyframe to keyframe, rather than a shape's corners, lines, and curves.

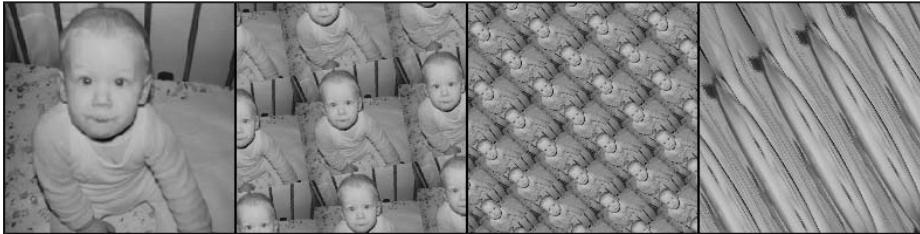


**Figure 7-15.** Gradients, like anything else, can be edited on keyframes, and those keyframes are tweenable.

1. Open the GradientTween1.fla file and drag the playhead along the timeline to see an example in action. Frame 1 contains a solid red fill. Frame 10 contains the built-in rainbow gradient, which is rotated 90 degrees in frame 20.

Frames 20 through 30 provide a bit of interest because they spell out a limitation of gradient shape tweens: it is not possible to tween one type of gradient to another. Well, we take that back. You certainly can, but the results are unpredictable. Flash tries its best to convert a linear gradient into a radial one, but between frames 29 and 30, the gradient pops from one type to the other.

2. Open the GradientTween2.fla file. This example shows a combination of gradient and shape change at the same time. Not only does the gradient fill transform, but anchor points move, and even stroke color (and thickness!) changes from keyframe to keyframe.
3. Experiment with solid colors as well as the Color panel's Alpha property. When you finish, close the file without saving the changes.
4. Even bitmap fills are tweenable, which, as shown in Figure 7-16, makes for some interesting visual possibilities. Open the BitmapFillTween.fla file and press the Enter/Return key. As with other types of gradients, use the Gradient Transform tool to manipulate gradient control handles at each keyframe, and then let the shape tween handle the rest. Easing works the same way.



**Figure 7-16.** Shape tween your bitmap fill transformations for some real zing!

## Motion tweening

When we left that hapless hot pepper hanging, it had been hoping to rotate. It didn't, and instead found its molecules tumbling in a frenzied jumble. We told you there was a much easier way to handle that rotation, and motion tweening is it. Shape tweens are for rearranging anchor points and animating gradients; motion tweens are for everything else, from enlivening text and imported photos to animating vector artwork drawn directly in Flash or imported from another application like Illustrator CS3 or Fireworks CS3.

In contrast to shape tweens, motion tweens require self-contained entities. These include symbols, primitives, drawing objects, and grouped elements, which many designers find easier to work with than raw shapes. Open PepperSymbol.fla, for example, and you'll see that it's easier to select the whole pepper without accidentally omitting the cap.

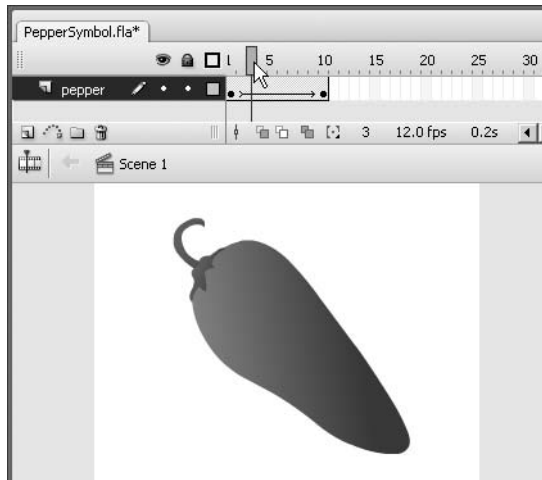
*Be aware that primitives and drawing objects blur the lines somewhat between what constitutes a shape and what constitutes a symbol. It is possible to apply both shape tweens and motion tweens to primitives and drawing objects, but many properties such as color, alpha, and the like—and in primitives, shape—are only properly animated with shape tweens. These “gotchas” tend to steer the authors toward a path of least resistance: use shapes for shape tweens and symbols for motion tweens. Within those symbols, use whatever elements you like.*

*One fundamental point: When it comes to motion tweens, always put each tweened symbol on its own layer. If you apply a motion tween to keyframes that contain more than one symbol, Flash will try to oblige—but will fail. It’s a simple rule, so abide by it and you’ll be happy.*

## Rotation

Let’s pick up with that rotation, shall we?

1. Open the PepperSymbol.fla file. This time, you’ll see a pepper symbol in the library because the shapes from the earlier PepperShape.fla have been placed inside a graphic symbol. Add a keyframe in frame 10. Select the Free Transform tool and rotate the artwork 90 degrees in either direction on that second keyframe. Sounds familiar, right? Here comes the difference.
2. Select Motion from the Tween property in the Property inspector. There it is! Drag the playhead back and forth to see a nice, clean rotation of the pepper. As you saw with shape tweens, the span of frames between the two keyframes changes color—this time, as shown in Figure 7-17, to purple—and a solid arrow appears within the span to indicate a successful tween.

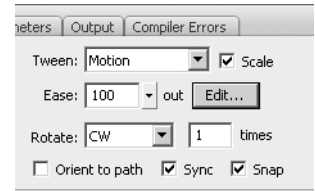


**Figure 7-17.** Motion tweens, indicated by an arrow between the keyframes, make rotations a snap.



3. Change the Tween property to Shape and the span of frames turns green, the color of shape tweens—but the solid arrow becomes a dashed line, indicating a failed tween. Change the Tween back to Motion and everything's right with the world.
4. Now, let's think about *real* rotation; topsy-turvy; a full 360 degree spin. How would you do it? (Hint: This is something of a trick question.) In a full spin, the pepper ends up in the same position at frame 10 as it starts with in frame 1, so there's not really a transformation to tween. Enter the Rotate drop-down menu in the Property inspector.

Notice that the Rotate setting is currently Auto. This is because you have already rotated the pepper somewhat by hand. Click the pepper in frame 10 and select Modify ► Transform ► Remove Transform to reset the symbol's rotation. In the Rotate drop-down menu, change the setting to CW (clockwise), as shown in Figure 7-18, and drag the playhead back and forth. Pretty neat! CCW (counterclockwise) rotates the tweened symbol in the opposite direction, and the text field immediately to the right specifies how many times to perform the rotation.



**Figure 7-18.** The Rotate property makes quick work of rotations.

## Motion tween properties

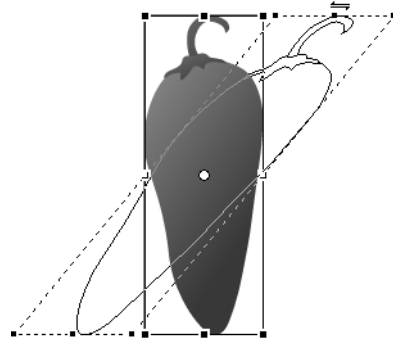
While we're looking at the Property inspector, let's go through the other settings. Here's a quick overview of motion tween properties:

- **Tween:** This one should already be familiar. The choices are None, Motion, and Shape.
- **Scale:** If a check mark is present, tweening for the current span of frames will include a transformation in scale (size), *if such a transformation exists*. If you haven't scaled anything, it doesn't matter what state the check mark is in. If scaling and other transformations are combined in a given tween, only the other transformations will show if the check mark is vacant.
- **Ease and Edit:** These settings apply a range of easing to the tween. The Edit button allows for advanced, custom easing. More on this in the "Easing" section of this chapter.
- **Rotate, [number of] times, and Orient to path:** These settings control the type of rotation and the number of times the rotation occurs. Only CW and CCW support the [number of] times setting. The Orient to path setting only applies to tweens along a motion guide (discussed later in the chapter).
- **Sync:** In our experience, most people don't even realize this property exists, but it can be a real time saver when you're dealing with graphic symbols. Unlike movie clips, which have their own independent timelines, graphic symbols are synchronized with the timeline in which they reside. Even so, there is a bit of flexibility: graphics can be looped, played through once, or instructed to rest on a specified frame of their own timeline. If a particular graphic symbol has been tweened numerous times in a layer, the presence of the Sync check mark means you can update these timeline options for all keyframes in that layer simply by making changes to the first graphic symbol in the sequence. In addition, Sync allows you to swap one graphic symbol for another and have that change ripple through all the synced keyframes in that layer.
- **Snap:** This option helps position a symbol along its motion guide (motion guides are discussed later in the chapter).

## Scaling, stretching, and deforming

We visited this topic in the “Shape tweening” section, and honestly, there’s not a whole lot different for motion tweens. The key thing to realize is that scaling, stretching, and deforming a symbol is like doing the same to a T-shirt with artwork printed on it. Even if the artwork looks different after all the tugging and twisting, it hasn’t actually changed. Shake it out, and it’s still the same picture. Shape tweening, in contrast, is like rearranging the tiles in a mosaic. For this reason, the Free Transform tool disables the Distort and Envelope options for symbols. These can’t be performed on symbols and therefore can’t be motion-tweened. Let’s take a quick look at the other options:

1. Return to the `PepperSymbol.fla` file and set the Rotation setting for the tween to None. Use the Free Transform tool to perform a shear transformation at frame 10. Shear? What’s that? Something you do with sheep, right? Well, yes, but in Flash, shearing is also called skewing, which can be described as tilting. With the Free Transform tool active, click the Rotate and Skew option, and then hover over one of the side transform handles (not the corners) until the cursor becomes an opposing double-arrow icon. Click and drag to transform the pepper (see Figure 7-19).



**Figure 7-19.** Motion tweening a symbol transformation

The outline preview gives you an idea what the symbol will look like before you let go of the mouse. Note that the skew occurs in relation to the transformation point, indicated by the small white circle. Drag this white circle around inside or even outside the bounding box of the pepper and skew again to see how its placement affects the transformation. Hold down `Alt` while skewing to temporarily ignore the transformation point and skew in relation to the symbol’s opposite edge.

2. We’ve been using the Free Transform tool quite a bit, so let’s try something different. Open the Transform panel (`Window > Transform`) and note its current settings. You’ll see the skew summarized near the bottom and, interestingly, the change in scale summarized near the top (see Figure 7-20).

From this, it becomes clear that skewing affects scale when applied with the Free Transform tool. To see the difference, select `Modify > Transform > Remove Transform` to reset the symbol. The scale area of the Transform panel returns to 100% horizontal and 100% vertical. Click the Skew radio button and type 38 into either one (but only one) of the skew input fields. Press `Enter/Return` to apply the change. Now enter 200 into the scale input fields at the top (the `Constrain` check mark means you only have to enter this number into one of them), and again press `Enter/Return` to apply the change. Slide the playhead back and forth to see two transformations tweened at once.

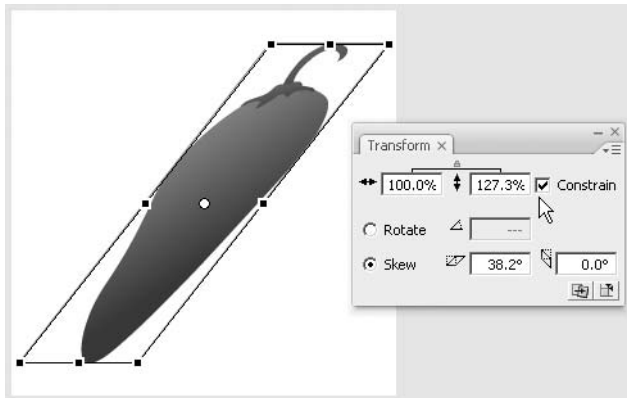


Figure 7-20. The Transform panel provides access to precision measurements.

## Easing

Here's where motion tweening begins to pull ahead of shape tweening. Easing is much more powerful for motion tweens, thanks to the Custom Ease In/Ease Out editor. Before we delve into that, though, let's look at a sample use of the standard easing controls for a motion tween, so you can see how much easier things are with the custom variety.

1. Open the MalletNoEasing.fla file. You'll see a hammer graphic symbol in the library and an instance of that symbol on the stage. Select the hammer and note that the transformation point—the white dot in the handle—is located in the center of the symbol.

We're going to make this hammer swing to the left, so select the Free Transform tool. Selecting this tool makes the transformation point selectable. Click and drag that point to the bottom center of the mallet (see Figure 7-21).

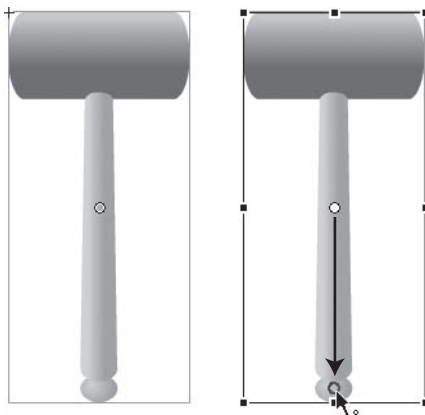
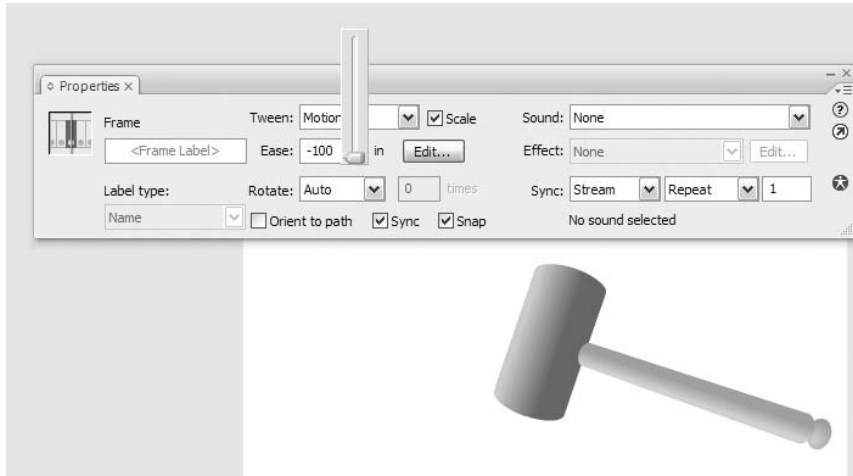


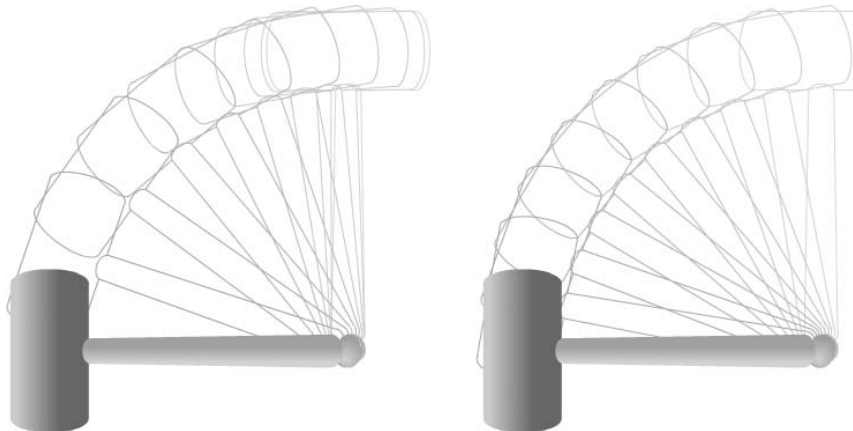
Figure 7-21. You'll have to move that transformation point to make the movement realistic.

2. Insert a keyframe at frame 10 (Insert ► Timeline ► Keyframe), and rotate the mallet at frame 10 to the left by 90 degrees. Apply a motion tween to the span of frames between 1 and 10, and scrub the timeline to see the effect. Not bad, but not especially realistic. How about some easing and bounce-back?
3. Drag the Ease slider all the way down to supply an ease of -100 (full ease in) to the tween, as shown in Figure 7-22.



**Figure 7-22.** The Ease slider determines how the hammer falls.

This means that the hammer falls slowly as it begins to tip and increases speed as it continues to fall (see Figure 7-23).



**Figure 7-23.** Ease in (left) vs. no easing (right). On the left, the hammer falls in a more natural manner.

4. This is a good start. To push the realism further, let's embellish the animation. Add new keyframes at frames 15, 20, 23, and 25. We're going to provide some tweening that makes the hammer rebound on impact and bounce a few times. At frame 15, use the Free Transform tool or the Transform panel to rotate the hammer to approximately northeast; in the Transform panel, this could be something like  $-55$  in the Rotate area. At frame 23, set the rotation to roughly east-northeast (something like  $-80$  in the Transform panel). A storyboard version of the sequence might look like Figure 7-24.

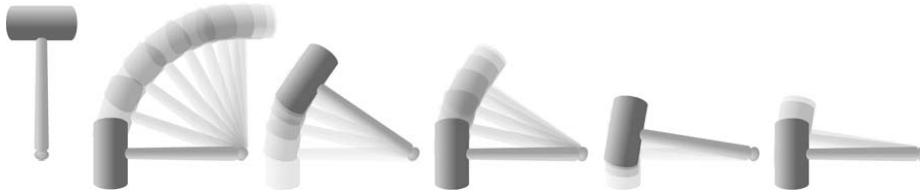


Figure 7-24. Using several keyframes to make the hammer bounce.

*The fading image trails—visual echoes of the mallet—are the result of something called onion skinning—very helpful in animation work. It's used here for illustrative purposes and is covered later in the chapter.*

5. Now that the mallet has been positioned, it just needs to be tweened and eased. You can either click separately into each span of frames and apply a motion tween, or click and drag across as many spans as you need (as shown in Figure 7-25). That way you can apply the tweens all in one swoop.

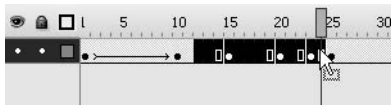


Figure 7-25. Tweens can be applied to more than one frame span at a time.

6. Finally, click into each span of frames to apply easing, for the final touch. Span 1 to 10 already has  $-100$ . Apply the following easing to the remaining spans:
- **Span 10 to 15:** 100 (full ease out)
  - **Span 15 to 20:**  $-100$  (full ease in)
  - **Span 20 to 23:** 100
  - **Span 23 to 25:**  $-100$

Drag the playhead back and forth to preview the action, and then test the movie to see the final presentation. If you like, compare your work with `MalletNormalEase.fla`. This exercise wasn't especially hard, but wouldn't it be even cooler if you could perform all of the above with a single motion tween?

## Custom easing

Introduced in Flash 8, the Custom Ease In/Ease Out dialog box unleashes considerably more power than traditional easing. Not only does it provide a combined ease in/out—where animation gradually speeds up *and* gradually slows down, or vice versa—it supports multiple varied settings for various kinds of easing, all within the same tween. Let's take a look.

To perform custom easing, you have to first open the Custom Ease In/Ease Out dialog box. To get to the dialog box, select a span of motion-tweened frames, and then click the Edit button in the Property inspector. The result is a graph with time in frames along the horizontal axis and percentage of change along the vertical axis (shown in Figure 7-26).

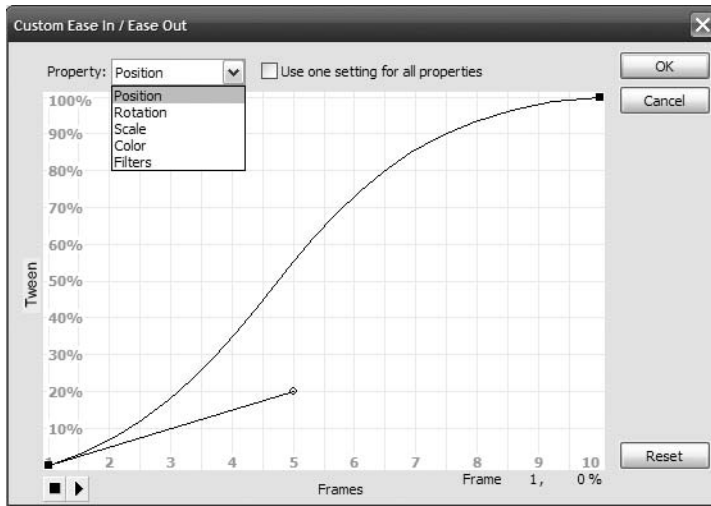


Figure 7-26. The Custom Ease In/Ease Out dialog box

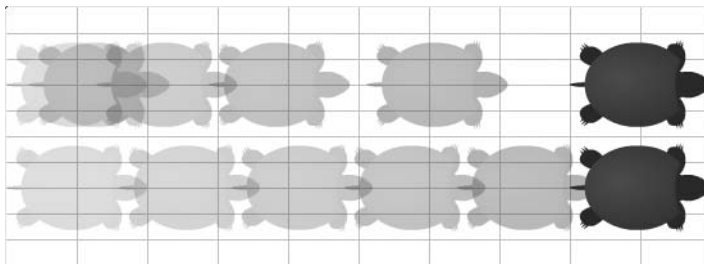
Here's a quick rundown of the various areas of the dialog box:

- **Property:** By default, this is disabled until you deselect the check mark next to it. If the check mark is present, custom easing—as specified by you on the grid—applies to all aspects of the tween symbol. If the check mark is absent, this drop-down menu lets you distinguish among Position, Rotation, Scale, Color, and Filters.
- **Use one setting for all properties:** When checked, this allows multiple properties to be eased individually.
- **Grid:** The Bezier curves on this grid determine the visual result of the custom easing applied.

- **Preview:** Click the two buttons in this area to play and stop a preview of the custom easing.
- **OK, Cancel, and Reset:** The OK and Cancel buttons apply and discard any custom easing. Reset reverts the Bezier curves to a straight line (no easing) between the grid's opposite corners.

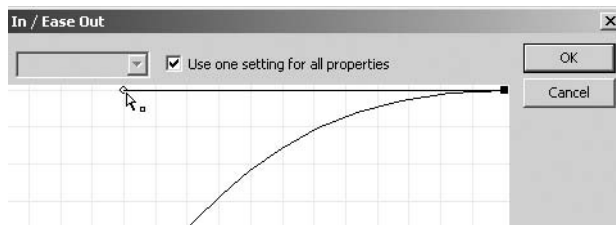
So, how does the grid work? Let's look at a traditional ease in to see how the Custom Ease In/Ease Out dialog box interprets it.

1. Open `CustomEasingComparison.fla` and set the Ease property to `-100` (a normal full ease in) for the tween in the top layer. Scrub the timeline to confirm that the upper symbol starts its tween more slowly than the lower one, but speeds up near the end. The lower symbol, in contrast, should advance the same distance each frame (see Figure 7-27).



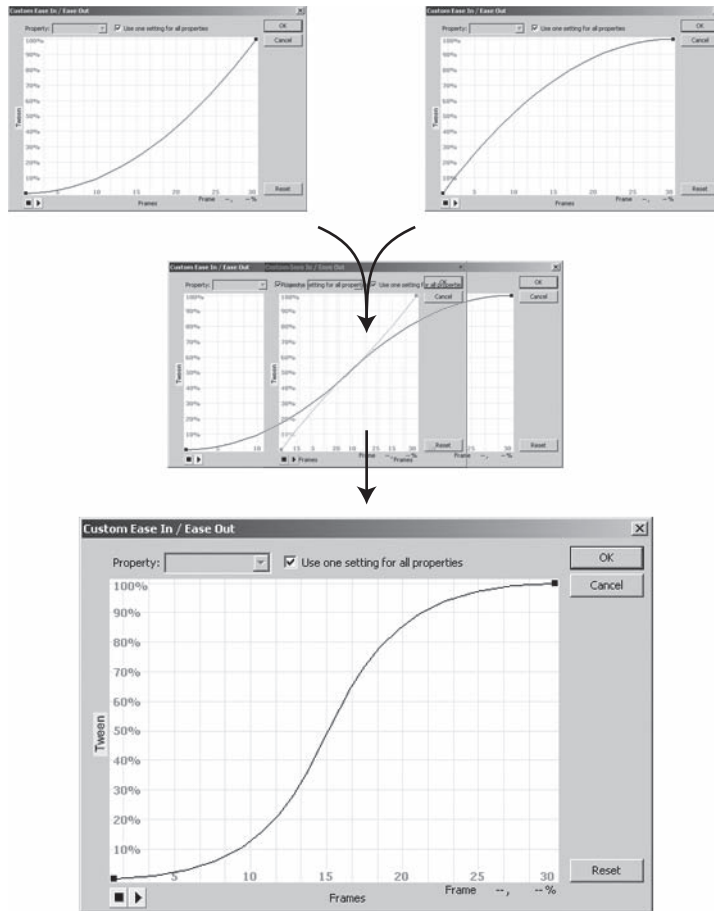
**Figure 7-27.** An ease in causes the upper symbol to start slower and speed up.

2. Click the Edit button to see what an ease out looks like on the grid. The curve climbs the vertical axis (percentage of change) rather slowly, and then speeds its ascent near the end of the horizontal axis (time in frames). Hey, that makes sense! Click Cancel, apply a full ease out (100), and then check the grid again . . . bingo, the opposite curve.
3. It follows that a combination of these would produce either a custom ease in/out (slow, fast, slow) or a custom ease out/in (fast, slow, fast). Let's do the first of those two. Click the upper-right black square in the grid to make its control handle appear. Drag it up to the top of the grid and about two-thirds across to the left, as shown in Figure 7-28.



**Figure 7-28.** Dragging a control handle to create a custom ease

- Click the bottom-left black square and drag its control handle two-thirds across to the right. The resulting curve—vaguely an S shape—effectively combines the curves you saw for ease in and ease out (see Figure 7-29).



**Figure 7-29.** An S shape produces an ease in/out (slow-fast-slow) tween.

- Click OK to accept this setting, and scrub the timeline or test the movie to see the results.
- Let's inverse this easing for the lower symbol. Select the lower span of frames and click the Edit button. This time, drag the lower-left control handle two-thirds up the left side. Drag the upper-right control handle two-thirds down the right side to create the inverted S curve shown in Figure 7-30. Click OK and compare the two tweens.



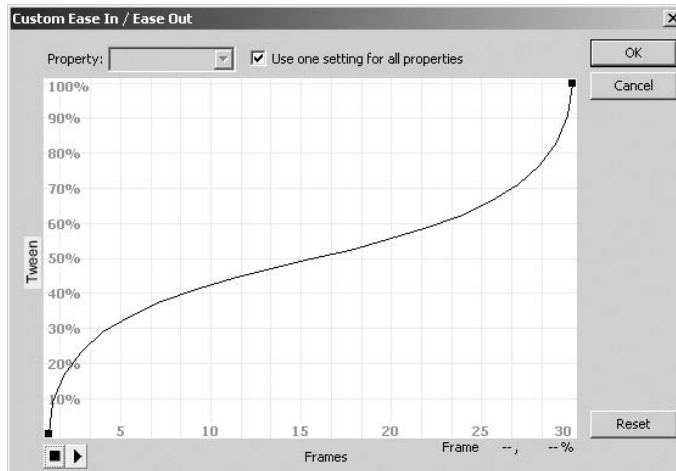


Figure 7-30. An inverted S shape produces an ease out/in (fast-slow-fast) tween.

Think this is cool? We're just getting started! By clicking anywhere along the Bezier curve, you can add new anchor points. This is where you can actually save yourself a bit of work.

1. Open `MalletNoEasing.fla` again. If you saved your work earlier, remove the tween and delete all frames except for frame 1. Use the mouse to click and drag from frame 2 to the right until you've selected them all, and then use `Edit > Timeline > Remove Frames`. Confirm that the mallet's transformation point is positioned at the bottom center of its wooden handle. Now add a new keyframe at frame 25 and apply a motion tween to the span of frames between 1 and 25.
2. Using the Free Transform tool at frame 25, rotate the mallet 90 degrees to the left. This may seem like *déjà vu*, but things are about to change. Because a tween is already applied, you can preview the falling mallet by scrubbing the timeline. Click in the tweened span of frames and click the `Edit` button in the Property inspector. We're going to emulate the same bounce-back tween we did earlier, but this time we're going to do it all in one custom ease.
3. When the Custom Ease In/Ease Out dialog box opens, click the Bezier curve near the middle and you'll see a new anchor point with control handles. Hold down `Shift` and click that new anchor point—it disappears. Add it again and straighten the control handles so that they're horizontal (as shown in Figure 7-31).
4. Repeat this process three more times, up the hill, as shown in Figure 7-32. This prepares the way for the sawtooth shape you'll create in the next step.

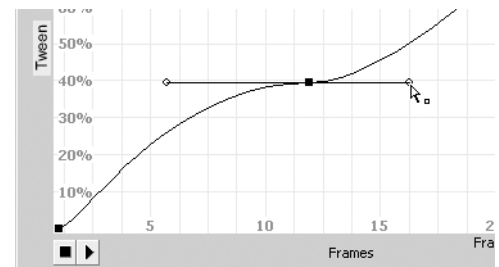


Figure 7-31. Starting a more complex custom ease

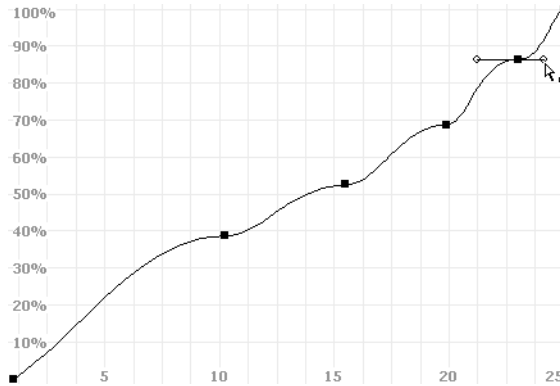


Figure 7-32. Continuing to add anchor points for a sawtooth curve

5. Leave the corner anchor points where they are. Position the four new anchor points as follows:

- 100%, 10
- 60%, 15
- 100%, 20
- 85%, 23

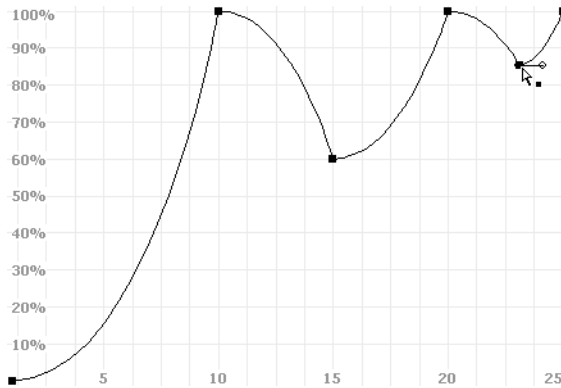
*You'll notice that the anchor points gently snap to the grid while you drag. To temporarily suppress this snapping, hold down the X key.*

6. You've probably heard of certain procedures described as more of an art than a science . . . well, we've come to that point in this step. Here's the basic idea, but it's up to you to tweak these settings until they feel right to you. To achieve the sawtooth curve we're after—it looks very much like the series of shark fins shown in Figure 7-33—click each anchor point in turn and perform the following adjustment:

- If it has a left control handle, drag that handle in toward the anchor point.
- If it has a right control handle, drag that handle out a couple of squares to the right.

You should get something like the shape shown in Figure 7-33.

7. Click the Preview play button to test your custom ease. It should look similar to the original series of mallet bounce-back tweens, only now you've saved yourself a handful of keyframes. How does this work? As depicted in the grid, and following the horizontal axis, you have an ease-in curve from frames 0 to 10, an ease-out curve from 10 to 15, an ease-in curve from 15 to 20, and so on—just like your series of keyframes from earlier in the chapter. The mallet moves from its upright position to its leaned-over position in the very first curve. From frames 10 to 15, the vertical axis goes from 100% down to 60%, which means that the mallet actually rotates clockwise again toward its original orientation, but not all the way. With each new curve, the hammer falls again to the left, and then raises again, but never as high. Compare your work with MalletCustomEasing.fla.



**Figure 7-33.** Shark fins produce a bounce-back effect.

On the final leg of our custom easing expedition, let's pull out all the stops and examine a tween that updates multiple symbol properties at once. You'll be familiar with most of what you're about to see, and the new parts will be easy to pick up.

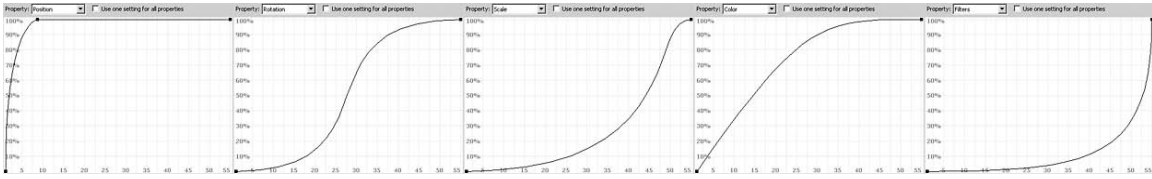
1. Open the CustomEasingMultiple.fla file. Select frame 1 and note that a movieclip symbol appears in the upper-left corner of the stage. It is solid green. Select frame 55 and note the changes. At this point, the apple is positioned in the center of the stage, much larger, more naturally colored, and has a drop shadow (see Figure 7-34).

From this, we can surmise that color and filters are tweenable—that's the new part—and in fact, they are. In frame 1, select the apple symbol itself to see that a Tint has been applied in the Property inspector, which is replaced by None in the other keyframe. Likewise, select the Filters tab at frame 55 and click the apple to see that a drop shadow has been applied that is not present in frame 1. These properties are no different from position and scale as far as tweens are concerned.



**Figure 7-34.** You are about to discover that it isn't only rotation that can be tweened.

2. Click into the span of tweened frames and note that a CW (clockwise) rotation has been specified for Rotation. The Tween type is Motion, and Scale is enabled (without it, the apple wouldn't gradually increase in size). The Ease property reads ---, which means custom easing has been applied. That's what we're after. Click the Edit button.
3. Thanks to the empty Use one setting for all properties check box, the Property drop-down menu is now available. Use the drop-down menu to look at the grid curve for each of five properties, all of which are depicted in the tween: Position, Rotation, Scale, Color, and Filters. Each curve has its own distinct curve, which translates into five individual custom ease settings for their respective properties (see Figure 7-35).



**Figure 7-35.** The Custom Ease In/Ease Out dialog box lets you specify distinct easing for five different tweenable properties.

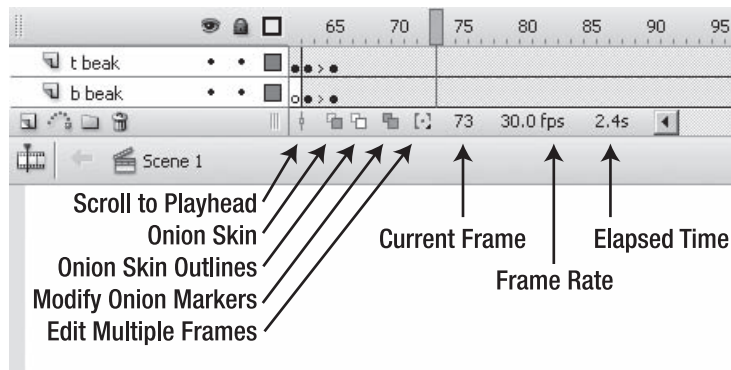
Click the check box to disable the drop-down menu. Ack! Have you lost your custom settings? Thankfully, no. Flash remembers them for you, even though they're hiding. Click the *Preview* play button to preview the tween with no easing (the default lower-left to upper-right curve). Click the check box again to see that the custom ease settings are still intact. Preview the tween again, if you like.

## Using animation

To this point, we've shown you a hefty animation toolbox. We've opened it up and pulled out a number of powerful tools to show you how they work. In doing so, we've covered quite a bit of ground, but there are still a handful of useful features and general workflow practices to help bring it all together. Let's roll up our sleeves, then, shall we?

## A closer look at the Timeline panel

Whether you use shape or motion tweens, the Timeline panel gives you a pint-sized but important dashboard to take advantage of while you work. Don't let its small size fool you. This strip (shown in Figure 7-36) along the bottom of the timeline lets you quickly find your bearings, gives you at-a-glance detail on where you are, and even lets you time travel to see where you've been—into both the past and the future.



**Figure 7-36.** The bottom edge of the timeline provides a collection of useful tools.

“OK, guys,” you may be thinking, “Time travel? Explain that one.” We will, but first let’s take an inventory of this useful, if small, real estate.

- **Scroll to Playhead:** In timelines that are long enough to scroll, this button centers the timeline on the playhead.
- **Onion Skin and Onion Skin Outlines:** These toggle two different kinds of onion skinning, which give you a “time machine” view of your work.
- **Modify Onion Markers:** Click this and you get a drop-down menu that controls the functionality of the onion skin buttons.
- **Edit Multiple Frames:** This allows you to select more than one keyframe at the same time, in order to edit many frames in one swoop.
- **Current Frame:** This indicates the current location of the playhead.
- **Frame Rate:** This indicates the movie’s frame rate. Double-click this setting to change it.
- **Elapsed Time:** Given the current frame and the movie’s frame rate, this indicates the duration in seconds of the playhead’s position. For example, in a movie with a frame rate of 24 fps, this area will say 1.0s at frame 24.

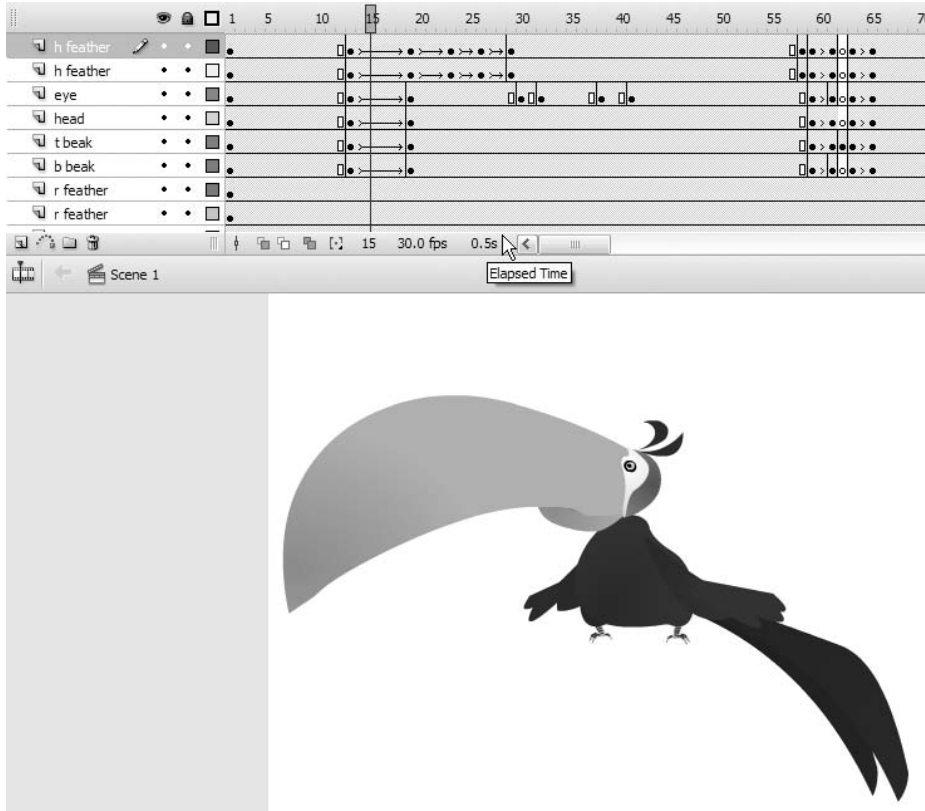
## Onion skinning

7

Traditional animators—the people who brought us the Mickey Mouse and Bugs Bunny cartoons we all grew up on—often drew their artwork on very thin paper over illuminated surfaces called lightboxes. This “onion skin” paper allowed them to see through the current drawing to what had gone on in the previous frames. In this way, they could make more informed choices on how far to move someone’s head . . . or the anvil about to fall on it.

Flash offers you the same benefit, but with much more flexibility. In Flash, you can choose to see through as many frames as you like—backward and even forward—in solids or in outlines.

1. Open the `YawningParrot.fla` file that accompanies this chapter. Note that the movie’s frame rate is 30 fps. Drag the playhead to frame 15, just as the bird begins to lower its head, and confirm that the Elapsed Time indicator reads 0.5s (see Figure 7-37). This makes sense: 15 divided by 30 is 0.5. Double-click the Frame Rate indicator to open the Document Properties dialog box. Change the movie’s frame rate to 60 fps and click OK. Note that the elapsed time is 0.2 seconds (still good: 15 divided by 60 is 0.2—if you don’t round up). One last observation: Change the frame rate to 15 fps and check the Elapsed Time indicator. You were probably expecting 1.0s, but the answer is a very close 0.9s. Why the discrepancy? We aren’t sure, but it is close enough to the original value to satisfy us. Change back to the original 30 fps.



**Figure 7-37.** Another really good reason this is called the timeline

2. Drag the playhead to the right far enough that the timeline starts to scroll a bit, and then leave the playhead where it is. Use the timeline’s scrollbar to scroll back to the left, which hides the playhead. To quickly bring it back, click the Scroll to Playhead button, which centers the timeline on the current frame. This is a good “you are here” panic button that’s useful for especially long timelines.
3. Position the playhead at frame 125 and click the Modify Onion Markers button. Choose Onion 5 from the drop-down menu. This positions two new markers on either side of the playhead, as shown in Figure 7-38.



**Figure 7-38.** Onion skinning adds two markers on either side of the playhead.

These markers extend five frames back and forward from the current position, which explains the name of the Onion 5 setting. What they show are semitransparent views of those frames fading as they get farther from the playhead—just like artwork on thin paper! Not only do they let you see back in time at previous frames, they also show artwork on future frames, which provides practical sequential context for any moment in time. In this case, you’re seeing 11 “sheets”; the one under the playhead (which is the darkest), and then five ahead and behind.

- Click **Modify Onion Markers** again and choose **Onion 2**, as shown in Figure 7-39. This reduces your view to five “sheets.” Drag the playhead slowly to frame 170 and back. Notice that the onion markers move with you.



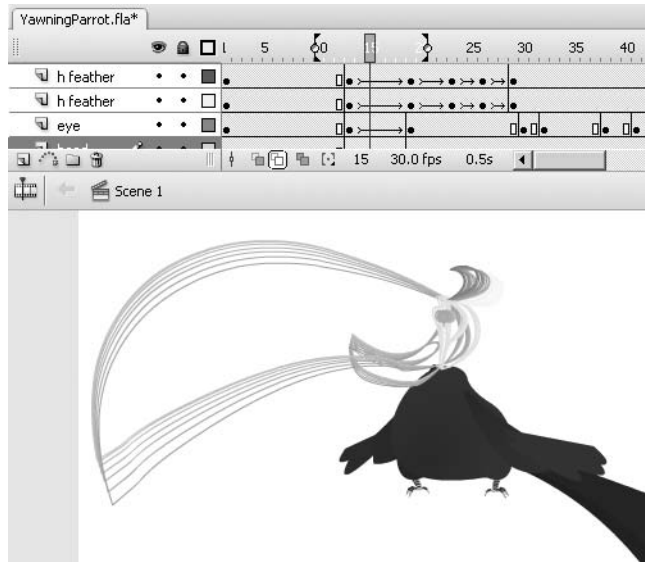
Figure 7-39. Various onion skin settings

7

- What are the other onion modifiers? **Onion All** spreads the onion markers along the whole timeline. Try it with this file—the result is overwhelming (and also makes it hard to drag the playhead around), but with timelines of little movement, it probably has its place. If you want some setting besides 2, 5, or All, drag the markers along the timeline yourself. If you like, you can look eight frames back and two frames forward—or any combination that suits your animation.

The top two choices work like this: **Always Show Markers** keeps the onion markers visible, even if you toggle the **Onion Skin** button off; and **Anchor Onion** keeps the onion markers from following the playhead.

- Choose **Onion 5** and drag the playhead to frame 15. Click the **Onion Skin Outlines** button. Note that the same sort of onion skinning occurs, but that the tweened areas are shown in wireframe format (see Figure 7-40). This makes it even clearer to see what’s moving and what isn’t.



**Figure 7-40.** Onion skin outlines show tweened artwork in a wireframe format.

*Remember, onion skinning is just as relevant to shape tweens as it is to motion tweens.*

## Editing multiple frames

Timeline animation can be painstaking work, no doubt about it. Even if you're using onion skinning, chances are good that you're focused on only a handful of frames at a time. There's nothing wrong with that—as long as you remember to keep your eye on the big picture, too. Sooner or later, it happens to everyone: artwork is replaced, your manager changes her mind, or you find that you've simply painted yourself into a corner and need to revise multiple keyframes—maybe hundreds—in as few moves as possible.

Fortunately, the timeline has a button called **Edit Multiple Frames**, which allows you to do just what it describes. That's the obvious answer, of course, and we'll cover that in just a moment, but it's worth noting that the concept of mass editing in Flash extends into other avenues.

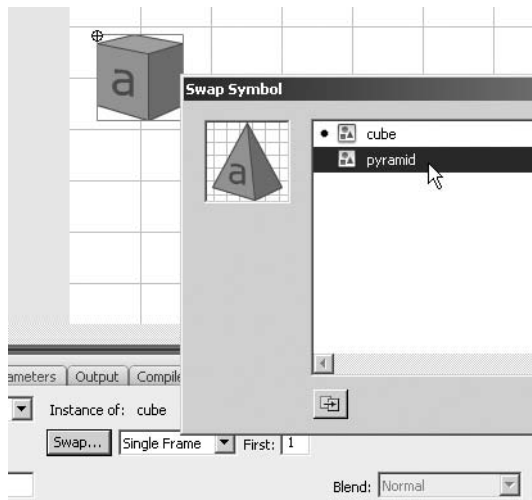
Due to the nature of symbols, for example, you can edit a library asset and benefit from an immediate change throughout the movie, even if individual instances of that symbol have been stretched, scaled, rotated, and manipulated in other ways. If an imported graphic file, such as a BMP, has been revised outside of Flash, right-click (PC) or Ctrl-click (Mac) the asset in the library and, from the context menu, select either **Update** (if the location of the external image hasn't changed) or **Properties**, and then click the **Import** button to reimport the image or import another one.

Sometimes it's not that easy. Sometimes you will have finished three days of meticulous keyframing only to learn that the symbol you've tweened isn't supposed to be *that* symbol



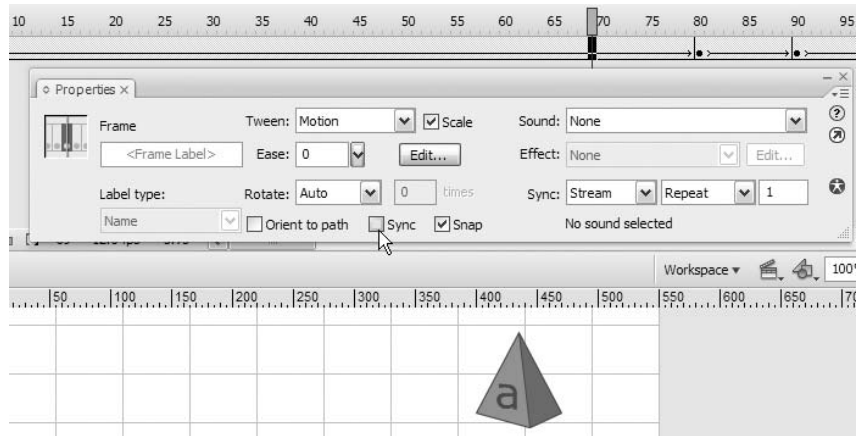
at all. Time to throw in the towel? Well, maybe time to roll the towel into a whip. But even here, there's hope . . . if you're using graphic symbols. It's easy enough to swap out symbols of any type for any other type at a given keyframe, but the swap only applies to the frames leading up to the next keyframe. With graphic symbols, it's possible to apply a swap across keyframes, but you have to know the secret handshake.

1. Open `SyncPropertyGraphic.fla` and note that a cube has been motion-tweened for you along a clockwise rectangular path. Use your imagination to picture the rectangular path as something more spectacular. Now, revel in that moment, because in this hypothetical world, you did that—and it's really cool. Here comes the drama: the boss eases into your cubical, apologetic at first, but steadily annoyed at having to elbow past your high-fiving buddies. Something is wrong, says the boss. Something is dreadfully wrong. The client wanted the pyramid, not the cube.
2. Select the cube at frame 1 and press the `Swap` button. (Remember, the boss is watching.) Select the pyramid symbol and press `OK`. Scrub the playhead a bit to confirm that the tween movement has picked up the new symbol. Smile as the boss leaves.
3. Now scrub to frame 80 and beyond. Look quickly over your shoulder. Good, the boss is still walking away. Why didn't the swap (shown in Figure 7-41) take? The answer rests on a tween property called `Sync`, which you can see in the Property inspector when you click anywhere in the span of frames that comprises a motion tween. The `Sync` property sets up a relationship between keyframes that locks their symbol in an unbreakable chain—well, unbreakable until you choose to remove the check mark from the `Sync` setting.



**Figure 7-41.** Swapping symbols can sometimes produce unexpected results.

4. Select the span of frames between each pair of keyframes, and click `Sync` to enable it (see Figure 7-42). As you do this, note that the small vertical line to the left of each keyframe disappears. This indicates the synchronized relationship. Note also that the pyramid swap occurs.



**Figure 7-42.** Tweens that are absent of Sync are “segregated” by a vertical line to the left of each keyframe in the timeline.

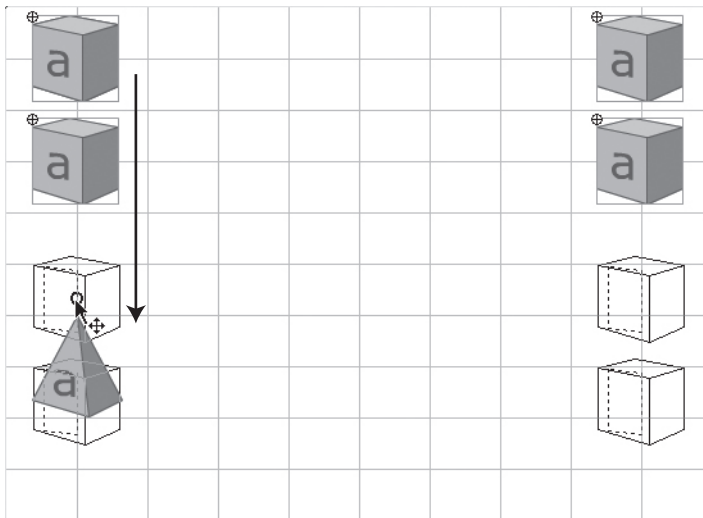
5. Now that all of the keyframes are synchronized, select any keyframe after frame 1 and use the Swap button to change the symbol back to the cube. You’ll find that you can’t. The Sync option prevents changes to any keyframe but the first in the chain.
6. Select the first keyframe and use the Swap button to change the symbol back to the cube. Scrub the timeline and verify that the swap has occurred across the board.

*There are actually two ways to apply a motion tween, and we’ve purposefully been steering you toward one of them so far in this chapter. Why? Because the other way has an interesting, but not at all obvious, side effect that is omitted by the Property inspector approach. As with shape tweens, you can right-click (PC) or Ctrl-click (Mac) between any two keyframes and select Create Motion Tween from the context menu. Applying the tween from this location automatically puts check marks in the Sync and Snap properties every time. This does not happen when a motion tween is applied via the Property inspector. With the Property inspector approach, Flash remembers whether Sync and Snap have already been chosen, and sets their check marks accordingly.*

*In addition to this, Create Motion Tween has the potential to create new library assets on your behalf, which you may not want. This happens when you use this approach to apply a motion tween to non-symbols, such as a shape, a primitive, or grouped elements. Try it and you’ll see: Flash will attempt to make the motion tween work even though you’ve applied it to the wrong sort of object. You’ll find two new symbols, Tween 1 and Tween 2, in the library—more, if you do it repeatedly—and Flash will apply motion tweens to those symbols instead.*

So much for updating content by swapping out symbols. You may be perfectly happy with the artwork as is—it may be the *placement* of content that’s out of whack. This is where the Edit Multiple Frames button makes its entrance. Using this button requires a bit of prep work, so let’s step through that:

1. First, you've got to decide on a range of possibly editable frames. This range extends both horizontally and vertically. Do you want to edit one layer only, multiple layers, or all layers? The easiest way to keep from editing the wrong layer is to temporarily lock it. Open `EditMultipleFrames.fla` and click the Lock icon in the Pyramid layer. This makes the Cube layer the exclusive focus of your attention. Next, make your horizontal decision. The extent of your onion skin markers determines the lateral range. Use the Modify Onion Skinners drop-down menu to select Onion All.
2. Click the Edit Multiple Frames button. At this point, you've chosen a valid range of editable frames and have activated the possibility to select them.
3. Now . . . to do it. Select `Edit > Select All` and drag the upper-left cube down so that it rests on the pyramid's peak, as shown in Figure 7-43. Thanks to the Edit Multiple Frames button, all four keyframes of this animation are moved at the same time in relation to each other.



**Figure 7-43.** The Edit Multiple Frames button lets you adjust many keyframes at the same time.

Test your movie to confirm that the cube now rests on the pyramid for the full duration of the animation. By following this procedure, you can edit not only the position, but also the scale, rotation, and any other property available to the element at hand, whether shape or symbol.

*In complex movies, you may find it tedious to temporarily lock a great number of layers. Instead of using `Select All`, you can simply select the desired layer by single-clicking its name. Hold `Shift` while clicking to select multiple adjacent layers and `Ctrl/Cmd` while clicking to select multiple non-adjacent layers.*

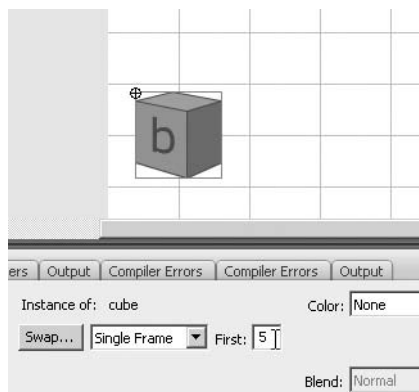
## Combining timelines

Pat your head. Good! Now rub your tummy. Excellent. Now . . . do those both at the same time. Until the undertaking snaps into place, it might seem an impossible feat, but once you manage to pull it off, you know you've done something pretty snazzy. Flash animations get interesting in the same way when you combine techniques and timelines. This is where the distinction between graphic symbols and movieclip symbols really comes into play. Both types of symbols have timelines, but each behaves in a different way. Understanding this paves the way toward good decision-making in your animations.

Movieclips operate independently of the timelines they sit in. You can create a 500-frame animation on the main timeline, and then transfer all those frames into a movieclip symbol, and everything will run the same—even if that movieclip only occupies a single frame on the main timeline. Not so with graphic symbols. Graphic symbols are synchronized with the timelines that contain them, so if you transfer all those frames into a graphic symbol, that symbol will have to span out a length of 500 frames in the main timeline in order for its own timeline to fully play.

While movieclips can be instructed with ActionScript to stop, play, and jump to various frames, graphics can only be told to hold their current position, play through once, or loop. This instruction comes not from ActionScript, but by Property inspector settings. ActionScript within the timelines of graphic symbols is not performed by a containing timeline. Sound in graphic symbols is also ignored by parent timelines.

1. Open `TimelineCombine.fla` and select the symbol at frame 1. Look in the Property inspector and you'll see that the Options for graphics drop-down menu, next to the Swap button, is set to Single Frame, and that the single frame shown is frame 1. The frame in question belongs to the timeline of this graphic symbol. Change this number to 5 and press Enter/Return. Depending how you left things in an earlier exercise, you'll either see a cube or a pyramid—but in both cases, you'll see the graphic's text content, a lowercase a, become a lowercase b, as shown in Figure 7-44.



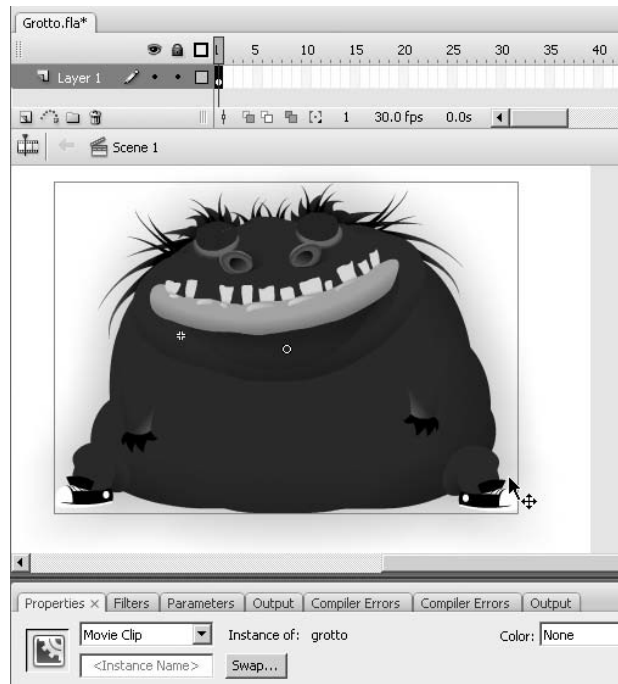
**Figure 7-44.** Changing the displayed frame of a graphic symbol

2. Double-click the cube or pyramid asset in the library and you'll see why. Both symbols have a timeline, and the text layer in each changes every five frames.

3. Select the symbol again in the main timeline. Change the Single Frame setting to Play Once, and change the First input field to 10. This updates the displayed letter to c and instructs the graphic symbol to play through the end of its timeline once. Drag the playhead slowly to the right to see the letters d, e, and so on, displayed through j while the symbol moves across the stage. At j, the symbol continues to move, but no longer updates its text. The reason for this is that the symbol's timeline has reached its end, but does not repeat.
4. Change the Play Once setting to Loop, and change First to 1. Scrub again and you'll see the letters start from a and repeat again from a after j is reached.

Designer and animator Chris Georgenes ([www.mudbubble.com](http://www.mudbubble.com)) has lent his talents to numerous cartoons on television and the Web, including *Dr. Katz, Professional Therapist*, Adult Swim's *Home Movies*, and, well, more online animation than either of us could shake a stick at. One of the giants in the field, Chris uses combined timelines to great effect in practically all of this Flash work. From walk cycles to lip-synching, Chris builds up elaborate animated sequences by organizing relatively simple movement into symbols nested within symbols. The orchestrated result often leaves viewers thinking, "Wow, how did he do that?!" Luckily for us, Chris was kind enough to share one of his character sketches, which provides a simplified example.

1. Open the *Grotto.fla* file from the examples folder for this chapter. Note that the main timeline only has one frame and only one symbol in that frame (see Figure 7-45). This base symbol is a movieclip, because Chris wanted a slight drop shadow effect on the friendly monster, and graphic symbols don't support filters.



**Figure 7-45.** Nested symbols allow you to take the most useful features of each symbol type.

2. Double-click this movieclip to enter its timeline.

Even with a basic example like this one, you may be surprised by the number of layers inside. Try not to feel overwhelmed! The layers, as shown in Figure 7-46, are neatly labeled. (Now that you see how a pro does it, start labeling your layers as well.) Also, although there are many of them, they all have a purpose. If you like, hide a number of layers by clicking in the eye column of each to see how each adds to the complete picture. What we're interested in is the mouth.

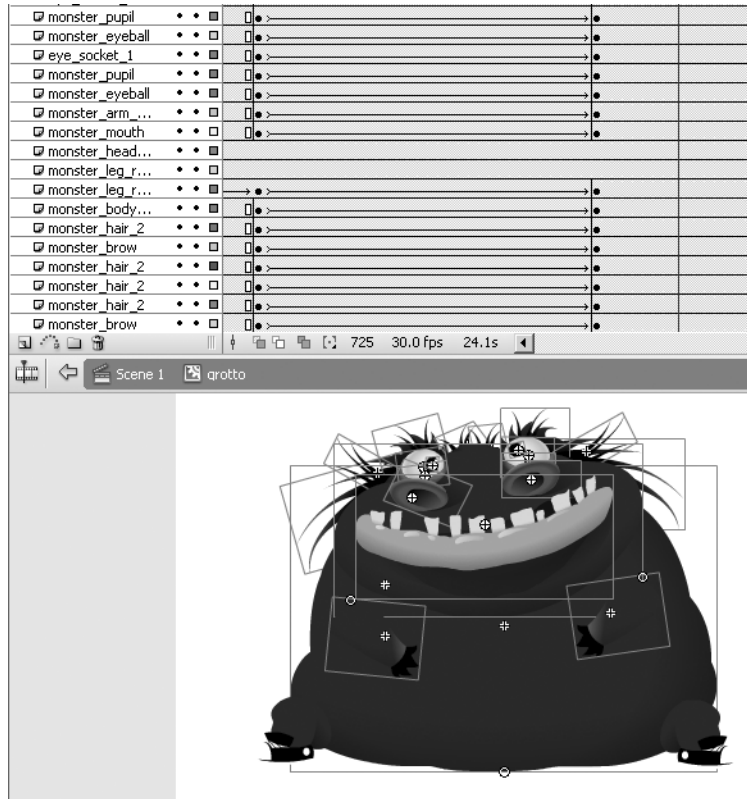
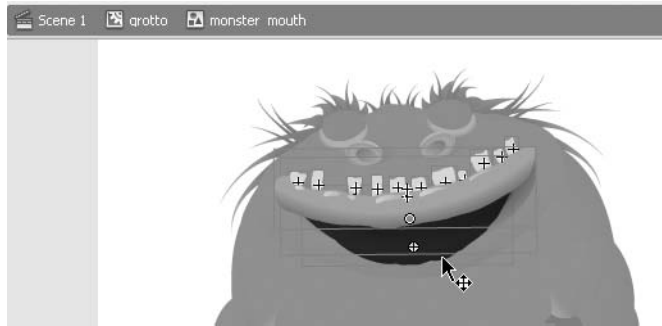


Figure 7-46. Complex images and animations are built up from simple pieces.

3. Double-click the mouth symbol to enter its timeline. Here too there is a handful of layers, comprising the lips, teeth, and a few shadows of this monster. There are 115 frames of animation here—mostly motion tweens, but also a shape tween at the bottom—and if you scrub the timeline, you'll see the mouth gently move up and down . . . this is Grotto breathing (see Figure 7-47).



**Figure 7-47.** Nesting timelines is a way to compartmentalize complexity.

Because the mouth symbol itself is a graphic symbol, its movement can be made to scrub along with the timeline of its parent.

4. Return to the grotto timeline by clicking the grotto movieclip icon in the breadcrumbs area at the bottom of the Timeline panel.

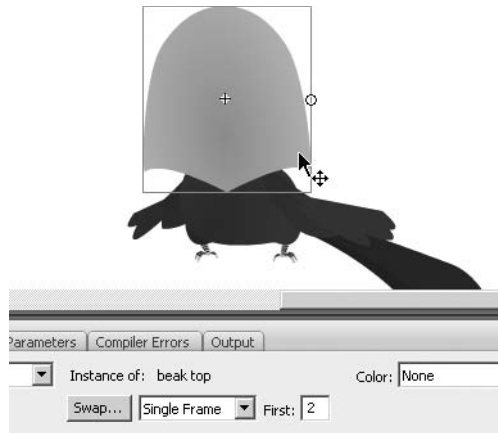
Drag the playhead to a keyframe, such as 11, and click the mouth symbol. Note that it's set to Loop in the Property inspector and starts at frame 11. Because the mouth symbol loops, the mouth itself can be tweened to various locations and rotations during the course of the grotto symbol's timeline. The complexity of the mouth's inner movement is neatly tucked away into the mouth symbol.

At any point, you can pause this breathing movement by adding a keyframe in the grotto symbol's timeline and changing the mouth symbol's behavior setting from Loop to Single Frame.

The phenomenon you've just seen can be nested as deeply as you like. Even limited nesting, like that in *Grotto.fla*, can, for example, be used to animate a bicycle—the wheels rotate in their own timeline while traveling along the parent timeline—or twinkling stars. Just keep in mind, if a given graphic symbol's timeline is, say, 100 frames long, and you want *all* of those frames to show, the symbol will have to span that many frames in the timeline that contains it. Of course, you may purposefully want to show only a few frames. Let's look at that parrot again for an example:

1. Open *YawningParrot.fla* and drag the playhead slowly back and forth between frames 60 and 65. As the head turns, the beak moves from left to right. A bit of motion tweening squashes the beak as it nears the crossover, and the shape changes completely in the middle at frame 62.
2. Select the upper beak at frame 61. Open the Transform panel (Window ► Transform) and note that the width of this symbol has been reduced to half. In the Property inspector, note that this symbol is an instance of the beak top asset in the library. It is set to Single Frame at frame 1.

3. Select the upper beak at frame 62. This symbol is still the beak top asset and is still set to Single Frame, but this time its First property is set to 2 (see Figure 7-48). All it takes is one quick frame to complete the illusion of a head turn!



**Figure 7-48.** Graphic symbols can be used as mini-libraries to keep the real library from overcrowding.

This is a perfect example of how a graphic symbol's timeline can be used to reduce clutter in the library. It's not hard to imagine how handy this would be for swapping out mouth shapes in the case of an animated character that speaks. Sure, you can use the Swap button to replace any symbol with another at any keyframe, but it is much less hassle to update the First field in the Property inspector for graphic symbols. This technique is one of those hidden gems that becomes a favorite once you realize it, and we thank Chris for sharing such a useful trick.

*For more information on character design, advanced tweening, and lip-synching techniques, search "Chris Georgenes" on the Adobe website ([www.adobe.com/](http://www.adobe.com/)) to see a number of Chris's articles and Macrochats (Flash-based recordings of live tutorial presentations).*

## Motion tween effects

A common question on the Adobe support forums is how to fade in an imported photo, and then fade it out again. People are comfortable enough importing a BMP or PNG, but when they drag it to the stage, there doesn't seem to be a way to adjust its transparency at all, much less over time. The trick here is to convert the photo into a symbol. The type of symbol depends on what effects you want to apply. Both graphics and movieclips support color effects such as Brightness, Tint, Alpha, and Advanced, but only movieclips support filters. Let's try it:

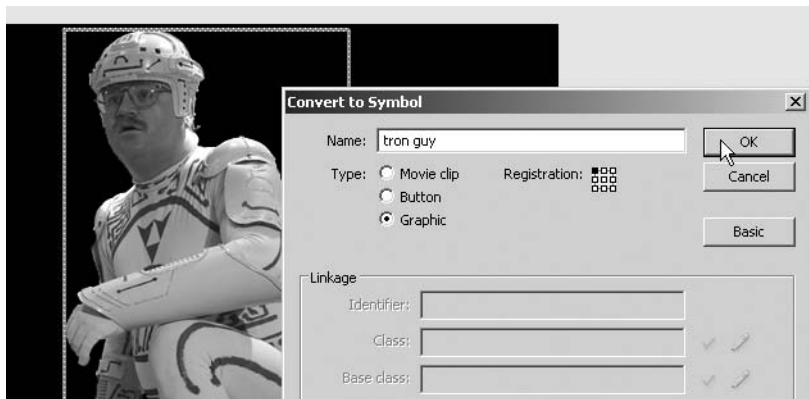


1. Create a new Flash document and save it as `TronGuy.fla`. Using the Property inspector, set the document's dimensions to  $550 \times 400$  and its background color to black.
2. Select `File > Import to Stage` to import the `tronguy.png` graphic file from the exercises folder for this chapter. Use the `Align` panel (`Window > Align`) to center the image.

*Who is this debonair futuristic fellow? Ladies and gentleman, we present to you Jay Maynard, better known on the Internet as Tron Guy ([www.tronguy.net/](http://www.tronguy.net/)). Jay has made numerous appearances on "Jimmy Kimmel Live" in his homemade costume inspired by the 1982 Disney film *Tron* and was good enough to let us use his likeness for this book.*

There is doubtless no better way to demonstrate a tweened Glow filter than to apply it to Tron Guy—but first, let's tween an alpha transition.

3. Select the imported PNG and note the absence of color styling properties. With the PNG selected, go to `Modify > Convert to Symbol` and choose `Graphic`, as shown in Figure 7-49. Name the symbol `tron guy` and click `OK`. Select the symbol and note the `Color` drop-down menu.



**Figure 7-49.** Converting an imported image to a symbol allows for color and alpha tweens.

4. Insert a keyframe at frame 10. Select frame 1 and choose `Alpha` from the `Color` drop-down menu. A slider will appear. Drag this down to zero, and then apply a motion tween between the two keyframes. Suddenly Tron Guy's entrance is visually more interesting.
5. To make it even more dramatic, choose the `Advanced` option, which makes a `Settings` button appear. Select the symbol at frame 10 and click the `Settings` button. In the `Advanced Effect` dialog box, drag the right-hand red and green sliders down to  $-225$ , and then click `OK`. Select the symbol at frame 1 and click the `Settings` button again. Drag the left-hand `Alpha` slider up to  $100\%$ . Drag the right-hand red, green, and blue sliders down to  $-225$ . Click `OK` and scrub the timeline to see the results.

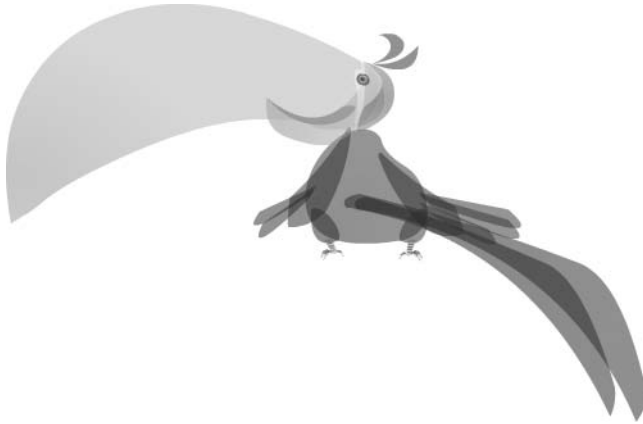
6. For the final touch, let's add some glow to Tron Guy's costume. Open the `TronGuyGlow.fla` file for this one, because we've outlined some of his circuits for you. Insert a keyframe in the costume layer at frame 20. Select the costume symbol at this frame and flip the Property inspector to the Filters tab. Add a Glow filter with the following settings:
  - Color: #0099FF
  - Blur X: 8
  - Blur Y: 8
  - Strength: 330%
7. Insert a keyframe in the circuits layer at frame 20 and add a Glow filter to the symbol on that layer. Use the same settings, except make the color #FFFFFF (white). Apply a motion tween between the keyframes in both layers. A single line of ActionScript in the scripts layer—`gotoAndPlay(10)`—loops the movie between frames 10 and 20. Test the movie to see your handiwork (see Figure 7-50).



**Figure 7-50.** Say, that looks just like the movie!

*The reason the costume layer's glow follows the contours of the costume is because this image is a PNG with a transparent background. If the photo had a solid background, the glow would outline a rectangle around the photo itself.*

If you motion tween the alpha property of nested vector art, you may be in for a surprise. Semitransparent graphic and movieclip symbols that are made up of other symbols don't fade out cleanly as a whole. Instead, each piece fades individually, as shown in Figure 7-51.



**Figure 7-51.** Unintentional X-ray effect caused by alpha reduction to nested symbol

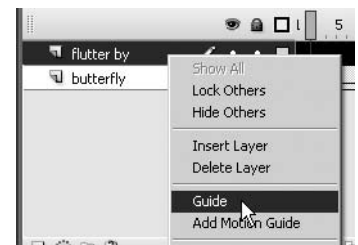
There are two ways to avoid this phenomenon. On solid backgrounds, replace the alpha tween with a tint tween set to the same color as the background. In the case of movieclips, you may alternatively leave the alpha tween as is, but set the blend mode to Layer. These solutions are demonstrated in the `FadingParrot.fla` file.

## Motion guides

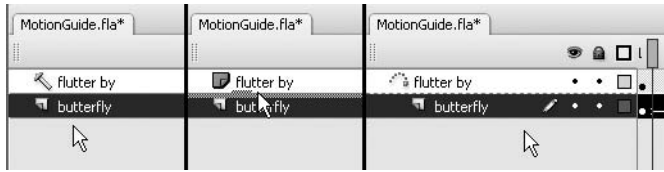
Tweening in a straight line is effortless, and we've shown how easing can make such movement more realistic. But what if you want to tween along a curve? Wouldn't it be great if we could tell you that's only marginally more difficult? Well, we can, and we'll even show you. The trick is to use something called a motion guide.

1. Open the `MotionGuide.fla` file that accompanies this chapter. You'll see a butterfly graphic symbol in one layer and a curvy squiggle in another. If you scrub the timeline at this point, you'll see the butterfly tween in a straight line with a slight rotation between frames 240 and 275. Butterflies don't really fly like that, so let's fix the flight pattern.
2. Right-click (PC) or Ctrl-click (Mac) the `flutter by` layer and choose `Guide` from the context menu, as shown in Figure 7-52. Its icon turns from a folded page to a hammer.

This changes that layer into a guide layer, which means anything you put into it can be used as a visual reference to help position objects in other layers. Depending on your snap settings (`View > Snapping`), you can even snap objects to drawings in a guide layer. Artwork in guide layers is not included in the published SWF and does not add to the SWF's file size. In this exercise, the squiggle is your guide—but setting its layer as a guide layer isn't enough. It must be a motion guide, as shown in Figure 7-53. To make this happen, gently drag the `butterfly` layer up and to the right. The hammer icon will change back to the folded paper icon, and when you let go, it will change again into what looks like a shooting comet.



**Figure 7-52.** Changing a normal layer into a guide layer

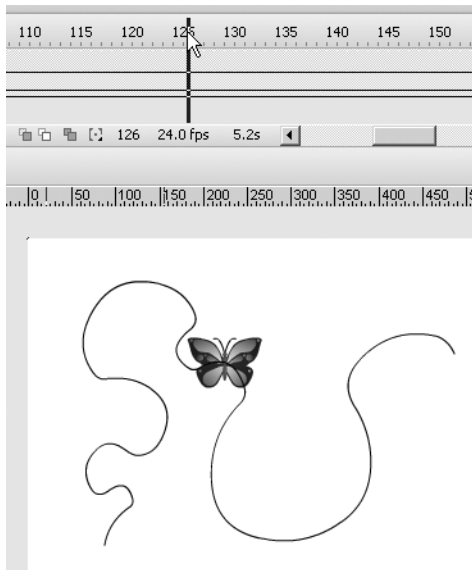


**Figure 7-53.** Changing a guide layer into a motion guide layer

The other way to create a motion guide layer is make it from scratch by selecting the layer you want to guide, and then pressing the Add Motion Guide button on lower left of the timeline.

*Motion guides must have a clear beginning and end point, as does the squiggle shown. Guides that cross over each other may cause unexpected results, so take care not to confuse Flash. Also, make sure your motion guide line extends the full length between two keyframes.*

3. Thanks to the Snap setting in the tweened frames (see the Property inspector while clicking anywhere inside the tween), the butterfly should already be snapped to the closer end point at the last keyframe. Scrub to make sure. The butterfly should follow the squiggle along its tween (as shown in Figure 7-54). If it doesn't, make sure to snap the butterfly to the squiggle's left end in frame 1 and right end in frame 240. Imagine tweening that by hand!



**Figure 7-54.** A motion guide affects the tweened path of a symbol.

4. Click anywhere inside the tween and put a check mark in the Orient to Path check box in the Property inspector. Scrub the timeline to see how this affects the butterfly's movement. The butterfly now points in the direction described by the squiggle.
5. To add even more realism, let's add some complexity, as described earlier in the "Combining timelines" section. Double-click the butterfly asset in the library to enter the Symbol Editor. Add a keyframe to the upper wings and lower wings layers in frames 5 and 10. In the body layer, click in frame 10 and extend the frames to that point (Insert > Timeline > Frame). Select both wings symbols at frame 5, and use the Free Transform tool to reduce their width by about two-thirds. Use the Alt/Option key to keep the transformation centered.
6. Motion tween the wings layers as shown in Figure 7-55, and test your movie to see the combined effect.

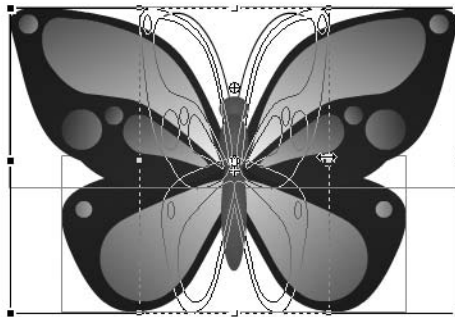
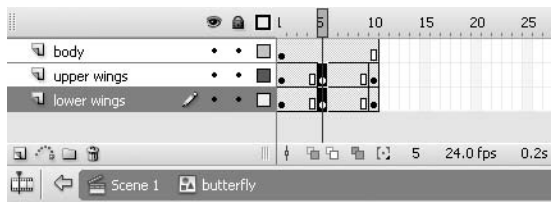


Figure 7-55. Tweening a timeline inside the butterfly graphic symbol

## Tweening a mask

In Chapter 3, you used text to create a mask. In this chapter, you'll use a shape, and you'll apply a shape tween to it to produce an iris wipe transition, like in the old movies. Animating masks is no more difficult than animating normal shapes or symbols. In fact, the only difference is the status of the layer that contains the mask itself.

1. Open the TweenMask.fla file that accompanies this chapter. You'll see three layers: a photo of one of the authors as a young boy, a text layer to provide some background texture, and a small yellow dot. Insert a keyframe at frame 30 in the dot layer. Use the Transform panel (Window ► Transform) to increase the size of the dot in frame 30 to 800%. This makes the dot much easier to manipulate.
2. Use the Free Transform tool to increase the size of the dot yet further, so that it matches the width and height of the photo. Because the dot is a shape, apply a shape tween between the keyframes in the dot layer. Scrub the timeline to see the result (shown in Figure 7-56). Easy as pie!



**Figure 7-56.** Masks can be tweened just as easily as regular shapes or symbols.

Often, once new designers get comfortable with motion guides and masks, they come to the realization that a layer can either be converted to a guide or mask layer, but not both. Naturally, the question arises, “Is it possible to tween a mask along a motion guide?” The answer is yes, and yet again, combined timelines come to the rescue.

1. Open the TweenMaskMotionGuide.fla file. The setup is very similar to the TweenMask.fla file, except that the dot layer is now named guide mask. Double-click the guide mask symbol to enter its timeline.
2. Confirm that a dot symbol is motion tweened in association with a motion guide. Return to the main timeline.
3. Right-click (PC) or Cmd-click (Mac) the guide mask layer and select Mask from the context menu. This nested combination gives you a motion-guided mask!

## Your turn: Making an animated button

By now, you should get the idea that combined timelines are useful things. Here’s a quick look at a very popular effect for the over state of a button symbol. Even a little bit of motion can add just the right touch to liven up an otherwise simple button.

1. Open the `AnimatedButton.fla` file that accompanies this chapter. Test the movie to see how the buttons currently work. It's certainly not bad looking, but plain vanilla nonetheless. We're going to add some animated glint to the `Over` frame.
2. Double-click the `glint` asset in the library to enter its timeline. There are three things to notice here:
  - A `scripts` layer tells the timeline to only play once (`stop()` in frame 5).
  - A `mask` layer constrains the animation to the shape of the button only.
  - A `shape-tweened` layer, named `glint`, moves a rounded rectangle from above to below the mask.
3. Double-click the button symbol to enter its timeline. Add a new layer above the `bg` (background) layer. Name the new layer `glint`. Insert a keyframe in the `glint` layer at the `Over` frame.
4. Drag the `glint` movieclip to the stage in the `Over` keyframe. Use the Property inspector to position the `glint` symbol at `x: 0` and `y: -30`. Insert a blank keyframe (Insert ► Timeline ► Blank Keyframe) in the `Down` frame of the `glint` layer. This keeps the animation from occurring while the mouse clicks the button; it will only show when the mouse hovers over the button and when the mouse releases from a clicked state, both of which lead to an over state.

## An even cooler animated button

This technique goes right back to the roots of Flash and the first efforts aimed at getting video to play in Flash. You will be dealing with it in greater depth in the next chapter, but here is a rather interesting technique that doesn't put objects in motion, but instead treats motion as a sort of flip book. Here's how:

1. Open a new Flash document, change the stage dimensions to 94 pixels wide by 44 pixels high, and set the frame rate to 24 fps. Name the Flash file `Circuit` and save it to the `Circuit` folder in your `Exercise` folder.

Inside the `Circuit` folder are a QuickTime movie named `Circuits` and a folder named `Images`, which contains 50 sequentially numbered JPG images. These images were created by opening the QuickTime movie in QuickTime Pro—you can do this with any video editor that has QuickTime output capability—and exporting the movie as an image sequence (as shown in Figure 7-57). This technique, called **rotoscoping**, breaks a video into a series of images (which in this case, we then saved to the `Images` folder).

2. Create a movieclip named `Circuit`, and when the Symbol Editor opens, select File ► Import ► Import to Stage.

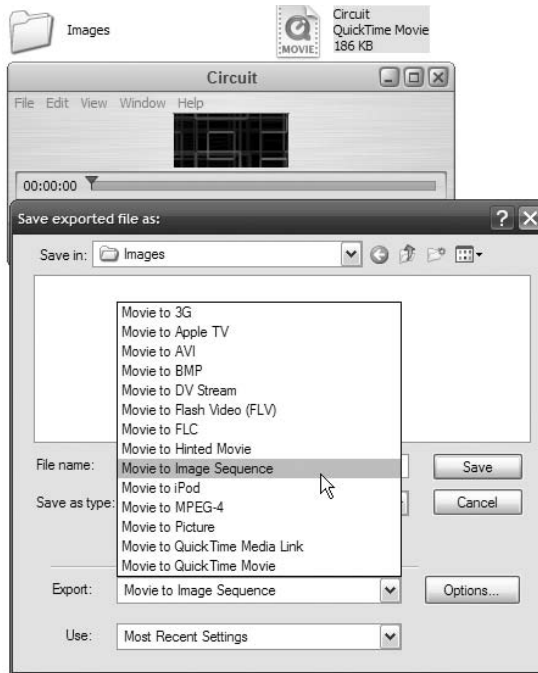


Figure 7-57. We start with a rotoscoped video.

3. When the Import dialog box opens, navigate to the Images folder and select the first image in the sequence (Image01), and click Open. Flash will grab the image, notice that there is a number after it, and think, “Hmmm, this seems to be part of a sequence.” This is why Flash asks you, as shown in Figure 7-58, if you want to import the entire sequence. Click Yes. You will see a progress bar appear; when it is finished, each image will appear in the timeline. The neat thing about this is that all the images are in exactly the same position in each frame, and they are also placed in the library.

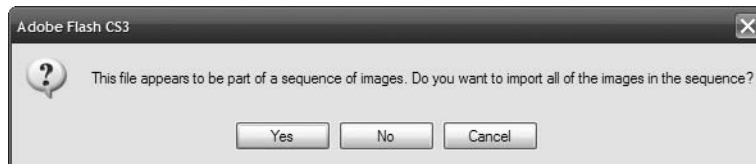
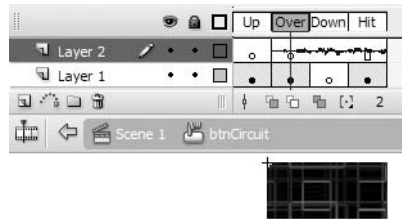


Figure 7-58. Flash, seeing a sequence of images, asks if it can import the entire sequence.

4. Import the Zap.mp3 file into the library.
5. Create a new button symbol named btnCircuit. Drag the Image01 file from the library to the stage and, using the Property inspector, set its x and y coordinates to (0, 0).
6. Add a keyframe to the Over frame of the button symbol, and drag the Circuit movieclip to the stage. Set its x and y position to (0, 0) using the Property inspector.
7. Insert a blank keyframe in the Down frame.



8. Insert a keyframe in the Hit frame, draw a box that is 94 pixels wide by 44 pixels high, and position it at (0, 0). The content in the Hit frame won't be visible. Hit frames are used by Flash to determine the hotspot for a button.
9. Add a new layer named Audio to the button timeline, and insert a keyframe in the Over frame of the Audio layer. Drag the Zap file from the library to the stage. Click the sound in the Over frame and set its property to Event. When the button is rolled over, sound in the Over frame will play, and the sequence of images in the movieclip will also start to play (see Figure 7-59).
10. Click the Scene 1 link to return to the main timeline and test the file.



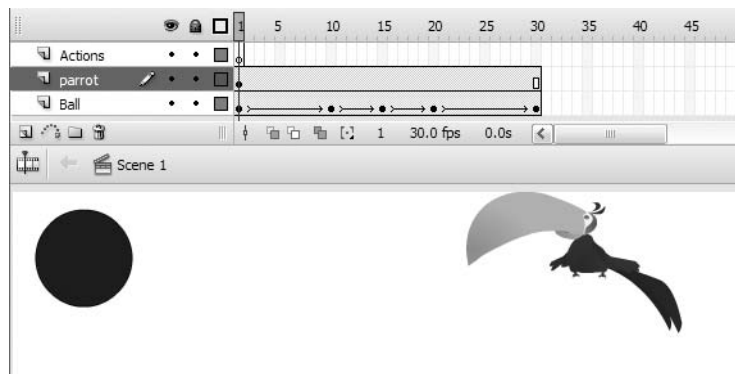
**Figure 7-59.** Couple audio with rotoscoping to add a bit of zing to an animated button.

## Copy motion as ActionScript 3.0

You may have noticed a distinct lack of ActionScript in this chapter. The reason is that the subject of programmatic motion simply can't be covered with any degree of thoroughness in one chapter. If you are really interested in the subject, then *Foundation ActionScript 3.0 Animation: Making Things Move!* by Keith Peters or *Foundation ActionScript 3.0 with Flash CS3 and Flex 2*, by Steve Webster and Sean McSharry (the companion volume to this book) are two excellent starting points. Still, we'd like to mention a really neat addition to Flash Professional CS3 that fits this chapter like a glove.

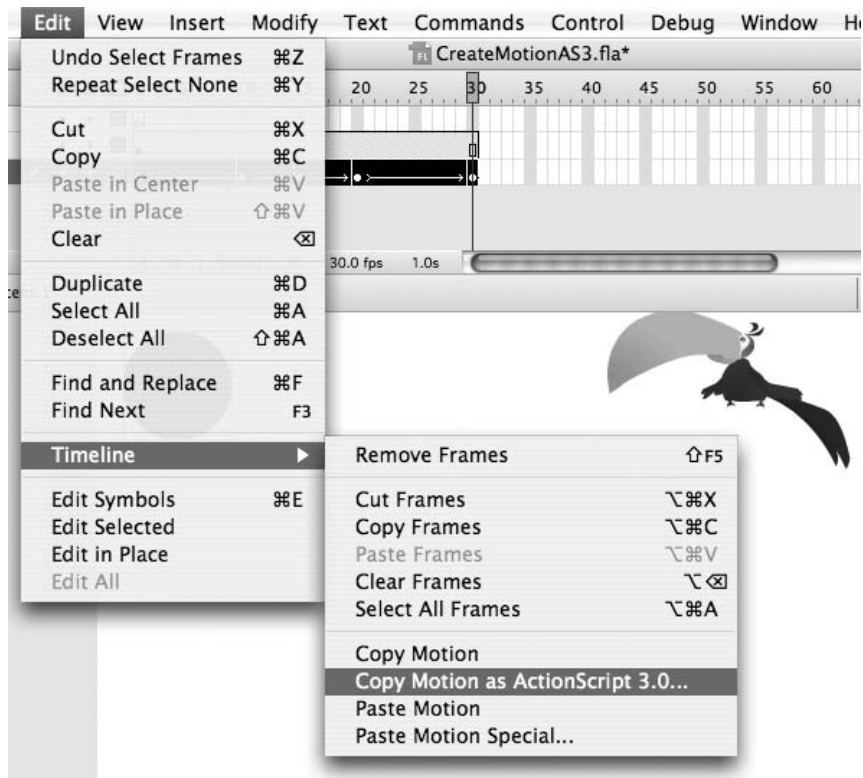
The feature is copy motion as ActionScript 3.0. Here's how it works:

1. Open the CreateMotionAS3.fla file. When the file opens, you will see we have added an animated ball and a parrot to the stage, as well as an Actions layer (see Figure 7-60).



**Figure 7-60.** We start with a ball and one slightly worried parrot on the stage.

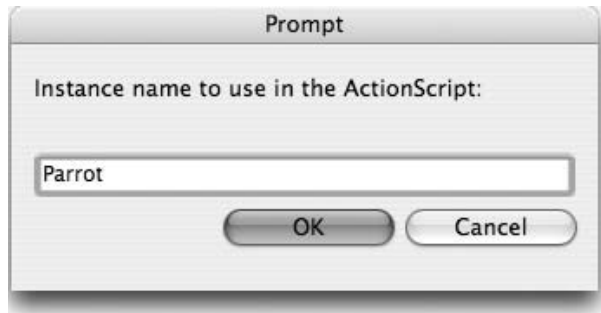
2. Scrub the playback head across the timeline. You will see the ball fall to the bottom of the stage, squash, stretch, and bounce back up to the top of the stage. Let's apply that animation to the slightly worried parrot.
3. Select the parrot on the stage and, in the Property inspector, give it the instance name of Parrot.
4. Select the first frame of the Ball layer, press the Shift key, and select the last frame of the layer. This selects all of the frames.
5. With the frames selected, either select **Edit** > **Timeline** > **Copy Motion as ActionScript 3.0**, as shown in Figure 7-61, or right-click (PC) or Ctrl-click (Mac) and select **Copy Motion as ActionScript 3.0** from the context menu.



**Figure 7-61.** You can access the command through the Edit menu item or the context menu.

6. When you select that menu item, a dialog box will open asking you for the name of the symbol to which the motion will be applied (see Figure 7-62). Enter Parrot and click OK.

What you have done is ask Flash to translate the motion of the ball into ActionScript and apply that same motion to the parrot. This all happens in the background, and when the motion is translated into ActionScript, the code is placed on the clipboard.



**Figure 7-62.** You must identify the instance to which the ActionScript will be applied.

7. Select the first frame of the Actions layer and open the ActionScript Editor. Click in the Script pane and select Edit ► Paste. The code will be pasted into the Script pane.
8. Close the ActionScript Editor to return to the main timeline. Save and test the movie. The parrot takes on the animation and distortion of the ball in the SWF (see Figure 7-63).



**Figure 7-63.** Being squashed sort of explains why the parrot looks worried.

Now that you know how this works, there are obviously some rules. The first one is that the motion must be a motion tween using a symbol, and the second is the code can only be applied to a movieclip on the stage. The great thing about this new feature is that the motion tween can contain the following properties (many of which we've talked about in this chapter):

- Position
- Scale
- Skew
- Rotation
- Transformation points
- Color
- Blend modes
- Orientation to path
- Cache as bitmap
- Frame labels
- Motion guides
- Custom easing
- Filters

The bottom line is that you can create some pretty amazing animation effects without writing a single line of ActionScript.

**Noggin nuggets of gold from a visionary rascal**

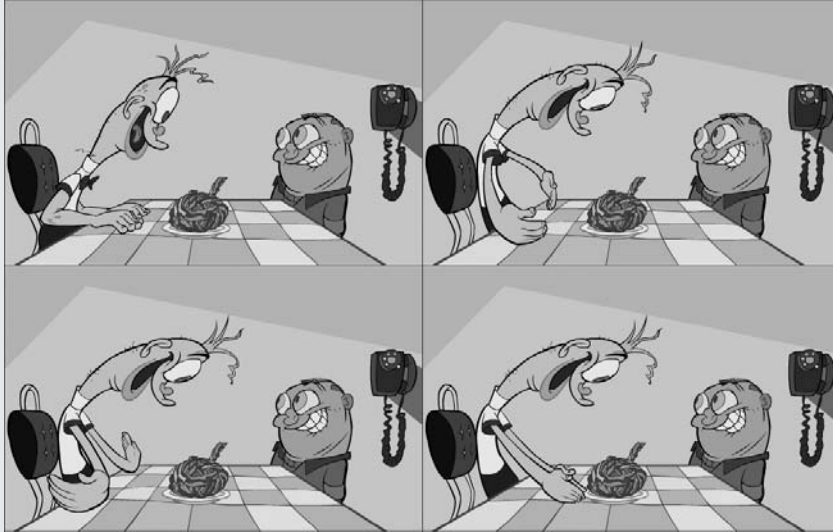
*Back in high school, one of the authors fancied himself a poet. As often happens in those formative years, the subject was introduced in terms of rhyme schemes. To be sure, there's nothing essentially wrong with that. The usual Romantic role models—Byron, Wordsworth, Keats, Longfellow, Emerson—wallowed in rhyme. It's a long-standing custom in many artistic disciplines to “study the masters” first, and for good reason. The masters figured out where all the pebbles were, which toughened their feet. Walk in their shoes, and you benefit in the same way.*

*Of course, once traditions are in place, the path is cleared for visionaries: inventive weirdos who see things differently, who dash off into the brush and break the rules. People who find new pebbles. Think e.e. cummings. What we've shown you in this chapter are a number of well-worn trails. Shape tweening and shape hints, motion tweening and easing . . . these are familiar corridors for many a Flash master. We encourage you to tramp along these paths until your shoes are good and comfortable (and then be at the ready to kick off your shoes and sprint with the visionaries).*

*If you can keep up with him, you'll want to chase the flapping longfellows of John Kricfalusi (<http://johnkstuff.blogspot.com/>), creator of the “The Ren & Stimpy Show” and pioneer of the Flash-animated cartoon series. A full decade ago, John broke new ground with the “The Goddamn George Liquor Program,” which had cartoon fans laughing until . . . well, until milk spurted from their noses. For Flash cartooning, that was an Internet first. What's John's rhyme scheme? Enjoy Flash for the useful tool it is, but pile up most of your eggs in that basket called your brain.*

*“David asked me to write up some tips about how to creatively use Flash. I guess my best advice is to lean on it as little as possible, to not use it as a creative crutch. Flash isn't inherently a creative tool. It's not like a pencil or a brush or talent.*

*I use it mainly as an exposure sheet to quickly test my drawings and animation to see if they work. Your best Flash tool is your drawing skill. You will always creatively be limited by your ability to make interesting drawings and movement. I see many animators using Flash mainly for its in-betweens, or “tweens” as they are now called. This little tool makes every movement look smooth. But if you want to compete against the best animators, whether in Flash or in traditional animation, you will be competing with drawings, acting, and real motion (see the following illustration). Real motion has non-mathematical in-betweening. Every in-between looks different and conveys information that mere tweening can't. Tweening just moves the same drawing from one place to another, and it's completely obvious when you watch most Flash cartoons that you are watching tricks, not animation.*



No amount of tweening can accomplish such joyous hand clapping: those are frame-by-frame drawings.

*Since I started using Flash back in caveman times, I've been trying to find ways to make it not look like Flash, to try to undermine all its computery tricks. I've tried different approaches. It's hard for me to draw my key poses directly on the computer, so I usually draw them in pencil and scan them in. Once they are in, I time them in the timeline to musical beats. When I'm satisfied with the rough timing, I then draw breakdown poses directly on a Cintiq ([www.wacom.com/cintiq/](http://www.wacom.com/cintiq/)) in the timeline. I constantly roll across the animation to see if the motion is smooth. If I'm animating to a dialogue track, I draw the mouth positions in Flash and, again, roll back and forth to see if the animation is working.*

*I am always trying new ways to beat Flash's limitations and don't have a perfect solution. The best thing about Flash, to me, is that you can instantly see if your animation works, because you can play it back right after you do it. But Flash isn't doing the creative part. The drawings are. My best advice for how to be good at Flash is to learn as much about drawing and traditional animation as you can. That'll put you ahead of every Flash animator who just drags around some simple primitive pictures. More and more real animators are starting to use Flash, so the competition is going to get tougher for those who are lacking in drawing skills.”*

## What you've learned

- The difference between a shape tween and a motion tween
- Various methods of using easing to add reality to your animations
- How to use the timeline and the Property inspector to manage animations
- The creation and use of motion guides in animation
- How to translate an animation into ActionScript

This has been a busy chapter. The path led from tweening shapes to turning animations into ActionScript that can be used to animate movieclips on the stage. In many respects, this is an important chapter, because whether you care to admit it or not, Flash is quite widely regarded as an animation program first—all that other cool stuff it does is secondary. Many of the techniques and principles presented in this chapter are the fundamentals of animation in Flash. If there is one message you should get from this chapter, it is pay attention to how things move.

It is the attention to detail that separates the pros like Chris Georgenes (and now you) from the rest of the crowd. Whether it is a ball landing on the floor, a parrot turning its head, or a mallet striking a nail, a passion for detail will be the difference between a great animation and one that is so-so.

Now that you know how to move stuff around the stage, let's look at one of the rising stars of Flash: Flash video. Things have really changed in Flash CS3, and to find out how, all you have to do is to turn the page.







## 8 VIDEO IN FLASH



When Macromedia, now Adobe, launched Flash 8 Professional and included the Flash Video (FLV) Encoder and the FLVPlayback component with the application, a valid argument could be made that this marked the final acceptance of Flash as a viable web video medium. As more and more sites started featuring Flash video, there was a corresponding decline in sites that used the web video solutions provided by QuickTime, Windows Media, and Real Player.

The reason has more to do with cunning than market acceptance. Flash Player by that point in time could be found on well over 90% of all computers on the planet. The thing is, most people didn't see Flash as a media player. They thought of it as being this "cute thing" that played animations. When they suddenly realized they could stream audio (Chapter 5) and video through Flash Player without excessive wait times or downloading a plug-in, it was basically "game-set-match" for the others.

What we'll cover in this chapter:

- Streaming video
- Encoding an FLV
- Using the FLVPlayback component and a video object to play video
- Using the FLVPlayback control components
- Playing full-screen video
- Adding captions to Flash video
- Adding filters and blend effects to video

Files used in this chapter:

- DisgruntledDan.mov (Chapter08/ExerciseFiles\_CH08/Exercise/DisgruntledDan.mov)
- DisgruntledDan.flv (Chapter08/ExerciseFiles\_CH08/Exercise/DisgruntledDan.flv)
- ThroughAdoor.flv (Chapter08/ExerciseFiles\_CH08/Exercise/ThroughAdoor.flv)
- Control.fla (Chapter08/ExerciseFiles\_CH08/Exercise/Control.fla)
- Captions.flv (Chapter08/ExerciseFiles\_CH08/Exercise/CaptioningVideo/Captions.flv)
- captionsFLV.xml (Chapter08/ExerciseFiles\_CH08/Exercise/CaptioningVideo/captionsFLV.xml)
- Alpha.mov (Chapter08/ExerciseFiles\_CH08/Exercise/Alpha.mov)
- DisgruntledDan.flv (Chapter08/ExerciseFiles\_CH08/Exercise/FullScreenSkin/DisgruntledDan.flv)
- Apparition.flv (Chapter08/ExerciseFiles\_CH08/Exercise/Apparition.flv)
- RainFall.fla (Chapter08/ExerciseFiles\_CH08/Exercise/Rain.fla)

- Rain.flv (Chapter08/ExerciseFiles\_CH08/Exercise/Rain.flv)
- BlobEffect fla (Chapter08/ExerciseFiles\_CH08/Exercise/BlobEffect fla)
- CuePoints.xml (Chapter08/ExerciseFiles\_CH08/Exercise/YourTurn/CuePoints.xml)
- VideoJam fla (Chapter08/ExerciseFiles\_CH08/Exercise/YourTurn/VideoJam fla)

The authors would like to take this time to thank William Hanna, Dean of the School of Media Studies, at the Humber Institute of Technology and Advanced Learning in Toronto, and Robert O'Meara, a faculty member with the Film and Television Arts program at Humber, for permission to use the videos in this chapter. The videos were produced by students of the Interactive Multimedia and Film and Television programs at Humber.

## Video on the Web

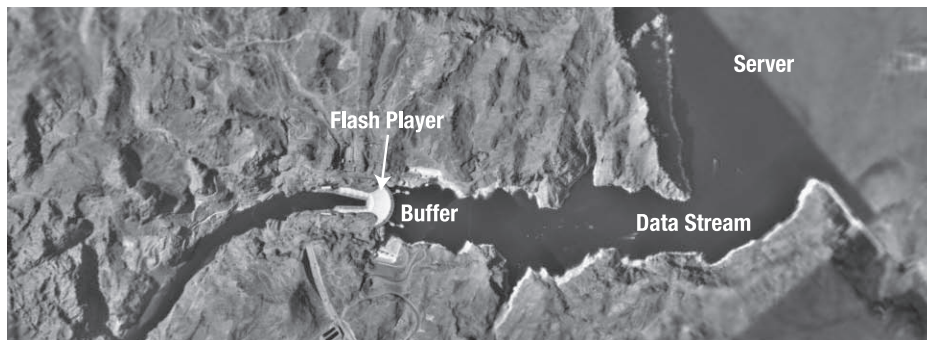
Before we turn you loose with creating and playing Flash video, it is critically important that you understand how it gets from the server to the user's machine.

The Flash video format uses the .flv extension. It can't be played anywhere else other than in Flash or through the use of a third-party Flash video player such as Riva FLV Player ([www.rivavx.com/index.php?id=422&L=3](http://www.rivavx.com/index.php?id=422&L=3)), Fluffy ([www.nothing.ch/research/applications/43](http://www.nothing.ch/research/applications/43)), or one offered by long-time Flash developer Martijn de Visser ([www.download.com/FLV-Player/3000-2139\\_4-10467081.html](http://www.download.com/FLV-Player/3000-2139_4-10467081.html)) that will play FLV files on your desktop. The key thing about this format is that the data is sent to the user's computer from the server where it is played by Flash Player. To help you understand this process, let's go visit the Hoover Dam in the United States.

The Hoover Dam was built in the 1930s to control the Colorado River. When the dam was completed, the water behind it backed up to form Lake Mead. This means the water flows along the Colorado River into Lake Mead, and the dam releases the water in the small lake directly behind it, in a controlled manner, back into the Colorado River. The thing is, if the water rushes to the dam and overwhelms it or the dam operator releases too much water, the people downstream from the dam are in for a really bad day.

Streaming video is no different from the water flow to the Hoover Dam and beyond.

The data in the FLV is sent, at a data rate established when the video was encoded, from the server to Flash Player, where it is held in a buffer and released, in a controlled manner, by Flash Player to the browser. If the flow is too fast—the data rate is too high for the connection—the browser is overwhelmed, and the result is video that jerkily stops and starts. This is due to the buffer constantly emptying and having to be refilled. In many respects, your job is no different from that of the crew that manages the flow of water from the buffer behind the Hoover Dam back into the Colorado River. When you create the FLV, the decisions you make will determine whether or not your users are in for a really bad experience (see Figure 8-1).



**Figure 8-1.** When it comes to Flash video, you control the Hoover Dam.

## Encoding an FLV

The first step in the process of creating the FLV file that will be used in the Flash movie is to convert an existing video to the FLV format. This means you will be working with digital videos that use the following formats:

- **AVI (Audio Video Interleave):** A Windows format that supports a number of compression schemes but also allows for no compression
- **DV:** The format used when video moves directly from a video camera to the computer
- **MPG/MPEG (Motion Pictures Experts Group):** A lossy standard for video that is quite similar to the lossy JPG/JPEG standard for images
- **MOV:** The QuickTime format

Do yourself and your user a favor and check out the compression used to create the video. If a lossy compressor was used, you are going to have a serious quality issue. The compressors used to create FLV files are also lossy, meaning you will be compressing an already-compressed video. You can check to see which compressor was used in either Windows Media or QuickTime by selecting **File** ► **Properties** in the Windows Media Player or **Window** ► **Show Info** in the QuickTime player. The resulting dialog box, shown in Figure 8-2, will indicate the compressor used.



**Figure 8-2.** Apple Lossless animation compressor is used.

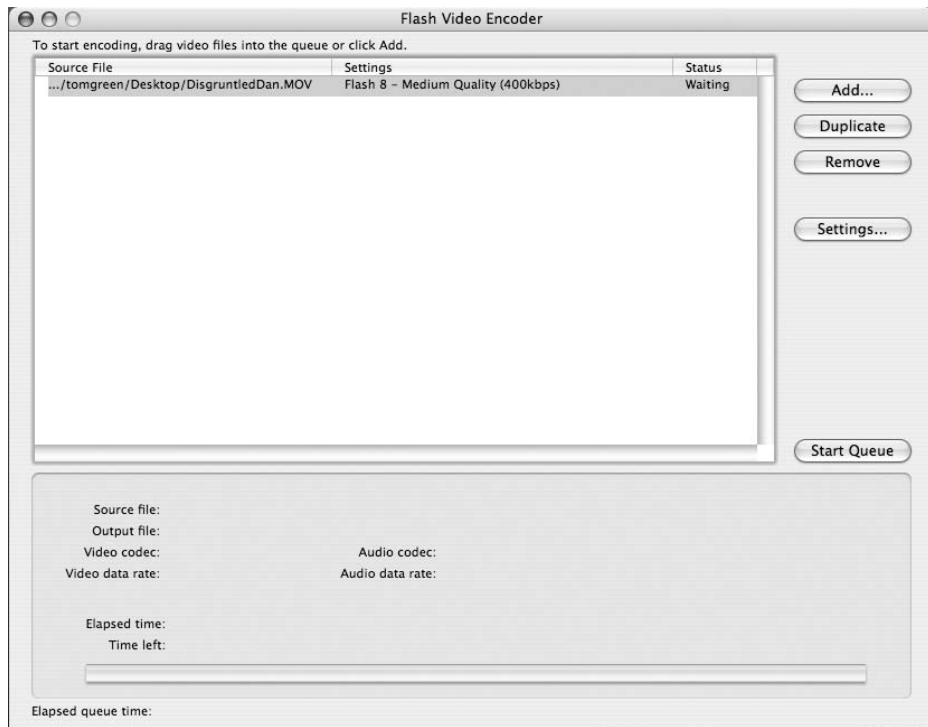
Surprisingly, the first step in the conversion process has absolutely nothing to do with Flash. Instead, open the video in your player of choice and watch the video twice. The first time is to get the entertainment/coolness factor out of your system. The second time you watch it, ask yourself a few questions:

- Is there a lot of movement in this video?
- Is the audio of major importance?
- Is there a lot of color in the piece?
- Is the video in focus, or are there areas where the image becomes pixelated?

The answers to these questions will determine your approach to encoding the video. The file you will be encoding is `DisgruntledDan.mov`. Go ahead, open it up in QuickTime and watch it twice.

Yes, the file is huge—277 MB. There is a reason. When creating Flash video, you need every bit of information contained in the video when you do the conversion. Uncompressed video is about as big as it gets. When you finish converting the video into an FLV, you will be in for a rather pleasant surprise.

1. Open the Adobe Flash Video Encoder found in C:\Program Files\Adobe\Adobe Flash Video Encoder on a PC or Macintosh HD\Applications\Adobe Flash Video Encoder on a Mac. When the Encoder opens, as shown in Figure 8-3, drag a copy of the *DisgruntledDan.mov* file into the render queue. Alternatively, you could click the Add button or select File ► Add and, using the Open dialog box, navigate to your Exercise folder, select the video, and click the Open button to add the video to the FLV Encoder.



**Figure 8-3.** A file is in the render queue waiting to be encoded.

2. Click the Settings button to open the Encoding Settings window shown in Figure 8-3. As you can see, this window is broken into two areas. At the top is a Preview area. Under this window is the current time indicator. It displays time in the format hours:minutes:seconds:milliseconds. The triangle at the top of the line is the jog controller. If you drag it back and forth, the video will follow along. Underneath the jog controller are two other triangles. The one on the left is the in point, and the one on the right is the out point. You can use these to trim the video. For example, assume there are 2 seconds of black screen and no audio at the end of the video. If you drag the out point to the start of the stuff you don't need, it will be removed when you create the FLV.

Here's a neat little trick: the preview controls are very precise, and reaching a precise point in time can be an exercise in tediousness. Assume you want the current video to last 3 minutes and 27 seconds instead of 03:27:266. Select the out point and press and hold the left arrow key. When the key is down, the milliseconds measure will reduce. When you are close to the 000 milliseconds point, release the key and then press it in slow succession. The millisecond number will reduce in 1-millisecond increments.

As shown in Figure 8-4, the bottom half of the window consists of a series of tabs that allow you to choose a preset encoding profile (not a good idea, and more on that later on), set the video compression and the audio compression, add cue points that can be accessed by ActionScript, and crop and resize the video.



**Figure 8-4.** The Encoding Settings dialog box allows you to choose a preset encoding profile and to set the in and out points for the video.

3. Click in the Output filename input box and enter DisgruntledDan.

If you have used the Flash Video Encoder prior to this release, you may notice the addition of a couple of buttons above the Encoding Profile drop-down menu. New to Flash CS3 is

the ability to save your custom settings as a profile and also to load that custom profile and use it. The other major change is the video and audio portion of the FLV Encoder have been given separate panels.

4. Click the Video tab to open the Encode Video panel shown in Figure 8-5. This is where you set the all-important video data rate. The various areas of the panel are as follows:
  - Video codec: You have two choices: On2 VP6 and Sorenson Spark. If your target Flash Player is Flash Player 7 or lower, your only choice is the Sorenson Spark codec.
  - Encode alpha channel: If your video contains an alpha channel, select this. Alpha channel video can only be encoded using the On2 VP6 codec.
  - Deinterlace: If your source video was prepared for television broadcast, the odds are almost 100% it was interlaced. Select this option to remove it.

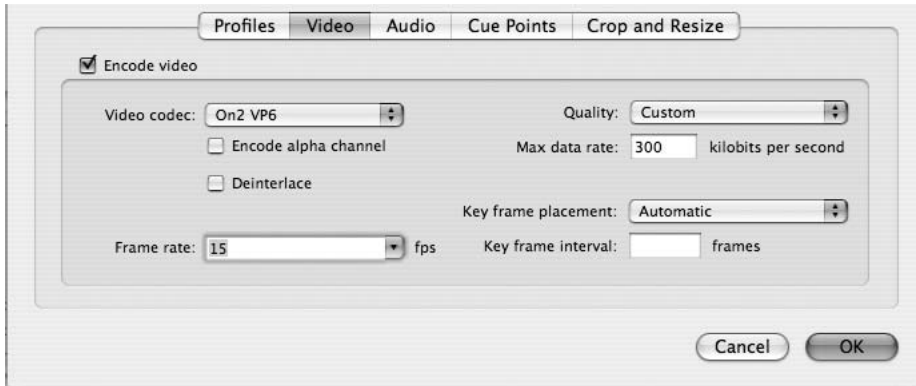
*Interlacing? Huh? Your TV screen shows alternate lines of the signal when it is playing. They appear so fast, the human eye is tricked into seeing them as a solid screen. The technique of splitting a video into alternating lines for TV broadcast is called **interlacing**. In many respects, the Encoder is not the place to do this. If you are receiving a file prepared for TV broadcast, ask the supplier to provide you with a deinterlaced, uncompressed version of the video.*

- Frame rate: This determines how often the video updates. The measurement is frames per second (fps). If you are unsure of which frame rate to use, a good rule of thumb is to choose a rate that is half that of the original file. If the original was prepared using the NTSC standard of 30 fps, select 15 fps. If the PAL standard was used, rates of 12 or 15 fps are acceptable.
- Quality: Choose a preset from this drop-down to set the data rate for the video track. You can also select Custom to enter your own value.
- Max data rate: You can choose the rate to be used. If you change the value, the Quality setting will change to Custom.

Be very, very careful when choosing a quality setting. For example, don't think you can "super size" the quality and set the data rate to, say, 1000 kilobits per second. Do that, and you can guarantee that residents downstream from the Hoover Dam are in for a really, really bad day. Also, you need to know the Max data rate setting is a bit misleading. That rate is for the video portion only. The data rate for an FLV is the sum of the audio and the video data rates. So what to choose? Until you become comfortable with creating FLV files, consider a combined audio and video data rate of around 350 kilobits per second as being a fair target.



- Key frame placement: This is one of those areas where, unless you have mastered video, it is best to let the software do the work.
- Key frame interval: Enter a value here, and the Key frame placement selection will change to Custom.



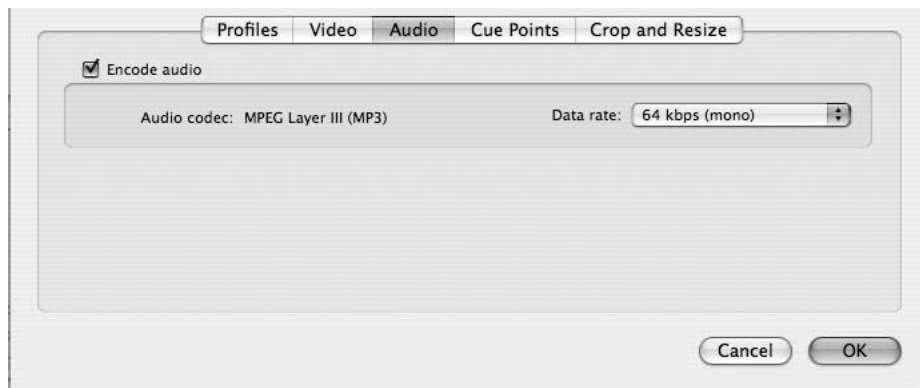
**Figure 8-5.** Setting the encoding values for the video portion of the movie

Remember that first question you were to ask—Is there a lot of movement?—at the start of the chapter? The answer determines key frame placement. If you are recording paint drying, having a keyframe every 300 frames of the video would work. If you are encoding a video of a Formula One race from trackside, you will want the keyframes to be a lot closer to each other, such as every 30 frames or so.

5. Specify the following values in the Video pane. When you finish, click the Audio tab, not the OK button, to open the Audio settings pane.
  - Video codec: On2 VP6
  - Quality: Custom
  - Max data rate: 300
  - Key frame placement: Automatic
  - Frame rate: 15
6. The Audio pane, shown in Figure 8-6, is where you manage the audio quality. You have to make two decisions:

- Stereo or mono?
- What will be the data rate?

Select 64 kbps (mono) from the Data rate drop-down menu. In fact, your two choices should be 48 kbps or 64 kbps. Anything lower results in an increasing degradation of audio quality. Still, 32 kbps is a good choice if the soundtrack is nothing more than a voiceover, and 16 kbps is ideal if the soundtrack is composed of intermittent sounds such as the frogs and wolves used in the Lake Nanagook project that started this book.



**Figure 8-6.** Setting the data rate for the audio portion of the movie

Unless there is a compelling reason—you are encoding a band’s video, for instance—staying with a mono setting should be your first choice. Outputting stereo will only serve to increase the final file size of the FLV.

Don’t think you can improve the audio track by outputting it as a stereo track if it was originally recorded in mono. Sure, you can change mono to stereo in these settings, but all you get are two identical mono tracks. It’s wasted bandwidth. Also, as we pointed out in Chapter 5, the default format for all audio in Flash is MP3. This explains why you only have that one choice in the Audio pane.

7. Click the Crop and Resize tab. You aren’t going to do anything here, but there is an aspect of this pane that you need to know about. When you click the tab, the Crop and Resize pane opens, and you can see the pane is split into three areas: Crop, Resize, and Trim. We aren’t concerned with the Crop and Trim areas. The Resize area, shown in Figure 8-7, is critical to your survival.

When digital video is created for your television, it is created at a 4:3 ratio. This ratio is called the video’s **aspect ratio** and fits most computer monitors. Other common examples would be widescreen TV video, which has an aspect ratio of 6:5, and HDTV, which uses a 16:9 aspect ratio.

For example, the video you are encoding has a physical size of 320 pixels wide by 240 pixels high. The width is easily divisible by 4, and the height is divisible by 3. If you need to resize a video, be sure to select *Maintain aspect ratio*. This way you avoid introducing artifacts (blocky shapes and other nastiness) into the video when it is resized.

While we are on the subject of resizing video, never increase the physical size of the video. If you need to change the size, use this area to reduce, not increase, the width and height values. Increasing the physical dimensions of the video from 320 by 240 to 640 by 480 will only make the pixels larger, just as it does in Photoshop and Fireworks. The result is pixelated video, and it will also place an increasing strain on the bandwidth.

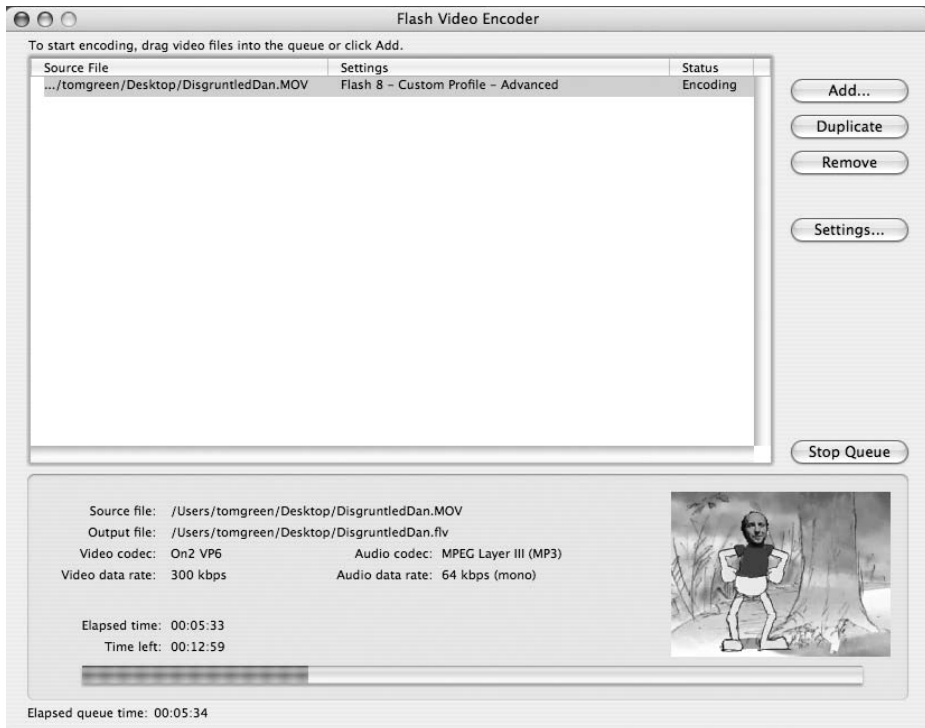


**Figure 8-7.** Setting the Data rate for the audio portion of the movie

In spite of our having said to never increase the size of a video, Flash Player 9 now permits full-screen video playback. We'll review this feature later on in the chapter.

8. Click OK to return to the render queue. Click the Start Queue button to start the process.

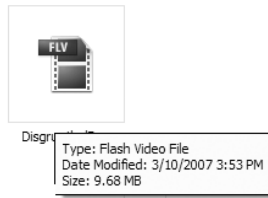
You will see the progress bar move across the screen as the video is being rendered, and you will also see the video being rendered in the Preview area shown in Figure 8-8. If you click the Stop Queue button, you will see a dialog box asking you whether you wish to stop the process or finish the render. If you have a number of videos in the queue, clicking the No button in the dialog box will stop the process, and an Errors dialog box will appear telling you that you stopped the render process. If you want to make changes to the settings or restart the render process, select the video—its status will be set to Skip in the Status area—and select Edit ► Reset Status.



**Figure 8-8.** Rendering an FLV

Here's an unknown technique that will make your life much less stressful. Selecting a video in the render queue and clicking the **Remove** button will remove it from the render queue. What if you have made a mistake and need to make a simple change to the video or audio settings? If the video is still in the render queue and its status is set to either **Skip** or **Completed**, you can select the video and select **Edit ▶ Reset Status** to put it back into the render queue, and clicking the **Settings** button will return you to the original video and audio settings. This is really handy in situations where you have messed up a cue point or two. For this to work, though, you can't move the video from its original folder.

9. When the encoding is complete, a green check mark will appear in the Status area. Close the FLV Encoder and open the Chapter 8 Exercise folder. If this is the first time you have used the FLV Encoder, you had better sit down. You will notice the FLV and the QuickTime movie are in the same folder. Check out the file size of the FLV. The size, as you see in Figure 8-9, has plummeted from 277 MB to 9.7 MB. Don't panic, this is common with the FLV Encoder. Remember, the On2 VP6 codec is lossy, and it really spreads out the keyframes. Both of these combine to create significant file-size reductions. This also explains why it is so important that the source video not be encoded using a lossy codec.



**Figure 8-9.** It is not uncommon to have an FLV shrink to 10% or less of the original file size.

## Playing an FLV in Flash CS3

Having encoded the video, the time has arrived to have it play in Flash. There are three ways to accomplish this task, listed here, and we are going to show you each method:

- Let the Import Video wizard do it for you.
- Use the FLVPlayback component.
- Use a video object.

The first two are actually variations on the same theme. Both will result in the use of the FLVPlayback component. The difference is the workflow. They each approach the task from opposite angles. The final method is the most versatile but involves the use of ActionScript. Regardless of which method you may choose, the end result is the same: you are in the video game.

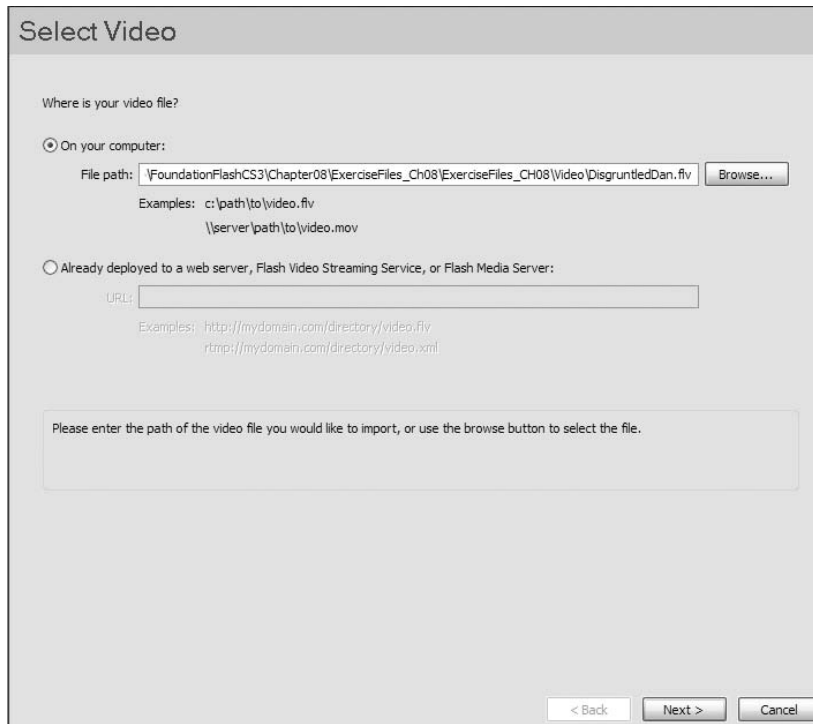
### Using the Import Video wizard

This example covers the steps involved in actually adding video to Flash. If you have never used Flash video, this is a great place to start. Let's get going:

1. Open a new Flash document and select File ► Import ► Import Video. This will open the Import Video wizard.
2. The first step in the process is to tell the wizard where your file is located. Click the Browse button and navigate to the folder where you placed the FLV created in the last exercise, or use the `DisgruntledDan.flv` file in your Chapter 8 Exercise folder. When the path is established, click the Next button to open the Deployment screen.

There are only two possible locations for a video: your computer or a web server. If the file is located on your computer, the Browse button allows you to navigate to the file, and when you select it, the path to the file will appear, as shown in Figure 8-10, in the File Path area. This rather long path will be trimmed, by Flash, to a relative path when you create the SWF that plays the video. The second choice requires you to add an absolute path to the file. If you have a lot of videos, you may have them located in a folder on your website. In this case, the path to `DisgruntledDan.flv` would be `http://www.mySite.com/FLVfile/DisgruntledDan.flv`. The path to the Flash Video Streaming Service or Flash Media Server would be a bit different. You would add a path that looks something like this: `rtmp://myHost.com/Dan`.

We won't be getting into the use of Flash Video Streaming Service or Flash Media Server in this book. All videos will be played back either locally or through an HTTP site.



**Figure 8-10.** Setting the path to an FLV using the Import Video wizard

3. The Deployment screen, shown in Figure 8-11, tells Flash how the video will be streamed into Flash Player and ultimately through the browser. Select Progressive download and click the Next button to open the Skinning pane.

As you can see, there are six deployment options. Here's what they mean:

- Progressive download from a web server: This option is one of the most common video delivery methods on the Web. In fact, it is the method used by YouTube to deliver video. A progressive download means Flash Player is constantly checking how much of the video has arrived, and if there is enough to start playing the video, the video starts to play. Though you might have inferred from this that there will be an inordinate wait time, this is simply not true. Usually only about one-half second of the video has to load before the video starts to play.

Although this is the most common option out there, it is also the least secure. The FLV file is downloaded into the browser cache, and if you are smart, you can copy it and use elsewhere. This is why such companies as recording studios, television networks, and movie studios have some sort of jihad against this format, because once the FLV arrives in the user's cache, they potentially lose control of the file's usage. If your client is adamant that the content rights must be protected, this option is not the one for you.

- Stream from Flash Video Streaming Service: There are a number of companies that will host and stream your video for you using Flash Media Server technology. It is quite secure—nothing arrives in the browser cache—and the network of servers used by these companies ensures your content is played on demand. You can find out more about this solution at [www.adobe.com/products/flashmediaserver/fvss/](http://www.adobe.com/products/flashmediaserver/fvss/).
- Stream from Flash Media Server: You either own your own server or use the services of an ISP to set up a media server account. Two companies that we have been exposed to are NI Solutions in Toronto ([www.nisolutions.ca](http://www.nisolutions.ca)) and Influxis located in Los Angeles ([www.influxis.com](http://www.influxis.com)). You can also try this out for yourself and learn how to use it by visiting the Flash Media Server Development Center on the Adobe site ([www.adobe.com/devnet/flashmediaserver/](http://www.adobe.com/devnet/flashmediaserver/)).
- As mobile device video bundled in SWF: This option essentially embeds the entire video into the Flash timeline in Flash Lite 2.0 and 2.1 Players used by cell phones and other mobile devices. This option is grayed out because you are targeting Flash 9 Player. We'll deal with mobile features in greater depth in Chapter 12.
- Embed video in SWF and play in the timeline: Not a good idea with this video, but a great idea if you have clips that are 1 or 2 seconds in duration.

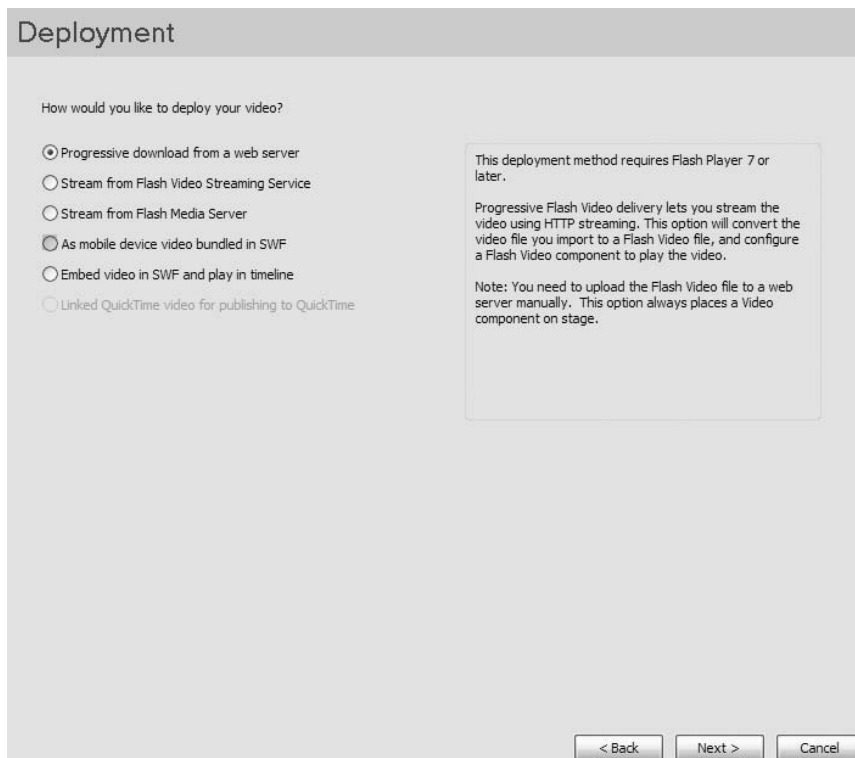
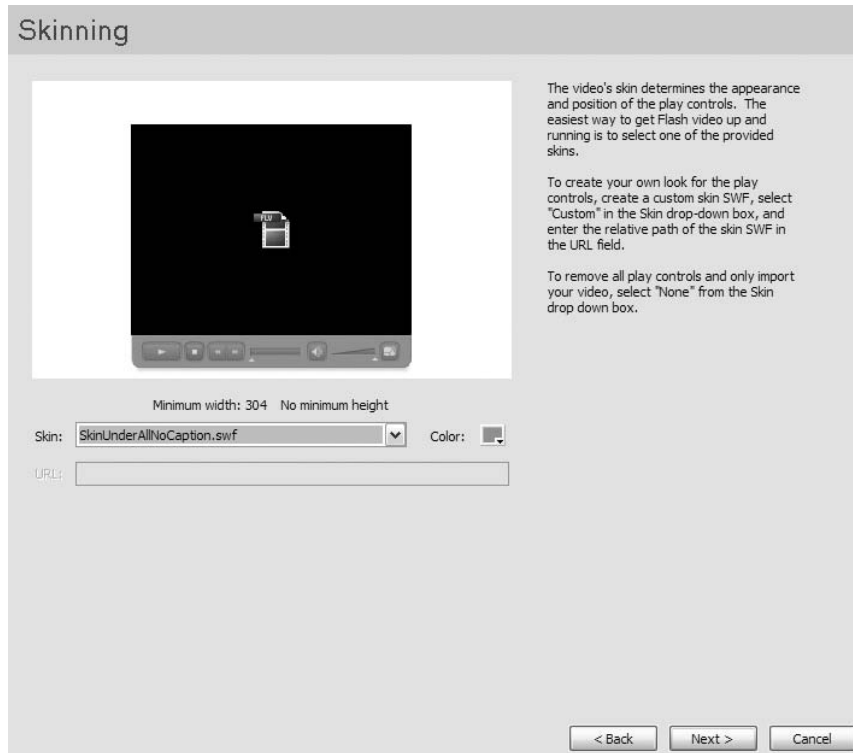


Figure 8-11. How will the video be deployed?

- Click the Skin drop-down menu to see the choices available to you. Click a skin style and the Preview area, shown in Figure 8-12, will change to show you the skin chosen. Click the color chip to open the Color Picker, choose a color, and the skin will change to that color. Select `SkinUnderAllNoCaption.swf` and pick a color. Select `SkinUnderAllNoCaption.swf` and pick a color.



**Figure 8-12.** What skin or control style will be used?

Skin? Think of it as a techie word for video controls.

Selecting None in the Skin drop-down means there will be no skin associated with the video. Choose this option if you are going to create your own custom controls or use the components in the Video area of the Components panel.

Pay close attention to the minimum width for each skin. For example, selecting `SkinUnderAll.swf` requires a video that is at least 330 pixels wide. Considering our video is 320 pixels wide, the skin is going to hang off of the sides of the video. You can see this in the preview.

This is a big change from the previous versions of Flash. You are essentially presented with two major skin groupings: **Over** and **Under**. Controls containing the word *Over* will place the control over the video, and the controls will be visible, if this option is chosen, when the user places the cursor over the video. Controls containing the word *Under* place the controls below the video, and they are always visible.



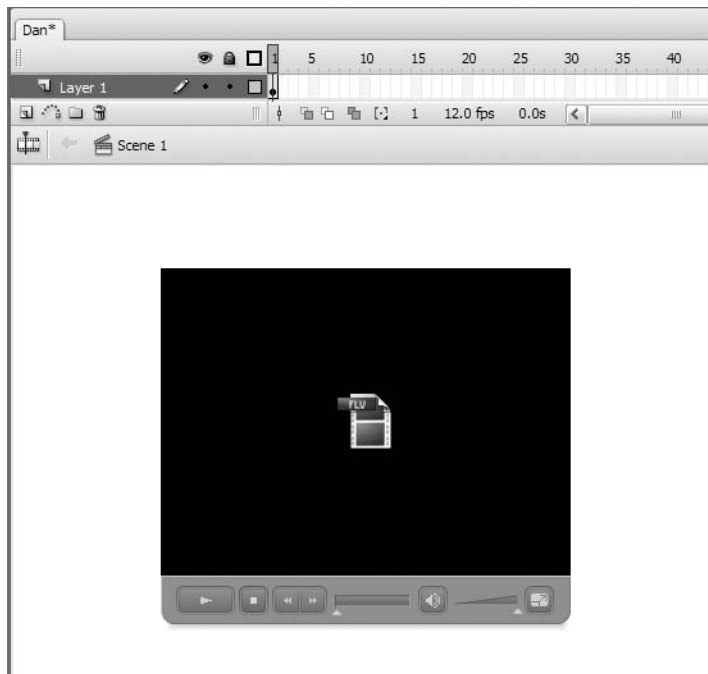
The URL input area lights up if you select Custom Skin URL in the Skin drop-down menu. If you have created a custom skin such as one containing a client's branding, you would enter the path or the HTTP address to the skin's location.

The ability to add a custom color to a skin is also a major improvement. This way you can, for example, use a client's corporate color in the controls . . . something unavailable to you without a lot of work in previous versions of Flash. You can even make the color semi-transparent—extremely useful in an Over skin—by setting the alpha to less than 100%.

5. Click the Next button to be taken to the Finish Video Import screen. This screen simply tells what will happen when you click the Finish button at the bottom of the pane.

The most important thing that will happen is you will be prompted to save the FLA file to the same folder as the FLV you linked to. The FLVPlayback component needs this path to ensure playback of the video. When the Save As dialog box opens, make sure you navigate to the folder containing your FLV. Name the file, and click the Save button to return to the Finish Import pane. Click the Finish button.

You will see a progress bar showing you the progress of the video being added to the Flash stage. When it finishes, the FLVPlayback component, shown in Figure 8-13, will be placed on the Flash stage.



**Figure 8-13.** The video is “good to go.”

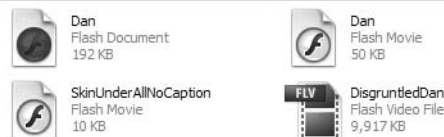
6. Click the video on the stage, and in the Property inspector, set its x and y coordinates to 0. Save the movie and test it. The video will start playing, as you see in Figure 8-14, in Flash Player. Feel free to try out the controls. Congratulations, you are in the video game.



**Figure 8-14.** Welcome to the video game.

7. Close the video in the SWF to return to the Flash movie. Select **Modify > Document** and, when the Document Properties dialog box opens, click the contents radio button to shrink the stage to the video and click OK to close the dialog box. Select the component on the stage and press the left or right arrow key a few times. Holy smokes! The controls, shown in Figure 8-15, are hanging off the stage. If the controls are hanging off of the stage, the odds are good, depending upon the embedding options in the HTML, they won't be visible on the web page. What's with that?

The simple answer is *This is a "gotcha," applicable only to the Under skins, that you need to be aware of.* When you use the FLVPlayback component, only the component is seen when you shrink the stage. The controls, which are a separate SWF added at runtime, aren't. If you are shrinking the stage and the only content on the stage is the FLVPlayback component, do yourself and your sanity a favor and manually change the stage dimensions. The width can be set to the width of the FLV, but add about 45 pixels to the height of the stage to accommodate the skin.



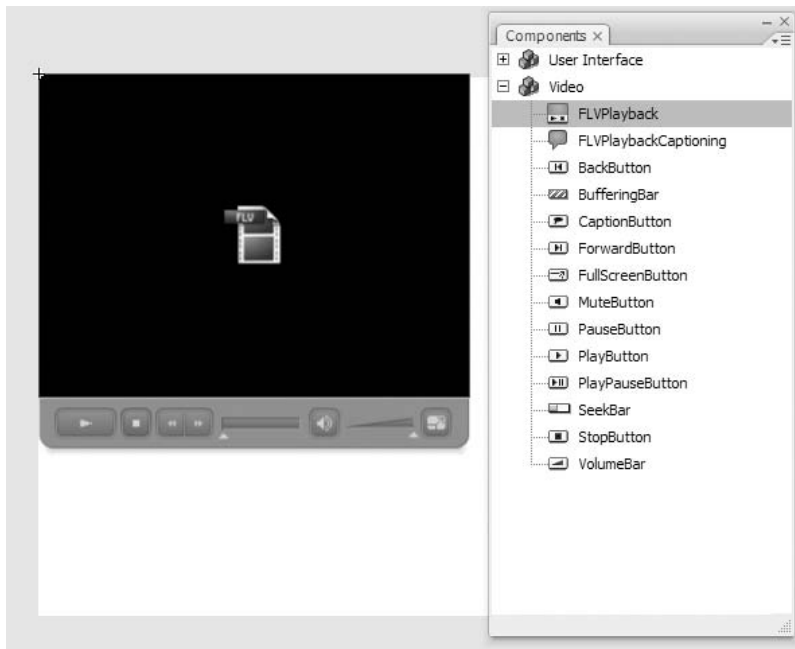
**Figure 8-15.** The two SWF files and the FLA must be in the same directory if you are uploading to a web page.

8. Change the stage dimensions to 320 by 285. Save the movie and test it.
9. There is one last thing you need to know before we move on. Open the Chapter 8 Exercise folder, which contains the FLV. As you see in Figure 8-15, it contains a number of files: the FLA, the SWF, another SWF containing the name of the skin, and the FLV. If you are going to be embedding this particular project into a web page, you must move the two SWFs and the FLV to the same directory on your website. If they are not in the same folder, the video will either not play or the controls won't be available. Why? Because we haven't concerned ourselves with the complexities of file paths in this exercise. Putting these files in the same folder equates to the least amount of hassle.

## Using the FLVPlayback component

In the previous exercise, you used the Import Video wizard to connect an FLV to the FLVPlayback component. In this exercise, you'll be doing the process manually. Once you are comfortable with it, you will discover this method to be a lot quicker than the previous one. Follow these steps:

1. Open a new Flash document and save it to your Chapter 8 Exercise folder. Remember, the FLA needs to be in the same folder as the FLV.
2. If it isn't open, open the Components panel by selecting Window ► Components. When the panel opens, click the Video category. Drag a copy of the FLVPlayback component, shown in Figure 8-16, onto the stage. When you do this, the first thing you will notice is the component has the same skin color from the previous exercise. This is normal. Also, if you open the library, you will see a copy of the component has been added to the library. This is a handy feature because you can use the library, not the Components panel, to add subsequent copies of the FLVPlayback component to the movie.

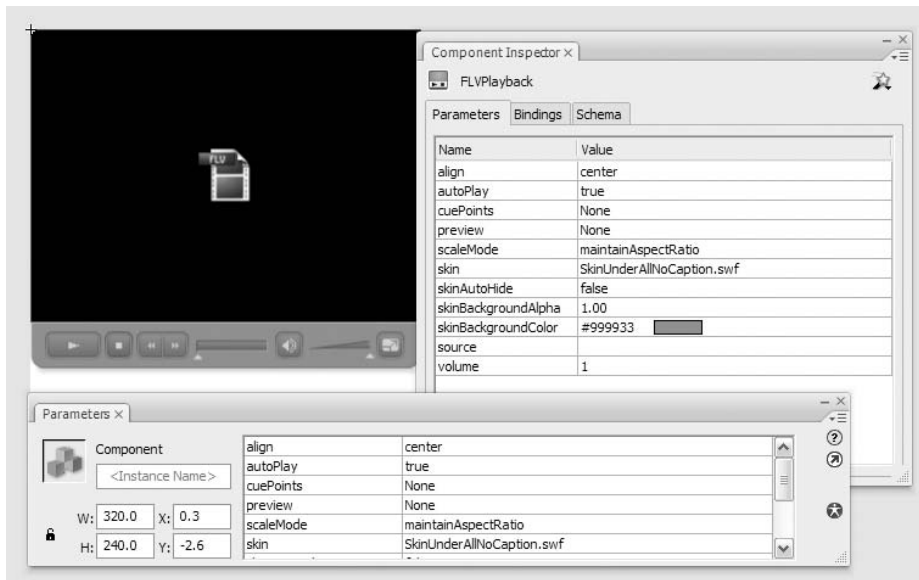


**Figure 8-16.** The FLVPlayback component is found in the Video section of the Components panel.

3. Click the component on the stage and click the Parameters tab of the Property inspector. The parameters, listed here, allow you to determine how the component will function:

- align: The choices in this drop-down menu have nothing to do with the physical placement of the component on the Flash stage. The choices you make here will determine the position of the FLV in the playback area of the component if the component is resized.
- autoPlay: Choose true, the default, and the video plays automatically. Select false, and the user will have to click the Play button in the component to start the video. In either case, the FLV file itself starts downloading to the user's computer, so keep this in mind if you put several FLV-enhanced SWFs in a single HTML document.
- cuePoints: If cue points are embedded in the FLV, they will appear in this area.
- preview: This feature is new to the component. If you select this, and an FLV is connected to the component, you can see the video without having to test the movie.
- scaleMode: Leave this at the default value—maintainAspectRatio—if video is to be scaled.
- skin: Select this, and the Select Skin dialog box will appear.
- skinAutoHide: Choose true, and the user will have to place the mouse over the video for the skin to appear. This only applies to skins that appear over the video.
- skinBackgroundAlpha: Your choices are any two-place decimal number from 0 to 1. 0 means the background is totally transparent and 1 means there is no transparency. 0.5 is semitransparent by 50%.
- skinBackgroundAlpha: Select this, and the Flash color chips appear.
- source: Click the Magnifying Glass icon, and the Content Path dialog box opens. From here you can either set a relative path to the FLV or enter an HTTP or RTMP address path to the FLV.
- volume: The number you enter—any two-place decimal number between 0 and 1—will be the starting volume for the video.

There is another place to see these parameters. Select Window ► Component Inspector and the Component Inspector panel, shown in Figure 8-17, will appear. Click the Parameters tab to bring up the FLVPlayback component parameters. We will be using this panel to show you the parameters for the component. The reason is that this panel, unlike the Parameters area of the Property inspector, shows you all of the parameters without scrolling. This makes things easier for you to follow.



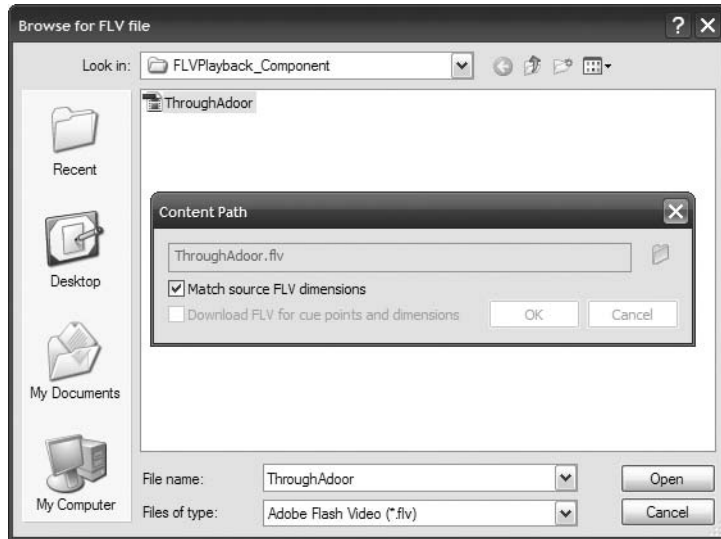
**Figure 8-17.** The FLVPlayback component uses parameters that can be set either in the Property inspector or the Component Inspector panel to determine its look and functionality.

4. With the component selected on the stage, use the following parameter values:

- autoPlay: false
- skinBackgroundColor: #999999 (medium gray)
- source: ThroughAdoor.flv

When you click the source parameter, be sure to click the Magnifying Glass icon to open the Content Path dialog box shown in Figure 8-18. Click the Navigate button—the File Folder icon—which opens the Browse for FLV file dialog box. Navigate to the Chapter 8 Exercise folder, select the video, and click the Open button to close the dialog box. The relative path to the FLV will appear in the Content Path dialog box. Also be sure to select the Match source FLV dimensions check box. Selecting this will size the component to the exact dimensions of the FLV file.

5. Save and test the movie in Flash Player. Click the Play button to start playing the video. When you have finished, close the SWF to return to the Flash stage.



**Figure 8-18.** Setting the content path to the FLV to be played in the component

6. Select the component on the stage and click the Parameters tab in the Property inspector.
7. Click the preview parameter and click the Magnifying Glass icon to open the Select Preview Frame dialog box (see Figure 8-19). Here you can watch a live preview of the video. Click Cancel to close the dialog box.



**Figure 8-19.** Live preview is new to Flash CS3.

This isn't the only purpose of the preview. The FLV controls in the dialog box are live, meaning you can scrub to a frame of the video. If you click OK, the frame will appear in the component, and the time of the frame will appear beside the `preview` parameter. This image is there only to show you how the video will appear in the component. This preview is only used at authoring time—think of it as a position-only graphic—and won't appear in the final SWF. To use the image as a poster frame or a graphic, click the Export button. The Export Image dialog box will appear, and you can save the image as a Fireworks PNG file and use it with ActionScript or import it into the library.

Why would you want to export a frame of the video? Frames can be used as movieclips or buttons to launch a video or as navigation elements to move the timeline, or even the web page, to where the video is located.

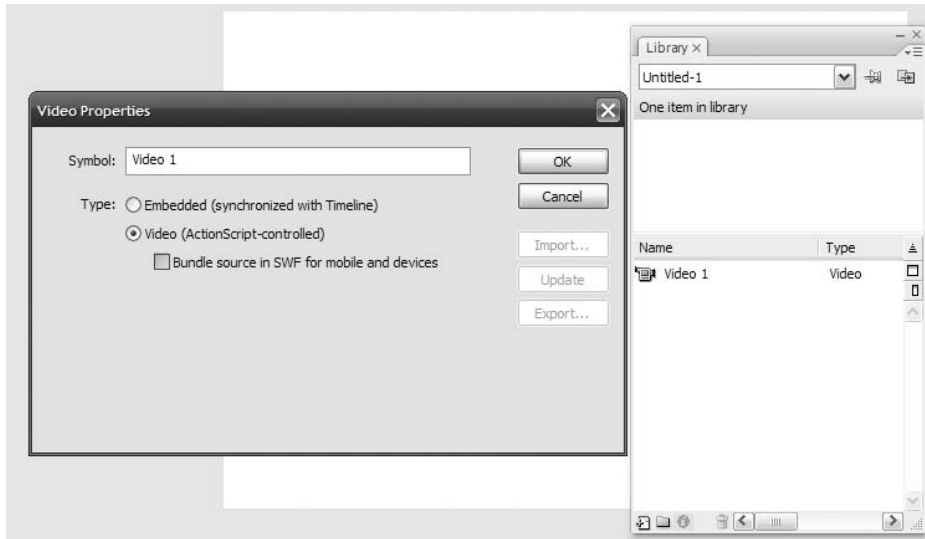
## Playing video using ActionScript

In the previous two exercises, you have seen different ways of getting an FLV file to play through the `FLVPlayback` component. In this exercise, you won't be using the component; instead, you'll let ActionScript handle the duties. It is a lot like connecting your new TV to the cable in an empty room. There are essentially three steps involved:

- Connect
- Stream
- Play

When you walk into the room where you are about to hook up the TV to the cable, the TV is sitting on a shelf, and there is a spool of coaxial cable sitting on the floor. When you screw the cable into the wall outlet, you are establishing a connection between the cable company and your home. When you screw the other end of the cable into the TV, the TV is now connected to the cable company. When you turn on the TV, the show that is flowing from the cable company to your TV starts to play. Let's connect our TV to an FLV. Here's how:

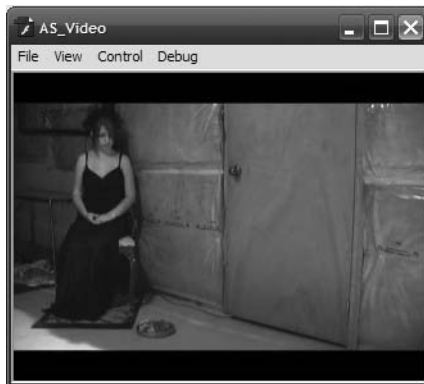
1. Open a new Flash document, and then open the Flash library. If you don't have the library in your panel group, select **Window** ► **Library** to open the library.
2. Click the library drop-down menu in the upper-right corner of the panel and select **New Video**. The **Video Properties** dialog box will open (see Figure 8-20). Make sure the **Video (ActionScript-controlled)** radio button is selected, and click **OK** to close the dialog box. If you open the library, you will see there is a little video camera named **Video 1** sitting in your library. This camera is called a **video object**, and it will be your TV.



**Figure 8-20.** Creating a video object that will play an FLV

3. Drag your video object from the library to the stage. When you release the mouse, it will look like a box with a big X through it. After a bit of ActionScript, it will display video, as shown in Figure 8-21. Click the video object and specify these values in the Property inspector:
  - Instance name: myVideo
  - Width: 320
  - Height: 240
  - X: 0
  - Y: 0

When you have finished, save this file to the Chapter 8 Exercise folder.



**Figure 8-21.** Eight simple lines of ActionScript code drive the playback of this video.



4. Add a new layer named `Actions`. Select the first frame of the `Actions` layer, open the `Actions` panel, and enter the following code:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
myVideo.attachNetStream(ns);
```

The first line establishes the `NetConnection` between the player and the server. The second line tells the player it is an HTTP connection, not an RTMP connection. The third line establishes the stream and, finally, the fourth line connects the video object named `myVideo` to the stream that is connected to the server.

5. Press Enter (PC) or Return (Mac) twice and enter the following code:

```
var listener:Object = new Object();
listener.onMetaData = function(md:Object):void {};

ns.client = listener;
```

If you don't have this listener in the code, you are going to have very mysterious compiler errors coming out of your ears. The reason is that most FLV files have metadata contained in them. For example, the duration or length of the file is often contained in the FLV metadata. ActionScript 3.0, and for that matter Flash Player 9, are trained to look for that metadata, and if they don't find it, they get a little frantic and fill your output panel with this sort of error message:

```
Error #2044: Unhandled AsyncErrorEvent:. text=Error #2095:
flash.net.NetStream was unable to invoke callback onMetaData.

error=ReferenceError: Error #1069: Property onMetaData not found ➡
on flash.net.NetStream and there is no default value.
at DanCode_fla::MainTimeline/ThroughAdoorfla::frame1()
```

The listener object and the `onMetaData` handler function team up to “chill out” ActionScript because you don't need to actually “do” anything with the event handler. You can if you want, but all you *have* to do to avoid errors is handle the event.

By setting the `client` property of the `NetStream` instance to the listener object, you have effectively told Flash CS3 to ignore the metadata in the FLV. This is the purpose of the last line.

6. Press Enter (PC) or Return (Mac) twice and enter the following code:

```
ns.play("ThroughAdoor.flv");
```

This line uses the `NetStream.play()` method to actually stream the FLV file into the video object on the stage. The important thing to note here is that the name of the video is a string because it is between quotation marks and the `.flv` extension is added to the name of the video.

To recap:

If you want to play video using ActionScript, here is all of the code you will need to get yourself started:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
var ns:NetStream = new NetStream(nc);
myVideo.attachNetStream(ns);

var listener:Object = new Object();
listener.onMetaData = function(md:Object):void {};
ns.client = listener;

ns.play("ThroughAdoor.flv");
```

The only thing you will ever need to do to reuse this code is to make sure the video object's instance name matches the one in line 4 and change the name of the FLV file in the last line.

7. Save and test the movie. When Flash Player opens, the video, as shown in Figure 8-21, starts to play.

You are probably thinking, “Hey, I have the FLVPlayback component. Why do I need code? The answer can be summed up in one word: size. The size of a code-driven SWF is about 1 KB, and its FLVPlayback counterpart weighs in at over 30 KB. The difference is due to the various control components—take a look in your library—that are added into the SWF. The increasing use of video in banner advertising is forcing developers to think small, because the maximum size of a Flash SWF that can be used in a banner ad is often no more than 30 KB. Obviously, the component is simply too “heavy” for use in banner ads. The other reason, which we won't be getting into in this book, is that there is going to come a point in your life when the FLVPlayback component simply isn't going to “cut it” any longer. When you reach this point, you will be creating your own ActionScript-driven controllers, and this will require the use of a video object. The real payback for you will come when you discover you can create your own custom controllers that weigh in under 10 KB.

## Using the FLVPlayback control components

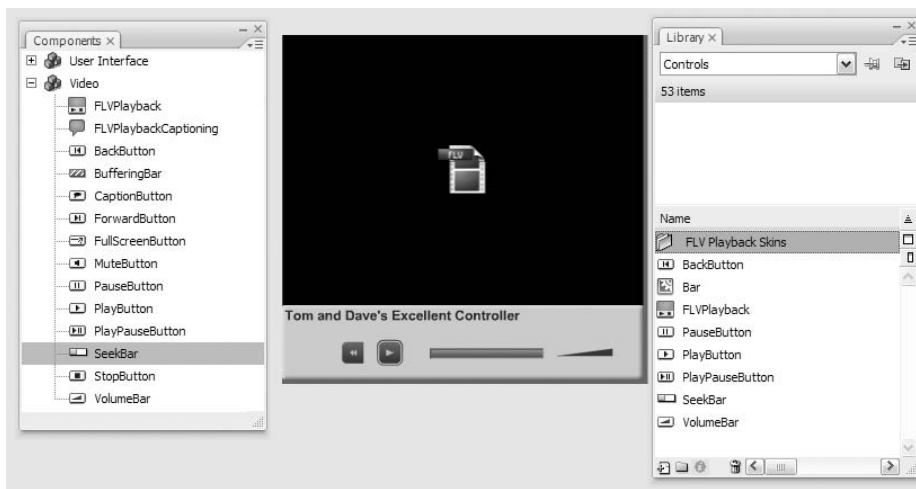
In the Video components area of the Components panel, there are a bunch of individual buttons and bars. They are there for those situations when you look at the skin options available to you and think, “That's overkill. All I want to give the user is a play button and maybe another one to turn off the sound.” This is not as far-fetched as it may seem. There are a lot of websites out there that use custom players that are nothing more than a series of the individual controls. In this exercise you will build a custom video controller using these controls. Let's get started:

1. Open the Control.fla document. When it opens, you will see that the only thing on the stage is a beveled box with a bit of branding on it. If you wish, feel to change the text in the Text layer to your name.
2. Select the Video layer and drag an FLVPlayback component to the stage. Click the Parameters tab in the Property inspector and set skin to none and source to ThroughAdoor.flv.

3. In the Property inspector, set the X and Y locations of the FLVPlayback component to 0.
4. Select the Controls layer and drag the following components to the stage:
 

■ BackButton	■ SeekBar
■ PlayPauseButton	■ VolumeBar
5. Hold down the Shift key and select each of the controls on the stage. Open the Align panel, and, being sure To stage is not selected, click the Center Align button. When you finish, your stage should resemble that shown in Figure 8-22.

If you open the library, you won't see the PlayPauseButton. You will see separate Play and Pause buttons. Don't panic. The PlayPauseButton is actually a combination of both of them.

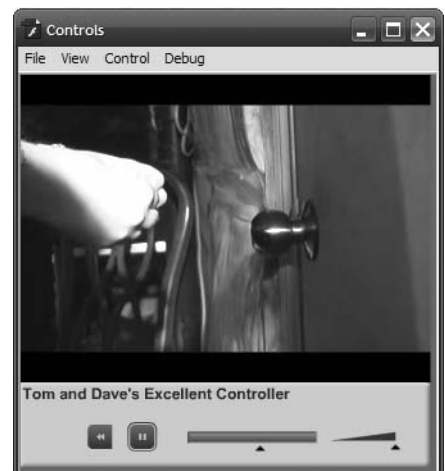


**Figure 8-22.** The video control components, when added to the stage, are also added to the library.

This is the point in this exercise where what you have done is about to shift from “interesting” to “way too cool.” With all of those components on the stage, you are probably preparing yourself, especially if you used them in Flash 8, to start writing a whack of code. Not any more. As long as the components are in the same frame as the FLVPlayback component, they become fully functional. Think about it . . . you have just created a custom video controller in a “code-free zone.” Don't believe us? Check it out yourself:

6. Save and test the movie. Drag the Seek control, shown in Figure 8-23, to the right and left. See . . . we told you.

**Figure 8-23.** A custom video control created in a code-free zone



## Using the FLVPlaybackCaptioning component

A couple of years ago, one of the authors had written a piece about Flash video and how easy it was to get video onto a website. The thrust of the article was that this was a wondrous technology and that video was about to sweep the Web. The reaction to the article was strongly positive, and the author was feeling pretty good about himself—that is, until he received the following e-mail:

*'Love your books and tutorials! They are very well explained. I have a question. Have you done any tutorials on how to add captions to videos? For example, there is a CC button in your "Talking Head" video box. I would love to learn how to write CC for that. I am deaf and would strongly advocate for all websites that have videos to have captions, but that won't happen right away due to \$ and timing. I will be making a small "Talking Head" video introducing myself in sign language, but I want to have captions for hearing people to know what I am saying :-)'*

In our zeal to get video out there, we tend to forget that accessibility is a major factor in our business. As well, accessibility is now the law around the world, and up until Flash CS3, video was somewhat or totally inaccessible to those with hearing impairments.

This isn't to say captions couldn't be added to video in Flash 8. They could, but it required quite a bit of work on the designer's or developer's part to get them to work properly. It usually involved XML, cue points in the FLV, and an understanding of how to use XML in Flash and to write the proper ActionScript to make it all work. Flash CS3 streamlines this process with the inclusion of the FLVPlaybackCaptioning component.

Before we get going, it is important you understand this is not a point-and-click workflow. Entering cue points by hand into the Video Import dialog box in Flash is tedious business. For all but the shortest of video clips, it makes best sense to use a special XML document to make it all work—easier to edit later, too—and then you need to "connect" that document to the FLVPlaybackCaptioning component.

### Timed text XML for captions

The FLVPlaybackCaptioning component allows for the display of captions in the FLVPlayback component through the use of a **Timed Text (TT) XML** document. If you open the captions.xml document you will see, as shown here, the Timed Text XML code used in this exercise:

```
<?xml version="1.0" encoding="UTF-8"?>
<tt xml:lang="en" xmlns=http://www.w3.org/2006/04/ttaf1 ▶
xmlns:tts="http://www.w3.org/2006/04/ttaf1#styling">

  <head>
    <styling>
      <style id="1" tts:textAlign="right"/>
      <style id="2" tts:color="transparent"/>
      <style id="3" style="2" tts:backgroundColor="white"/>
      <style id="4" style="2 3" tts:fontSize="20"/>
    </styling>
  </head>
```

```

<body>
  <div xml:lang="en">

    <p begin="00:00:00.25" dur="00:00:03.25">Dreamweaver users
now have access to Flash Video. Didn't have it before.</p>

    <p begin="00:00:04.20" dur="00:00:03.07">And if you were to
talk to a Dreamweaver user about three or four years ago</p>

    <p begin="00:00:08.03" dur="00:00:01.04">and ask, "You want
to put video on a web page?"</p>

    <p begin="00:00:09.11" dur="00:00:04.00">They would look at
you and go "Yeah.Dude.Yeah.Right.Uh Huh. Next."</p>

  </div>
</body>
</tt>

```

You may notice the format is a bit different from that you may be used to when writing an XML document. This is because Timed Text is a specification used for captioning set by the World Wide Web Consortium, and the XML document prepared for use with the FLVPlaybackCaptioning component must follow that standard.

If you really want to dig into the specification, it can be found at [www.w3.org/AudioVideo/TT/](http://www.w3.org/AudioVideo/TT/).

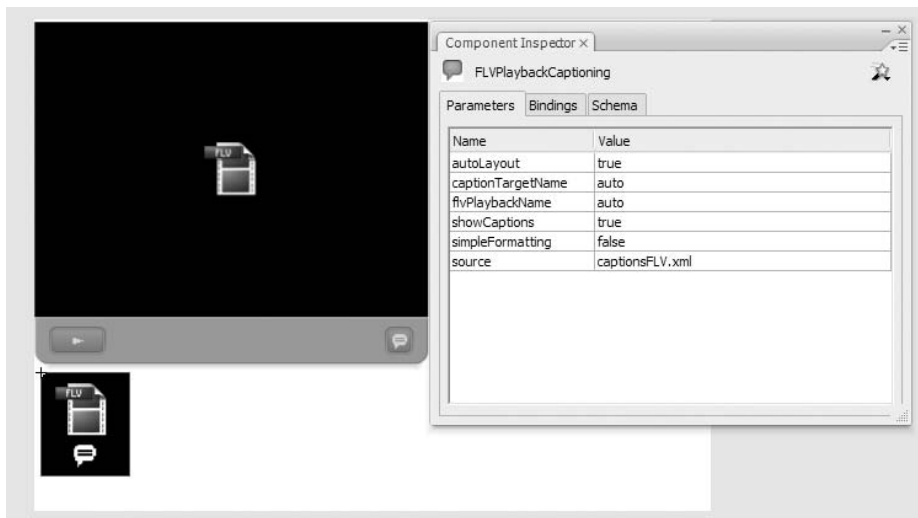
You will notice that you can set the styling for the text, and that each caption needs to have a start and an end point. This means each caption must have a `begin` attribute, which determines when the caption should appear. If the caption does not have a `dur` or end attribute, the caption disappears when the next caption appears or when the FLV file ends. The `begin` attribute means “This is where the caption becomes visible.” The `dur` attribute means “This is how long the caption remains visible.” Alternatively—and this is really a matter of taste—you can omit `dur` and replace it with `end`, which means “This is where the caption stops being visible.”

Where do you get those numbers? You can use the time code in the FLV Encoder to find them, or you can use the time code displayed in the QuickTime or Windows Media Player interfaces. Another place would be in the video editing software used to create the video in the first place.

Follow these steps to apply the captions in the preceding XML example to a video:

1. Open a new Flash document and save it to the CaptioningVideo folder in your Chapter 8 Exercise folder.
2. Drag an FLVPlayback component to the stage and set its source to `Captions.flv` and the skin parameter to `SkinUnderPlayCaption.swf`. Name the layer video.
3. Add a new layer named `Captions`. Drag a copy of the FLVPlaybackCaptioning component to this new layer.

4. Select the FLVPlaybackCaptioning component and click the Parameters tab. As shown in Figure 8-24, the parameters you see are
  - **autoLayout**: A value of true lets the FLVPlayback component determine the size of the captioning area.
  - **captionTargetName**: This parameter identifies the movieclip or text field instance where the captions can be placed. The default is auto, which means the component will make that decision.
  - **flvPlaybackName**: This is the instance name for the FLVPlayback component, which is set in the Property inspector. If there is only one instance of the component, leave the value at the default of auto.
  - **showCaptions**: If set to false, the captions will not display.
  - **simpleFormatting**: If you have no formatting instructions in the XML document, set this to true. Otherwise, leave it at the default value of false.
  - **source**: The location of the Timed Text XML document used to supply the captions.



**Figure 8-24.** The FLVPlaybackCaptioning component and its parameters

5. Click the source parameter and enter captionsFLV.xml as the value for the parameter, and change the showCaptions setting to true.
6. Save and play the video. The captions, as shown in Figure 8-25, will appear.



**Figure 8-25.** The captions will appear over the video.

Be careful with this feature. The example shown assumes the controls will appear under the video. If your controls or skins are to appear over the video, they will hide the captions. To nix this, add a dynamic text field to the stage under the video, give it an instance name, and link the captions to it using the `captionTargetName` parameter for the `FLVPlaybackCaptioning` component. Finally, set the `autoLayout` parameter to `false`; otherwise, Flash puts the new text field right back inside the video.

## Preparing and using alpha channel video

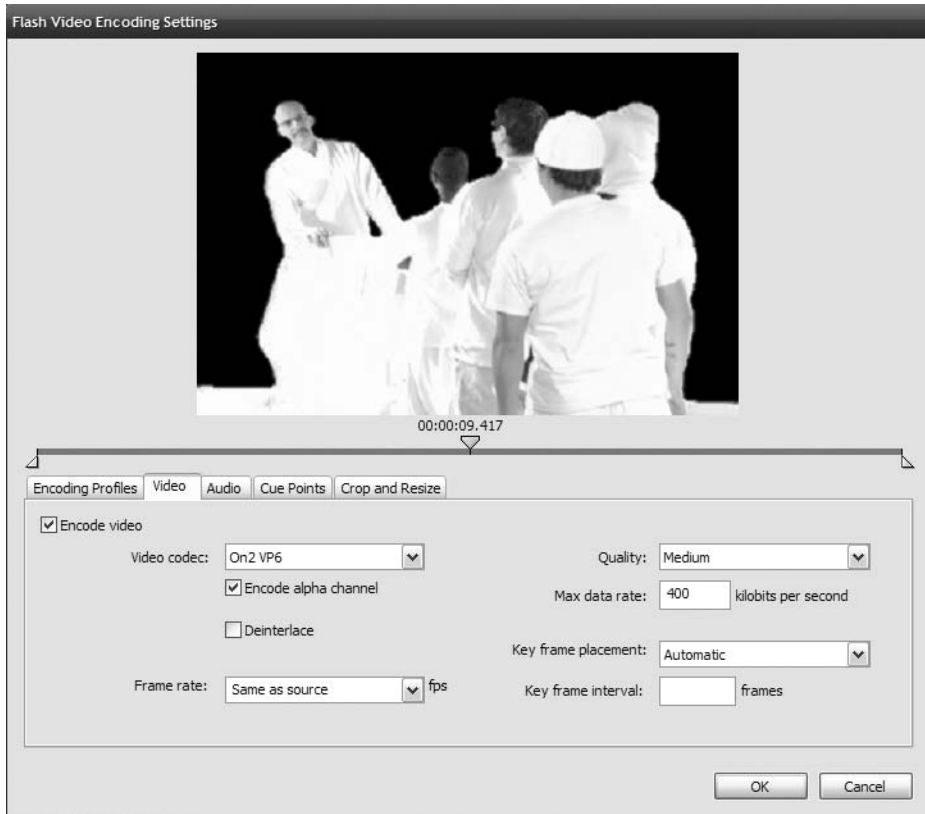
There will be times when you need a talking head video or you want to artificially move the subject of the video from the studio to another location. These are the instances where an alpha channel video fits the bill.

If you watch the weather on your local TV station, you are seeing this in action. The weatherman stands in front of a green wall and starts pointing to fronts and cloud formations. The thing is, the stuff he is pointing at isn't on the wall. The weatherman is pulled out of the green background and superimposed on the radar image or whatever else he is pointing at. The type of video where a green or blue background is removed, or "keyed," is called **alpha channel video**. If you are a Photoshop CS3 user, you are quite familiar with the concept of an alpha channel or masking channel. The only difference between those created in Photoshop CS3 and those created in a video editing application such as After Effects is the channel or mask is in motion.

The ability to use this type of video was introduced in Flash 8 Professional. To use this feature in Flash CS3, you need to use the On2 VP6 codec in the Flash Video Encoder. This means that if your target Flash Player is Flash Player 7 or lower, you can't use alpha channel video.

In this exercise, you are going to encode a small clip of a young adult who has just been informed by his friend that he is dead as the result of being hit by a bus. You are going to encode the video and place it over an image in Flash. Let's get started:

1. Open the Flash Video Encoder and import the Alpha.mov file into the render queue.
2. Click the Settings button and name the file Alpha in the Output filename area. Click the Video tab.
3. Select the On2 VP6 codec from the Video codec drop-down menu and select the Encode alpha channel option shown in Figure 8-26. If you fail to select this check box option, you will lose all transparency in the background.



**Figure 8-26.** Make sure you select the Encode alpha channel option.

How do you know you have been handed a video containing an alpha channel? Open it in the QuickTime player and check the movie information. If the codec used to prepare the video is Animation and the number of colors is Millions +, the channel is there.

4. Reduce the Max data rate setting to 300 kilobits per second and change the frame rate to 15 fps. Click OK to return to the render queue. Click the Start Queue button. When the render process is finished, quit the Flash Video Encoder.



5. Open the AlphaEx.fla file in Flash. When it opens, you will see we have tossed an image of a store into the Background layer.
6. Select the Video layer and drag an FLVPlayback component to the stage. Click the Parameters tab in the Property inspector and set source parameter to your alpha channel video, and set the skin parameter to None. With the component selected, set its X and Y location in the Property inspector to 0.
7. Save and test the movie. The video, as shown in Figure 8-27, appears over the background image.



Figure 8-27. Alpha channel video in action

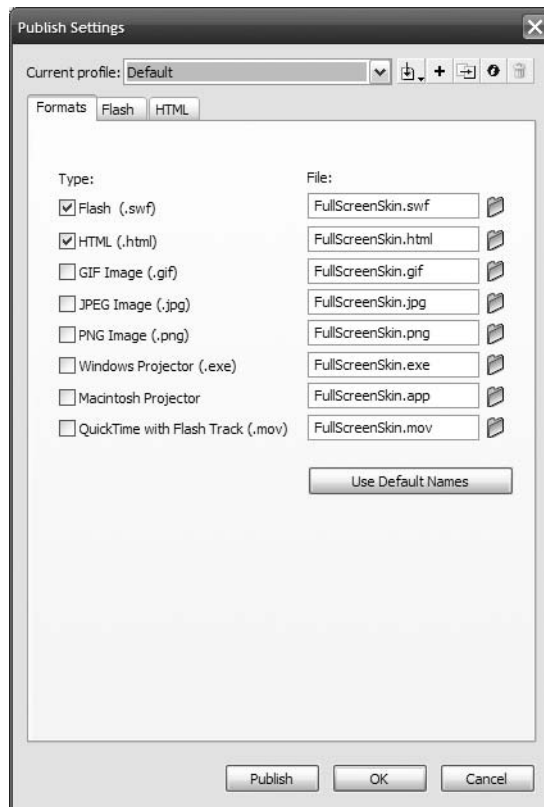
## Going full screen with video

8

In the autumn of 2006, Adobe quietly announced that full-screen Flash video was no longer a dream. They released it as a part of the Adobe Flash Player 9 beta, and even though it was well received, many felt the process was a bit too convoluted. Guess what happened on the way to Flash CS3? Depending on how you wish to approach the application of full-screen video, it can be either dead simple to achieve or require a bit of poking around with ActionScript and in the web page's HTML. In this exercise, you are going to explore both methods. Here's how:

1. Open a new Flash movie and save it to the FullScreenSkin folder in your Chapter 8 Exercise folder.
2. Set the stage size to 400 by 300 pixels and set the stage color to #006633 (dark green).
3. Drag an FLVPlayback component to the stage and specify the following parameters:
  - skin: SkinOverAllNoCaption.swf
  - skinAutoHide: true
  - skinBackgroundColor: #999999 (medium gray)
  - source: DisgruntledDan.flv

4. Save the file as FullScreenSkin.fla.
5. Select File ► Publish Settings to open the Publish Settings dialog box shown in Figure 8-28.



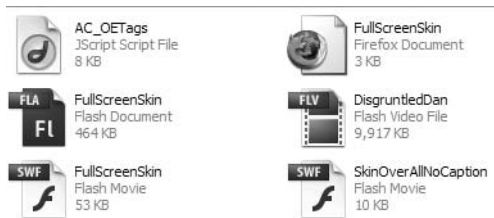
**Figure 8-28.** The Publish Settings dialog box

6. Make sure the Flash and HTML options are selected. Click the Use Default Names button to trim off the path, if there is one, and click the Publish button. When the progress bar finishes and closes, click the OK button to close the dialog box. When you return to Flash, save the file.

You have completed the first part of the process. The skin chosen contains a Full Screen button in the bottom-right corner. The next step is to let the browser know that the video is to be played full screen.

To start, minimize Flash and navigate to the folder where you saved the SWF and the HTML files. When you published the HTML file you actually created two files: the first is the HTML file that contains the SWF, and the second is a file named AC\_OETags.js which, as shown in Figure 8-29, is a JavaScript file. This file is what allows the SWF to play in

Internet Explorer 7 or later without alerting the user to “Click to activate and use this control.” That browser by default blocks active content, such as a SWF file, in a web page, and this JavaScript file does the “unblocking” chores.



**Figure 8-29.** The only file that doesn't get uploaded is the FLA file.

All of these files must be in the same root directory if you plan to upload the project to a web server.

If you are a Dreamweaver CS3 user, you can skip the HTML step in the Publish Settings dialog box. When you place a SWF into a Dreamweaver page, Dreamweaver will handle the active content unblocking chore automatically.

7. Open the HTML file in either your favorite HTML editor such as Dreamweaver CS3 or in a word processing application.
8. Locate the `<script language = "javascript">` tag inside the `<body>` tag (not the `<head>` tag). Click at the end of the line `'salign', ''` and add a comma. The line now looks like this:

```
'salign', ', '
```

Now press Enter (PC) or Return (Mac). Enter the following code:

```
'allowFullScreen', 'true'
```

What you have just done is to make the Full Screen button functional. This tells the browser it really is OK to allow for full-screen playback of the video.

9. Scroll down to the `<noscript>` area of the HTML where the `<object>` and `<embed>` tags can be found. Click at the end of the `<object classid...>` tag and enter the following line (see Figure 8-30):
 

```
<param name="allowFullScreen" value="true" />
```
10. Scroll down to the `<embed src...>` area and click once between `align="middle"` and `allowScriptAccess = "sameDomain"`. Enter the following:
 

```
allowFullScreen="true"
```
11. Save the HTML file and open it in a browser. When the video starts, click the Full Screen button in the bottom-right corner of the controller. The video fills the screen. You can either press the Esc key or click the Full Screen button in the controller, as shown in Figure 8-31, to reduce the video to actual size.

```

34     'movie', 'FullScreenSkin.swf',
35     'align', '',
36     'allowFullScreen', 'true' ← Line 1
37 ); //end AC code
38 }
39 </script>
40 <noscript>
41 <object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" codebase=
    "http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,0,0" width="320" height="270" id="FullScreenToggle"
    align="middle">
42 <param name="allowFullScreen" value="true" /> ← Line 2
43 <param name="allowScriptAccess" value="sameDomain" />
44 <param name="movie" value="FullScreenSkin.swf" />
45 <param name="loop" value="false" />
46 <param name="quality" value="high" />
47 <param name="bgcolor" value="#006633" />
48 <embed src="FullScreenSkin.swf" loop="false" quality="high" bgcolor="#006633" width="320" height="270" name="FullScreenToggle"
    align="middle" allowFullScreen="true" allowScriptAccess="sameDomain" type="application/x-shockwave-flash" pluginspage=
    "http://www.macromedia.com/go/getflashplayer" />
49 </object>

```

**Figure 8-30.** Add a line to the JavaScript parameters and to the object and embed tags in the HTML to get the full-screen playback working.



**Figure 8-31.** Full-screen video is a reality with Flash CS3.

The choice of an OverAll controller is deliberate. This controller becomes visible when the user rolls over the video. If the user clicks the Full Screen button, the video will expand to full screen without the controller interfering in the screen area.

## When video is not video

To this point in the chapter, we have treated video as video content. This is great, but there are going to be occasions where video becomes content and does not require a player, captions, or even full-screen capability. In this case, video can be imported directly into a Flash movieclip and becomes fully accessible to Flash as content on the stage.

Before we start, we want you to be real clear on a fact of “video life:” video files are large, and importing any of the files you have worked with to this point in the chapter directly onto the Flash timeline would be a major error. When considering working with video content on the Flash timeline, think short—loops of about 2 seconds—and think small—the physical size of the video should match precisely the area of the stage where it will be used.

The FLV files used in this exercise were all created in Adobe After Effects 7 Professional. The creation of the videos used is beyond the scope of this book but is covered in some depth in *From After Effects to Flash: Poetry in Motion Graphics* by Tom Green and Tiago Dias (friends of ED, 2006).

Try a couple of exercises to see what we are talking about:

1. Open a new Flash document and change the stage size to 468 pixels wide by 60 pixels high, which is a common banner ad size.
2. Select File ► Import ► Import Video. When the Select Video dialog box opens, navigate to the Apparition.flv file in your Chapter 8 Exercise folder. Click the Next button to open the Deployment window.
3. In the Deployment window, select Embed video in SWF and play in timeline. Though you are going to see a missive on the right side of the dialog box warning you of the evils of this technique, the file isn't that big. Click the Next button to open the Embedding window.
4. In the Embedding window, select Embedded video from the Symbol type drop-down menu. Also be sure the check boxes for Place instance on stage, Expand timeline if needed, and Embed the entire video are selected as shown in Figure 8-32. Click the Next button to open the Finish Video Import window. Click the Finish button to return to the Flash stage.

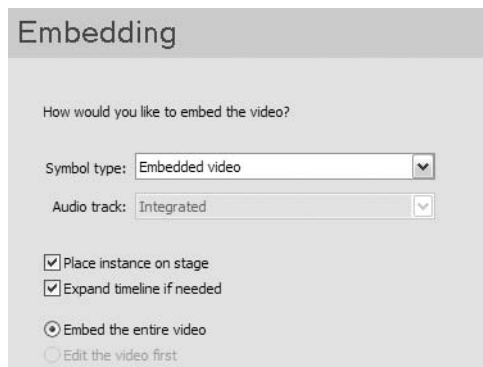


Figure 8-32. Embedding an FLV file in the Flash timeline

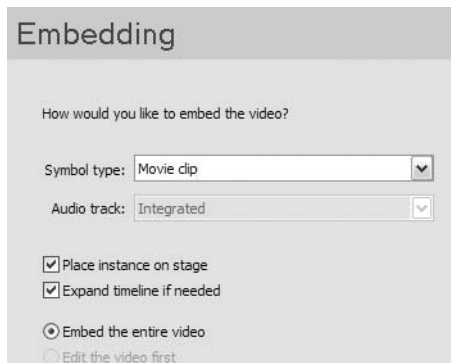
5. You will see a progress bar, and when it finishes, the video will be on the stage, and the timeline will expand to accommodate the number of frames in the video. Select the video, and in the Property inspector, set its X and Y coordinates to 0. If you open the library, you will also see the video is in a video object.
6. Add a new layer to the timeline and enter your name. Save and test the movie. The weird ghostlike apparitions, shown in Figure 8-33, move around behind your name.



**Figure 8-33.** Embedded video can be used as content.

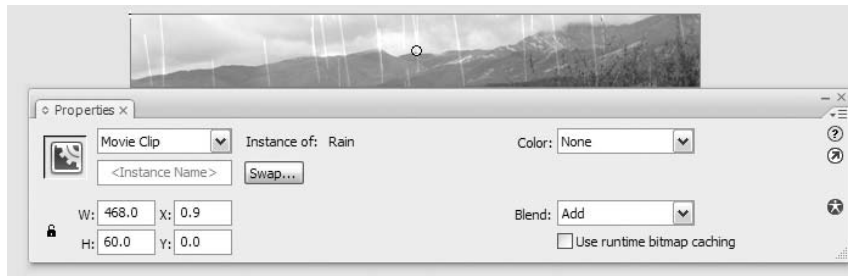
In this next exercise, you are going to create a rainy day in the mountains of Southern California. In this technique, you will discover the power of matching Flash's blend modes with video. Here's how:

1. Open the Rainfall.fla file in your Chapter 8 Exercise folder. When it opens, you will see we have placed an image of the mountains on the stage.
2. Click the first frame of the Video layer. Select File ► Import to stage. When the Import dialog box opens, select the Rain.flv file and click Open.
3. This will launch the Import Video wizard. Embed the video in the timeline, but this time, when you reach the Embedding window shown in Figure 8-34, select Movie clip as the symbol type. This is a good way to go, because it routes all the necessary timeline frames into a movieclip timeline, rather than expanding the main timeline off a mile to the right.



**Figure 8-34.** Embedded video can be turned into a Flash movieclip.

4. Drag the movieclip from the library to the first frame of the Video layer, and using the Property inspector, set its X and Y coordinates to 0. Obviously a big, black movieclip that hides the mountains isn't doing the job. Let's fix that.
5. Select the movieclip on the stage, and in the Property inspector, set the movieclip's blend mode to Add. The rain, as shown in Figure 8-35, becomes visible.

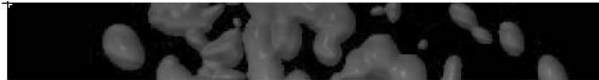


**Figure 8-35.** Use the Add blend mode to remove the black background in the FLV.

**6.** Save and test the movie.

So far you have discovered how video content can interact with Flash content. In this final exercise, you are going in the opposite direction: Flash content interacting with video content.

1. Open the `BlobEffect.fla` file. You will see we have already placed an embedded video on the timeline. The video is a blobs effect. To see it, open the Blobs movieclip in the library, and when the Symbol Editor opens press Enter (PC) or Return (Mac). As you can see in Figure 8-36, green blobs ooze from the top of the window and coalesce into a giant blob, which then splits apart into smaller blobs.



**Figure 8-36.** We start with some green blobs, which is an FLV file embedded into a movieclip.

2. Click in the Text layer, select the Text tool (or press T), and enter your name. Use a font and size of your choosing, but change the color of the text in the Property inspector to #FFFF00 (bright yellow).
3. With the text selected, convert the text to a movieclip symbol named Name.
4. With the Name movieclip symbol selected, select Overlay from the Blend drop-down menu. The text will disappear. This is because the overlay mode either multiplies or screens the colors, depending on the destination color, which is the color immediately under the text. In this case, the yellow text is against a black background, meaning you can't see the effect.
5. Save and play the movie. Notice how the text, as shown in Figure 8-37, changes and becomes visible as the blobs pass under it.



**Figure 8-37.** A classic example of Flash content interacting with video content

## Your turn: XML captions for video

In the exercise in the section “Timed text XML for captions,” you used the Timed Text XML standard for adding captions to a video. In this exercise, you will be adding captions using a completely different “flavor” of XML and method of getting the captions and their respective cue points into the FLV file.

There are four ways of adding cue points to an FLV file:

- Add them when you create the FLV file in the Flash Video Encoder.
- Add them using the `FLVPlayback` component’s parameters.
- Add them using the `addASCuePoint()` method in ActionScript.
- Add them using an XML document.

The first two methods are what we call “destructive.” Once you add a cue point using those two methods, it can’t be removed. This means if your timing is off, the video will have to be reencoded and new cue points added.

Here’s some self-defense if you go this route. Don’t remove the video from the render queue until the video is approved for play. In this circumstance, and it only works for cue points added in the Flash Video Encoder, you select the video in the render queue and select `Reset Status` in the `Edit` menu. When you return to the `Cue Points` tab, they will all be there and can be removed and changed.

The last two ways are the most flexible because, if the timing is off, you simply open the code and change a number.

This exercise concentrates on using an XML document to insert the cue points. Before we dig into the XML, it is important you understand that in Flash video, there are two flavors of cue points. The first type of cue point is called a navigation cue point. Navigation cue points do exactly what the name implies: they are used to navigate, or seek, to keyframes in the video itself. If you create a navigation cue point, Flash will actually insert a keyframe at that point in the video. Event cue points are the most common. They tell Flash and/or ActionScript to do something when they are encountered. This is why the cue points you will create are event cue points. They will be used to tell Flash to display a caption.

Though we think Timed Text XML is the way to go when using XML to insert captions, you may just decide to use “plain old” XML to do it. If you do, there is a very specific format you must follow. Let’s look at it:

1. Open the `CuePoints.xml` document in your `YourTurn` folder. You can use Dreamweaver CS3 or even a word processor for this purpose. When the document opens, the first “chunk” of code you will encounter is the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<FLVCoreCuePoints>
  <CuePoint>
    <Time>9000</Time>
    <Type>event</Type>
    <Name>fl.video.caption.2.0.0</Name>
```



```

<Parameters>
  <Parameter>
    <Name>text</Name>
    <Value><![CDATA[<font face="Arial, Helvetica, _sans" ↵
size="12">Look ... up in the sky ... look...</font>]]></Value>
  </Parameter>
  <Parameter>
    <Name>endTime</Name>
    <Value>11.0</Value>
  </Parameter>
</Parameters>
</CuePoint>
</FLVCoreCuePoints>

```

This is the syntax that must be used. Deviate from it at your own peril. The first line declares the doctype, and the second line tells Flash that anything between the `FLVCoreCuePoints` tags is to be used within a cue point.

Each cue point you will add must be enclosed between `<CuePoint>` and `</CuePoint>` tags. The `<Time>` tag is the start of the cue point, and this number must be expressed in milliseconds. The next tag, `<Type>`, tells Flash that the cue point is to be an event cue point, and the tag following it, `<Name>`, is the name of the cue point.

The rules regarding naming are rigid. The `<Name>` tag must be `fl.video.caption.2.0` followed by a series of sequential numbers to guarantee uniqueness. In our sample XML, it goes `fl.video.caption.2.0.0`, `fl.video.caption.2.0.1`, and so on.

8

The parameters contain the styling data for the text that will appear in the caption and an end time for the caption. Notice how we used the `<i>` tag to identify who is speaking by setting the person's name in italics. HTML tags may be used only if they're supported by Flash; a list of these may be found in the "HTML formatting" section of Chapter 6. The `endTime` property, which must be expressed in seconds, will be the time when the caption disappears from the screen. This number can either be an integer (no decimals) or can contain up to three decimal places.

Finally, you may optionally contend with using color in captions, and there are a couple of rules involving this as well. If you scroll down to `caption 2.0.7`, you will see the text in the caption uses `#FF0000`, which is a bright red. A couple of lines later the `backgroundColor` parameter changes the background color of the caption to `0x01016D`, which is a dark blue.

The key here is how the colors are identified. Colors are specified by hexadecimal values, but the *indication* that the color is in hex—`#` or `0x`—depends on where it's being stated. The first change to the red uses the pound sign, `#`, as traditionally used in HTML. Why? Because it appears within HTML-formatted content. The second change—to the dark blue—uses the format for specifying hexadecimal notation in ActionScript, `0x`. If you do change the background color of a caption, that color will "stick." This means all subsequent captions will use this background color. If you only need a single change, like our example, change the `backgroundColor` parameter back in the next cue point. In our case, we changed it to black again (`0x000000`), as seen in `caption 2.0.8`.

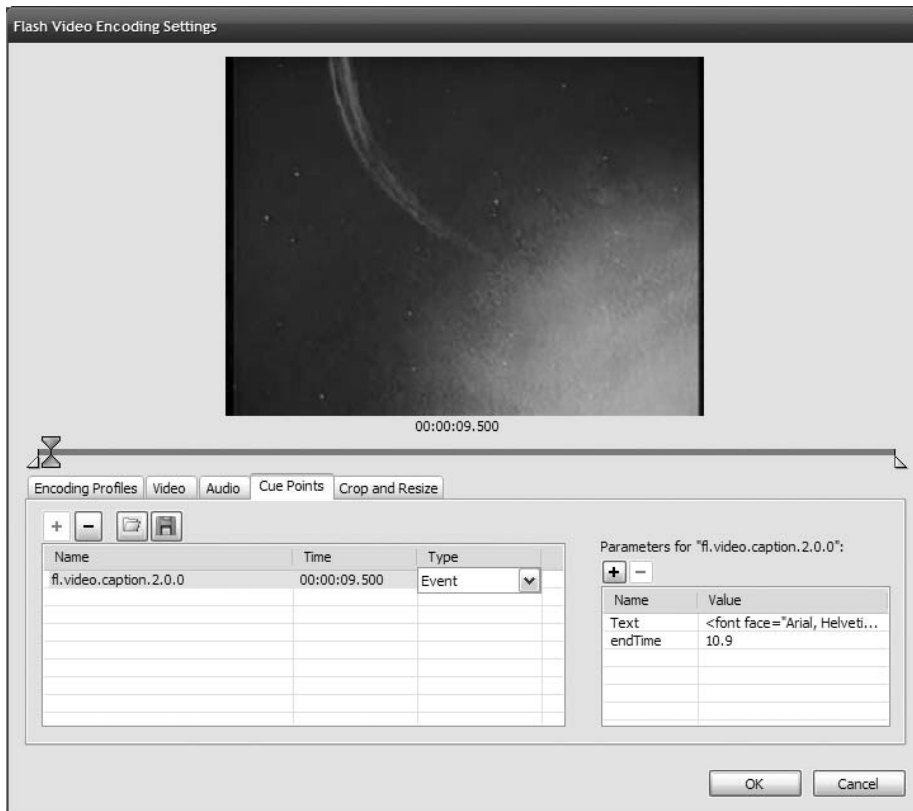
Do your sanity a favor and separate each caption with an empty line or two in the XML. This makes them easier to read and locate. The space, called **whitespace**, will be ignored by Flash.

So what does all of this have to do with cue points and FLV files? You are about to find out. First, though, you need to download a cartoon.

In the 1940s, the original Superman cartoons were produced by a gentleman named Max Fleischer. Though we aren't going to get into the details, a small number of these cartoons have entered the public domain—that means they are free for you to download and use. One of them, “Superman: the Mechanical Monsters,” is the cartoon you will be captioning. In order to remain purer than pure, we aren't including the cartoon in the Exercise downloads. We would respectfully ask that you head over to [www.archive.org/details/superman\\_the\\_mechanical\\_monsters](http://www.archive.org/details/superman_the_mechanical_monsters). The file on the left side of the page can be downloaded. In theory, it doesn't matter which file you download—many exist, at different compressions and file sizes—but we used the 256 KB MPEG4 (27MB) version.

As an aside, we find it rather fascinating that the copy of the video that plays on the page is Flash Video. A low quality one . . . but Flash Video all the same.

2. Now that you have downloaded the movie, open the Flash Video Encoder and drag the video from its location into the render queue.
3. In the Encoding Profiles window, enter Superman as the Output filename. Click the Video tab.
4. When the Video settings open, ensure you are using the On2 VP6 codec, change the Max data rate value to 275 and specify a frame rate of 15 fps. Click the Audio tab.
5. When the Audio settings open, change the Data rate setting to 64 kbps(mono). Click the Cue Points tab.
6. As shown in Figure 8-38, this is where all of the pain, sweat, and aggravation that went into creating the XML document comes into play. The care and diligence you put into ensuring all of the tags in the XML document are correct are about to pay off. How so? Manually add the first cue point to give you a taste of manually adding cue points. Scrub the playback head of the FLV to the 00:00:09.500 mark of the video.
7. Click the + sign, which is the Add Cue Point button. Enter `fl.video.caption.2.0.0` as the name of the cue point. Notice how the default value for Type is Event.

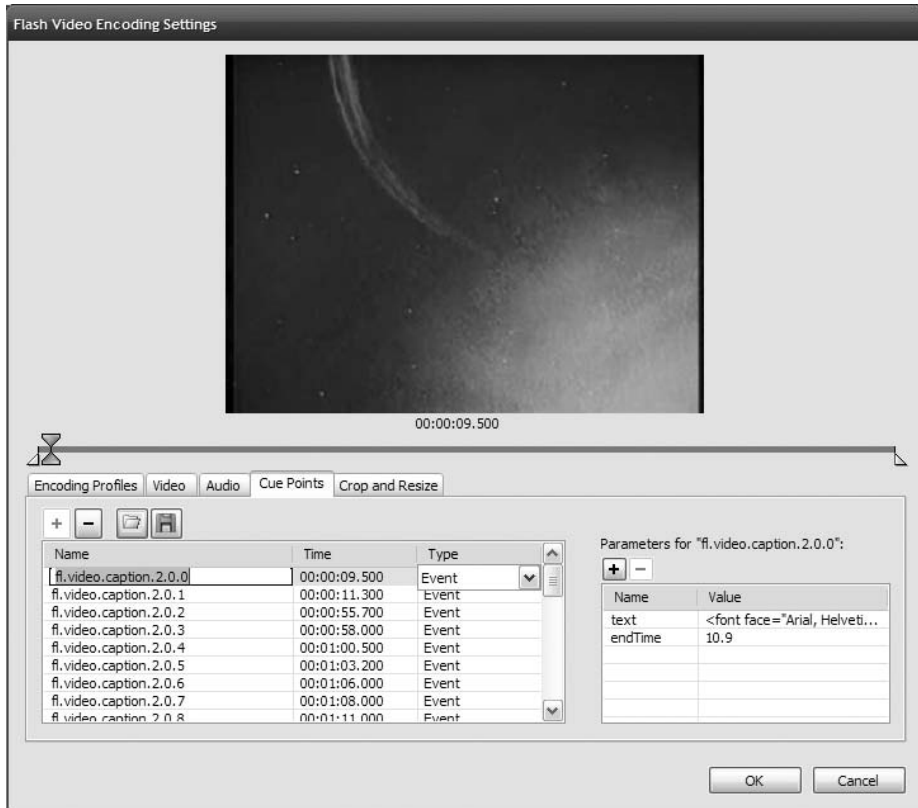


**Figure 8-38.** Manually adding cue points to an FLV

8. Click the Add Parameter button and enter Text into the name area. Click in the Value area and enter `<font face="Arial, Helvetica, _sans" size="12">Up in the sky, look!</font>`.
  9. Click the Add Parameter button and enter endTime as the name and 10.9 as the value.
- Now repeat steps 7 and 8 about 30 more times to add the remaining cue points. (Yeah, we are kidding.)

Obviously, going the manual route is tedious at best. Surely there must be an easier method. There is: embed the `CuePoints.xml` document right into the FLV file. If you have used Flash 8, this method might seem a bit unfamiliar. It is. The ability to embed an XML document into an FLV file is new to Flash CS3. Here's how:

1. Select the cue point and click the Remove Cue Point button (the – sign) to remove the cue point just added.
2. Click the Navigate button—it looks like a file folder—in the Cue Points window. This will open the Load Cue Points File dialog box. Navigate to the YourTurn folder, select the CuePoints.xml file, and click the Open button.
3. When you return to the Cue Points window, you will notice all of the cue points in the XML document have been added. If you select the first one, as shown in Figure 8-39, you will also see that the parameters have also been added.



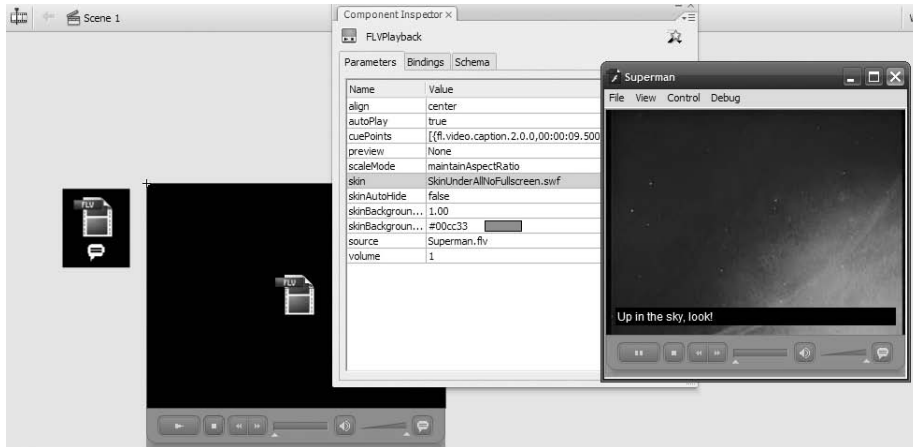
**Figure 8-39.** Load the XML, and the cue points are added in less than one second.

4. Click the OK button to return to the render queue, and click the Start Queue button to encode the cartoon.
5. Open Flash CS3 and create a new document. Save this document to the YourTurn folder.
6. Drag an FLVPlayback component to the stage, add a skin (we used SkinUnderAllNoFullScreen.swf), and set the source to the FLV file just created.

7. Drag a copy of the FLVPlaybackCaptioning component onto the pasteboard. This component only needs to be in the SWF (not necessarily the stage) for it to work. If you put it on the pasteboard, it won't be mistaken as a piece of content.

You will notice you don't have to add the CuePoints.xml document as a parameter in the FLVPlaybackCaptioning component. You only need to do this when using Timed Text captions.

8. Save and test the movie. Notice how the captions automatically appear (see Figure 8-40).



**Figure 8-40.** The FLVPlaybackCaptioning component only need to be in the SWF. . . not on the stage.

If you think this exercise is nothing more than mildly interesting, you would be making a profound error in judgment. One of the reasons Flash video rarely appears on government or other publicly funded/subsidized websites is because video was, for all intents and purposes, inaccessible. The ability to easily add captioned video and to turn the captions on and off has opened up a market that was otherwise closed to Flash designers and developers.

## Playing with alpha channel video

In this final exercise in this chapter, we introduce you to a couple of new concepts. The first is that video doesn't necessarily have to use the FLVPlayback component and reside on the main timeline for it to work. The second concept is that just because it is video is no reason for not having fun with it. Let's start jamming with video:

1. Open the VideoJam.fla file in the Chapter 8 Exercise folder. You will notice we have provided the background image.
2. Create a new movieclip symbol and name it Video.
3. In the Symbol Editor, open the library and select New Video from the library dropdown menu. Just click OK when the Video Properties dialog box opens.
4. Drag the video object from the library onto the stage, and in the Property inspector, give it the instance name of myVideo, set its X and Y position to 0, and change its width and height values to 320 and 214.

5. Add a new layer to the movieclip and name it Actions. Select the first frame of the Actions layer, open the ActionScript Editor, and enter the following code:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
var ns:NetStream = new NetStream(nc);
myVideo.attachNetStream(ns);

var listener:Object = new Object();
listener.onMetaData = function(md:Object):void {};
ns.client = listener;

ns.play("Alpha.flv");
```

6. Return to the main timeline, select the Video layer, and drag your new movieclip symbol to the stage. Save and test the movie.

What you have just discovered is video can be put into a movieclip and will still play on the main timeline. This is an important concept for two reasons:

- The resulting SWF is under 30 KB, meaning you can use it in banner ads. In fact, if you want it to be even smaller, remove the image, and the file size drops to 1 KB.
- Objects contained in movieclips are open to creative manipulation

Let's check that last point out:

1. Select the movieclip on the stage and click the Filters tab. Click the + sign to open the Filters drop-down menu and select Drop Shadow.
2. In the Drop Shadow filter options of the Filters panel, apply these values:
  - BlurX: 15
  - BlurY: 15
  - Strength: 75%
  - Quality: High
  - Distance: 10
3. Test the movie. The people in the video, as shown in Figure 8-41, have all developed shadows. This is because the video, like a box drawn in a Flash file, a Fireworks CS3 PNG, or a Photoshop CS3 image, contains an alpha channel. In the case of video, this channel moves, and Flash applies the drop shadow to the channel.
4. This looks OK, but how about we give the subjects a bit of depth? Select the movieclip on the stage, click the Filters tab, and add a Bevel filter to the video.

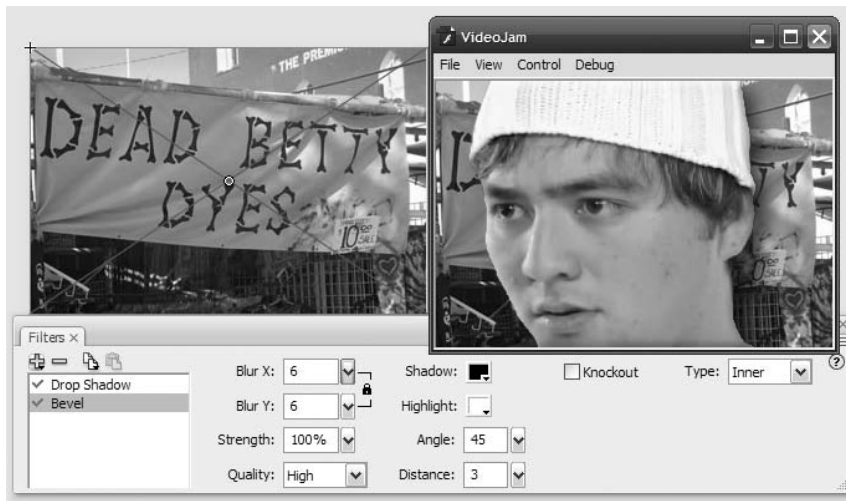


**Figure 8-41.** Filters can be applied to video contained in a movieclip.

5. In the Bevel filter options of the Filters panel, specify these values:

- BlurX: 6
- BlurY: 6
- Quality: High
- Distance: 3

6. Save and test the movie. The subjects, shown in Figure 8-42, take on a bit of depth, and you have also added a hint of backlighting. Don't get aggressive with filters; subtlety counts.



**Figure 8-42.** Multiple filters can be applied to video.

Hang on, these guys are ghosts. Can you turn them into ghosts? You bet.

1. In the Filters panel, select the Drop Shadow filter and select Knockout, Inner Shadow and Hide Object.
2. Test the movie. You have a 3D ghost.

Interesting, but can you do better. Of course.

3. In the Filters panel, select the Drop Shadow filter and deselect Knockout, Inner Shadow and Hide Object.
4. Click the Properties tab in the Property inspector.
5. Select the video on the stage and select Overlay from the Blend drop-down menu. Test the video. The subjects take on a “ghost-like” appearance, as shown in Figure 8-43.



**Figure 8-43.** Don't be afraid to use the blend modes to create some interesting effects.

## What you've learned

- How video can be streamed from your web server
- How to use the Flash Video Encoder
- How to encode video containing an alpha channel
- Several methods of embedding and streaming video without the use of the FLVPlayback component



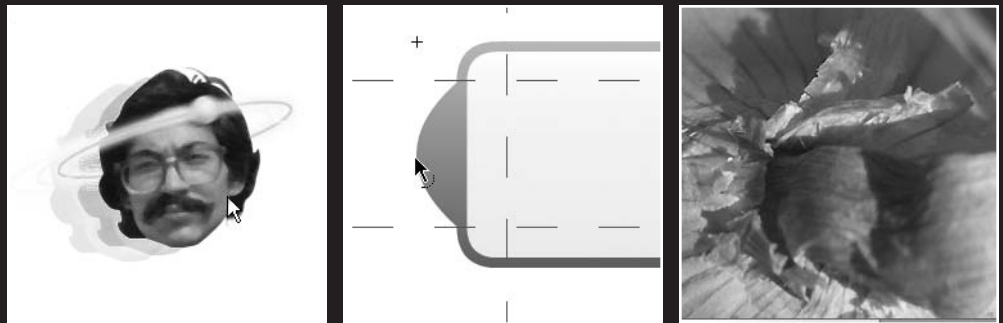
- How to add Timed Text captions to a video and how to use the `FLVPlaybackCaptioning` component
- A method of creating captioned video through the technique of embedding a video into the FLV file
- The power of the creative use of filters and blend effects that can be applied to video

This has been quite the chapter, and we suspect you are just as excited about the possibilities of Flash video as we are. The key to the use of Flash video is really quite simple: keep an eye on the pipe. The Flash Video Encoder is one of the most powerful tools in the Flash Video arsenal, and mastering it is the key to Flash video success. From there, as we showed you in several exercises, the only limit to what you can do with Flash video is the one you put on your creativity.

As you started working with the Flash video components, we just know you were wondering, “How do those UI components work?” Great question, and we answer it in the next chapter.



## 9 USING THE FLASH UI COMPONENTS TO BUILD INTERFACES



Since early in its life, Flash has proven itself the leader in web animation. In recent years, that dominance has nudged into the realm of online applications as well. For user-facing applications, you need user interface (UI) elements, plain and simple—something to receive input from the person viewing your content or to display information in a specific way, such as in a grid or selection box. Sure, you’ve already seen how button symbols work, and you’re aware that input text fields accept hand-typed content. Those make a good start, but they’re also nothing more than the tip of the iceberg.

The UI components that ship with Flash CS3 are an improvement over the Flash 8 set in a number of ways: size (much smaller), performance (faster, better) and ease of customization.

*As a bonus, Flash CS3 even gives you the previous set, known as the v2 components, but those only work with ActionScript 2.0. That’s an important point! They’re for publishing older movies if you have to. Choosing the Flash document type or changing your publish settings between ActionScript 3.0 and 2.0 automatically updates the Components panel to offer the correct set. You cannot mix and match components designed for different versions of ActionScript. In ActionScript 1, you lose the UI components altogether.*

What we’ll cover in this chapter:

- Using the Flash CS3 UI components
- Using ActionScript 3.0 to control components
- Changing component skins

Files used in this chapter:

- Button01.fla (Chapter02/ExerciseFiles\_CH09/Exercise/Button01.fla)
- Button02.fla (Chapter02/ExerciseFiles\_CH09/Exercise/Button02.fla)
- Button03.fla (Chapter02/ExerciseFiles\_CH09/Exercise/Button03.fla)
- Button04.fla (Chapter02/ExerciseFiles\_CH09/Exercise/Button04.fla)
- CheckBox.fla (Chapter02/ExerciseFiles\_CH09/Exercise/CheckBox.fla)
- ColorPicker.fla (Chapter02/ExerciseFiles\_CH09/Exercise/ColorPicker.fla)
- ComboBox.fla (Chapter02/ExerciseFiles\_CH09/Exercise/ComboBox.fla)
- DataGrid.fla (Chapter02/ExerciseFiles\_CH09/Exercise/DataGrid.fla)
- Label.fla (Chapter02/ExerciseFiles\_CH09/Exercise/Label.fla)
- List.fla (Chapter02/ExerciseFiles\_CH09/Exercise/List.fla)
- Mug01.jpg (Chapter02/ExerciseFiles\_CH09/Exercise/Mug01.jpg)
- Mug02.jpg (Chapter02/ExerciseFiles\_CH09/Exercise/Mug02.jpg)
- Mug03.jpg (Chapter02/ExerciseFiles\_CH09/Exercise/Mug03.jpg)
- Mug04.jpg (Chapter02/ExerciseFiles\_CH09/Exercise/Mug04.jpg)
- Mug05.jpg (Chapter02/ExerciseFiles\_CH09/Exercise/Mug05.jpg)

- Mug06.jpg (Chapter02/ExerciseFiles\_CH09/Exercise/Mug06.jpg)
- Mug07.jpg (Chapter02/ExerciseFiles\_CH09/Exercise/Mug07.jpg)
- Mug08.jpg (Chapter02/ExerciseFiles\_CH09/Exercise/Mug08.jpg)
- NumericStepper.fla (Chapter02/ExerciseFiles\_CH09/Exercise/NumericStepper.fla)
- Onion.jpg (Chapter02/ExerciseFiles\_CH09/Exercise/Onion.jpg)
- ProgressBar.fla (Chapter02/ExerciseFiles\_CH09/Exercise/ProgressBar.fla)
- RadioButton.fla (Chapter02/ExerciseFiles\_CH09/Exercise/RadioButton.fla)
- ScrollPane.fla (Chapter02/ExerciseFiles\_CH09/Exercise/ScrollPane.fla)
- Slider.fla (Chapter02/ExerciseFiles\_CH09/Exercise/Slider.fla)
- TextArea.fla (Chapter02/ExerciseFiles\_CH09/Exercise/TextArea.fla)
- TextInput.fla (Chapter02/ExerciseFiles\_CH09/Exercise/TextInput.fla)
- TileList.fla (Chapter02/ExerciseFiles\_CH09/Exercise/TileList.fla)
- UILoader.fla (Chapter02/ExerciseFiles\_CH09/Exercise/UILoader.fla)

Anyone familiar with HTML development knows how easy it is to add a check box, radio button, or other form element into a document. These are usually used in “contact us” pages, online surveys, and other application scenarios. Flash components provide you the same set of “widgets,” but you also get a whole lot more, including components not possible in a browser alone. A smidgen of ActionScript is required to wire them together, but for the most part, these are drag-and-drop convenient. In any case, this chapter will help you make sense of it all.

Out of the box, the Flash UI components are styled in a modest, attractive manner that comfortably fits a broad range of designs. Of course, Flash being what it is—free from the relative constraints of HTML—you may want to customize their appearance, and you can. Designers and developers familiar with Flash 8 might warn you with a shudder that you’re in for a barrel of headaches. Tell the old-timers they can breathe easy. Things have improved considerably in Flash CS3.

We’ll start our exploration with the Button component and spend a bit more time with it than the others, simply because once you “get it,” you get it. To be sure, certain components are more complex than others, and we certainly won’t skimp as we visit each one—but if you’re a complete newcomer, you may want to read through the “Button component” section first, and then breeze the other headings until you find components of interest to you.

## Button component

At first glance, the Button component is just another button symbol, but the two shouldn’t be confused. As discussed in Chapter 3, button symbols have a specialized timeline, made of Up, Over, Down, and Hit frames. As such, button symbols are very malleable: Over artwork can be made to spill over the button’s Up shape, paving the way for quick-and-dirty tooltips and other tricks. Hit artwork can make the button invisible—but still clickable—if

it is the only frame with content. In contrast, the Button component has no discernible timeline. It's a self-contained component (as shown in Figure 9-1) and is much more conservative (at first glance) than its wild, partying cousin.

*Using one or more instances of the Button component in your movie will add 15 KB to the SWF if no other components share the load.*

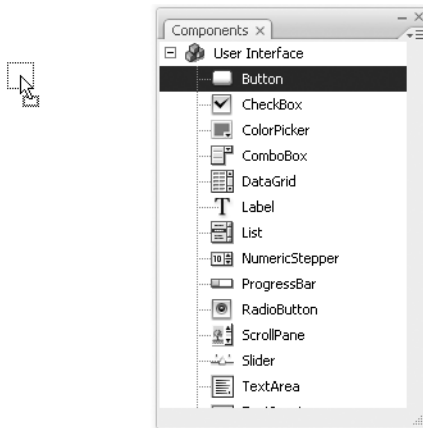
Poke My Belly

**Figure 9-1.** The Button component. Pretty conservative, even without the tie.

## Using the Button component

What makes the Button component so special? In two words, *consistency* and *toggleability*. The first of those, consistency, will be evident in each of the components we visit. If you accept the default skin for every component, you'll get a reliable uniformity among your UI widgets. The second word—well, we admit it, *toggleability* isn't a word—but what it means is that you get a button that optionally stays pressed after you click it, and releases again when you click it a second time. This useful feature is possible without a lick of ActionScript knowledge. Let's see how:

1. Start a new Flash document and open the Components panel (Window ► Components). In the Components panel, open the User Interface branch by clicking the + button or double-clicking the words User Interface. When this branch is open, the + button becomes a -, and you'll see the list of available UI components. Drag an instance of the Button component to the stage, as shown in Figure 9-2.

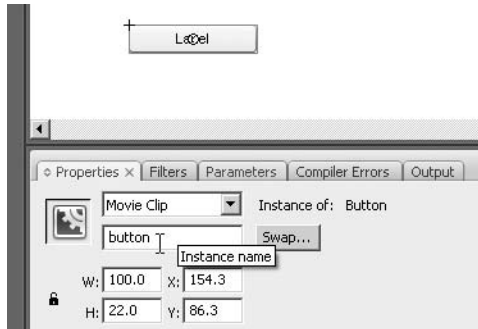


**Figure 9-2.** Adding a UI component to the stage is as easy as dragging and dropping.

Doing this drops a copy of the Button component and a folder named Component Assets into your library. You can ignore the Component Assets folder for the time being. Any time you want additional Button instances, drag them from your library.

2. The first thing to do is give your button an instance name. Select the button by clicking it once, and then type `button` into the Instance Name field of the Property inspector, as shown in Figure 9-3.

*It is also possible to provide the instance name via the Parameters tab.*



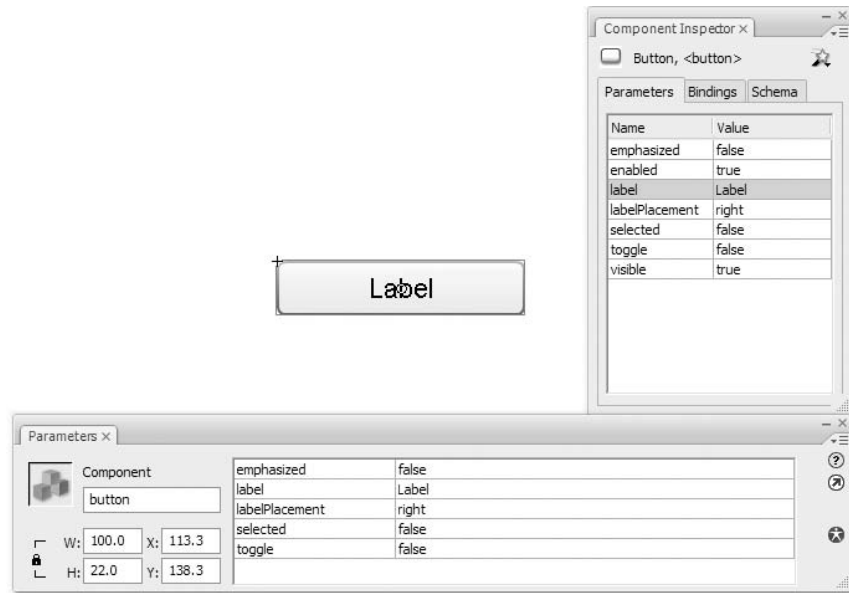
**Figure 9-3.** Always give your component instances an instance name.

Under normal circumstances, you should make your instance name something more meaningful than the generic *button*, but for now, this will do. If you like, use the Free Transform tool to change the dimensions of the button. Note that it resizes much like any symbol, but its text label stays the same size.

*Skewing or rotating the button makes its label disappear because font outlines are not embedded on their own.*

3. By default, the button's label is the self-descriptive term *Label*. Let's change that. Click the Parameters tab (shown in Figure 9-4), and double-click the right column in the label row. Change the word *Label* to *Activate*. When this button becomes a toggle, you'll make it actually activate something. For now, leave the `toggle` parameter at its default setting of `false`.

*Another useful tool for changing a component's parameters is the aptly named Component Inspector panel, found in the Window menu. Open this and all of a component's parameters are available in one screen, meaning you don't have to scroll tediously through parameters in the Property inspector. Which method is best? Whatever works for you.*



**Figure 9-4.** The label parameter determines the text used in the button.

4. Rename your button's layer from *Layer 1* to *button*, and create a new layer. Name the new layer *scripts* and click inside frame 1 of the *scripts* layer. Open the *Actions* panel and enter the following *ActionScript*:

```
button.addEventListener(
    MouseEvent.CLICK,
    function(evt:MouseEvent):void {
        trace("By George, I've been clicked!");
    }
);
```

5. Test your movie (Control ► Test Movie) to verify that a button click sends the message "By George, I've been clicked!" to the *Output* panel.

*For an explanation of how this ActionScript works, be sure to read Chapter 4.*

6. To make this button a *toggle*, return to the *Parameters* tab and change the *toggle* parameter to *true*. Test the movie again, if you like, to confirm that the button now stays in when you click it and pops out again when you click it a second time. Compare your work with *Button01.fla* in the *Exercise* folder for this chapter.



To actually make use of this toggled/untoggled state, you will need to use the `BaseButton.selected` property of the `Button` component instance on the stage. Many button-like components, including `Button`, `CheckBox`, and `RadioButton`, inherit from the `BaseButton` class family tree, which means that they support a `selected` property like their ancestor does. The button's instance name lets you access this property easily.

1. Open the `Button02.fla` file that accompanies this chapter. This file picks up where we left off in the previous exercise. The only difference is a movieclip containing a PNG image has been added to the library. You're going to make this movieclip draggable, but only when the button is pressed. Create a new layer and name it `mystical dude`. Select the new layer and drag an instance of the movieclip `dude` to the stage. Give this movieclip the instance name `dude`.
2. In the `scripts` layer, select frame 1 and add the following new ActionScript beneath the existing code:

```
dude.addEventListener(
    MouseEvent.CLICK,
    function(evt:MouseEvent):void {
        if (button.selected == true) {
            dude.startDrag();
        }
    }
);
dude.addEventListener(
    MouseEvent.CLICK,
    function(evt:MouseEvent):void {
        dude.stopDrag();
    }
);
```

The key here is the `if` statement in the `MouseEvent.CLICK` handler. The `if` evaluates the button's `selected` property as described previously. When it's set to `true`, dragging commences, as shown in Figure 9-5; otherwise, dragging is ignored.



**Figure 9-5.** Checking the button's `selected` property allows you to perform actions only when the button is clicked.

*To see the full list of events available to the Button component, look up the Button class in the [ActionScript 3.0 Language and Components Reference](#). Don't forget to select the [Show Inherited Styles](#) hyperlink beneath the Events heading!*

One final note before we start playing with the looks of this component. Unlike normal library assets, UI components add to the weight of your movie whether or not they're used in the timeline. This is why seasoned Flash developers regard these things in much the same way Dracula regards garlic. The reason for this is that components are set to export for ActionScript. Right-click (PC) or Ctrl-click (Mac) any component in your library and choose **Linkage** to see for yourself.

The first UI component in your movie usually adds the most weight, proportionately speaking, to the SWF. Some components weigh more than others, but all of them rely on a base framework that provides functionality for the whole set. For this reason, your first instance of Button will add 15 KB. The second and third instances won't add anything. Your first CheckBox instance, on its own, will add 15 KB, and additional CheckBox instances will add nothing. However, if you *already have* a Button instance in the movie and *then* add a CheckBox, the combined total of both components is only 16 KB.

*To remove the weight of these components, in case you change your mind and decide to omit them from your design, delete the component(s) and Component Assets folder from the library.*

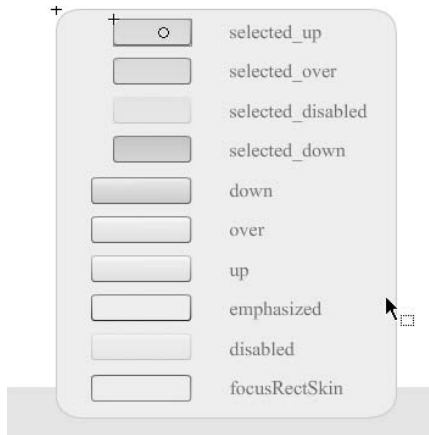
## Changing the Button component's appearance

What you're about to see can be achieved with most of the UI components, not just Button. (Some components have few or no visual elements, so there are exceptions.) This is good news, because it means you'll get the basic gist right off the bat. There are two ways to alter a UI component's appearance: skinning and styling. The first generally deals with the material *substance* of the component—the shape of the clickable surface of a button, the drag handle of a scrollbar—and the second generally deals with text, dressing, and padding.

### Skinning

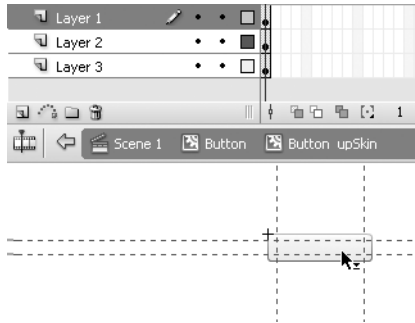
Before Flash CS3, the practice of skinning UI components was an exercise in alchemy. Only the wisest and purest of wizards would trust themselves to toss mysterious ingredients into the frothing cauldron. All of that has changed. In fact, it couldn't get much easier.

1. Create a new Flash document and drag an instance of the Button component to the stage. Double-click the button and you'll see a convenient "buffet table" of the various visual states available to the button, as shown in Figure 9-6.



**Figure 9-6.** Skinning UI components is just way easy.

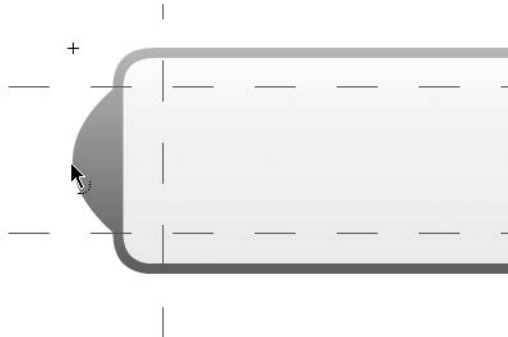
2. The up skin is the button's default appearance. Double-click that and you'll come to the symbol that represents the up skin for this component, complete with 9-slice scaling (as shown in Figure 9-7).



**Figure 9-7.** A mere two levels in, and you're ready to change the appearance of the button.

3. This skin happens to be made of three layers, but it really doesn't matter. Other skins may be different, in this component or another. Select an area in one of these layers and change the button's appearance, perhaps like Figure 9-8—but the choice is yours.

Make sure that the existing shapes, or any new ones, align to the upper left (0, 0) of the symbol's registration point. Adjust the 9-slice guides as necessary. See Button03.fla for an example with minor changes to the up and over skins.



**Figure 9-8.** Adjust the existing shapes or create new ones.

4. Select **Edit > Edit Document** to return to the main timeline. What the . . . ? In the authoring environment, your button hasn't changed. Folks, this is a fact of life with skins in Flash: there is no preview mode for skinning. Test your movie to see that your alteration appears as the new up skin in the published SWF. Hover over the button to verify that the other skins function as before.

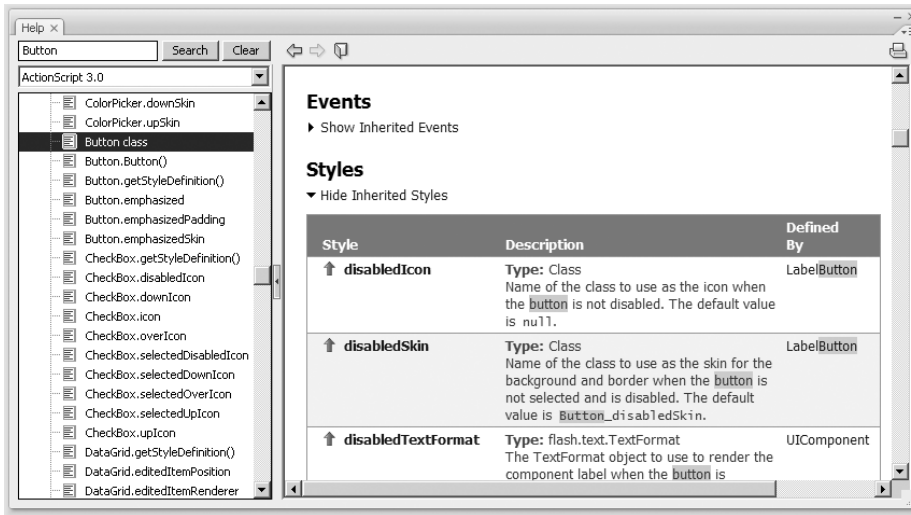
*To reskin a component completely, every skin symbol must be edited or replaced.*

## Styling components

As you've seen, components are easy enough to customize, even if a complete job takes a while. You may have noticed an important omission, however, while poking around the skin symbols. Even though the `Button` component features a text label, none of the skins contains a text field. What if you want a different font in there, or at least a different color? ActionScript to the rescue.

Each component has its own list of styled elements. Many overlap, but you can see the definitive list for each in the class entry for that component. For example, filter the Help panel for ActionScript 3.0 (as shown in Figure 9-9), search the word *Button* to find the `Button` class entry, and then browse the `Styles` heading. Don't forget to click the `Show Inherited Styles` hyperlink to see the full listing.

Components that include text elements, such as the `Button` component, support the inherited `UIComponent.textFormat` style, which lets you make changes to the button's label. Other button styles include the inherited `LabelButton.icon`, which lets you specify an optional image for the button in addition to text.



**Figure 9-9.** UI component styles are listed under the class entry for each component in the Help panel.

For this sort of styling, ActionScript allows you to affect the following:

- All components in a document
- All components of a certain type (for example, all `Button` components)
- Individual component instances

Let's see it in action:

1. Open the `Button04.fla` file that accompanies this chapter. You'll see three instances of the `Button` component and one of the `CheckBox` component (as shown in Figure 9-10). Note that each has its own label.



**Figure 9-10.** Styling is about to change these components.

2. Open the Actions panel (Window ► Actions) and type the following ActionScript into frame 1 of the scripts layer:

```
import fl.managers.StyleManager;
import fl.controls.Button;
```

```

this.stop();

var fmt1:TextFormat = new TextFormat();
fmt1.bold = true;
fmt1.color = 0xFF0000;

var fmt2:TextFormat = new TextFormat();
fmt2.color = 0x0000FF;
fmt2.bold = false;

StyleManager.setStyle("textFormat", fmt1);
StyleManager.setStyle(Button, "textFormat", fmt2);

btn2.setStyle("icon", "star");

```

Test the movie and note the following changes:

- The check box's label is red and bold.
- The buttons' labels are blue and not bold.
- The second button contains an icon.

Chapter 6 discusses the `TextFormat` class in detail, but there are a few twists here that deserve some clarification.

First up are the opening two lines, which make use of the `import` statement. We've been sidestepping this one so far because the `import` statement isn't often necessary in timeline code. In ActionScript 3.0 class files—that is, code written outside of Flash altogether—the `import` statement is not only more prevalent, it's actually *required* at the opening of each class to let the compiler know which other classes you intend to use. In contrast, Flash takes care of this for you—for the most part—with keyframe scripts. This just happens to be an exception. Without those first two lines, Flash will get confused about what you mean later when you mention `StyleManager` and `Button` directly.

*These hierarchical class arrangements are called packages. To find the package for other components so that you can carry the preceding styling knowledge to other scenarios, look up the component's class in the *ActionScript 3.0 Language and Components Reference*. The package is always listed first.*

Two variables, `fmt1` and `fmt2`, are declared and set to instances of the `TextFormat` class, each with its own styling. Here's where it gets interesting. The `StyleManager` class has two methods you can use to apply styling to components. Both methods are static, which means they're invoked on the class itself, rather than an instance. The first of these, `StyleManager.setStyle()`, applies formatting to all components. In this case, we're setting the `textFormat` style of all components that have a `textFormat` property to the `fmt1` `TextFormat` instance. We programmed this style to make text red (0xFF0000) and bold, and it is indeed applied to all three buttons and the check box. You can specify any style you like, but the `textFormat` style is common to many.

“Wait a minute, guys,” you may be saying. “Only the check box is red!” This is true. The reason for this is the other method, `StyleManager.setComponentStyle()`. That one applies styling to all components of a certain type, which explains the fact that it accepts three parameters. Here, we’ve specified `Button`, and then set the `textFormat` style of all `Button` instances to `fmt2`. This overrides the red, bold formatting of `fmt1` applied in the previous line. Comment out the second `StyleManager` line and test your movie again to prove it.

A good way to tell which style will take effect is to remember this: the more specific the style—for example, `Button` components vs. all components—the higher priority it has.

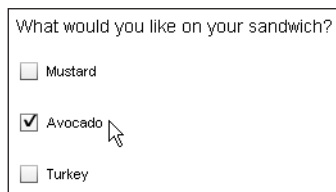
Finally, the `UIComponent.setStyle()` method is invoked specifically on the `Button` instance whose instance name is `btn`. It works just like `StyleManager.setStyle()` in that it accepts two parameters: the style to change and the setting to change it to. Here, the `LabelButton.icon` style, which `Button` inherits, is set to “star”, which is the linkage class of the star asset in the library. Right-click (PC) or Ctrl-click (Mac) the star asset and choose `Linkage` to verify.

And now you’ve had a quick tour of the lobby and one of the rooms here at the UI Component Hotel. There are other rooms, of course, some more elaborate than others, but the layout for each is basically the same.

*We’re pleased in a big way about the current UI component set, but even Paradise has its trouble. Some components—specifically `List`, `ComboBox`, `TileList`, and `DataGrid`—only obey certain styles, such as `textFormat`, when they’re set for all components by way of `StyleManager.setStyle()`. For component-specific and per-instance formatting, these four culprits require something called a custom cell renderer, which gets into the sort of programming not covered by this book. You have two workarounds: either set `textFormat` for all, then tweak other components’ styles individually, or—for `List`, `TileList`, and `DataGrid` only—specify the instance name and use `setRendererStyle()` instead.*

## CheckBox component

You met `CheckBox` briefly in the “Button component” section, but let’s take a closer look. This component is essentially a toggle button with its label on the side. Click the box or its label, and the box gets a check mark (as shown in Figure 9-11); click again, and the check mark goes away.



**Figure 9-11.** The `CheckBox` component is essentially a toggle button with its label on the side.

The Parameters tab of the Property inspector is fairly light for `CheckBox`: `label` sets the text label (left, right, top, or bottom), `labelPlacement` determines the position of the label, and `selected` lets you show an instance with the check mark by default. Double-click any `CheckBox` instance to change the skinning for all. Styling works as described in the “Button component” section.

*Using one or more instances of the `CheckBox` component in your movie will add 15 KB to the SWF if no other components share the load.*

Let’s take a look at how to interact with check boxes via `ActionScript`:

1. Open the `CheckBox.fla` file that accompanies this chapter. Note that each `CheckBox` instance has its own label and instance name.
2. Open the Actions panel (Window ► Actions) and enter the following `ActionScript` into frame 1 of the scripts layer:

```
addEventListener(Event.CHANGE, changeHandler);

function changeHandler(evt:Event):void {
    var str:String = "";
    if (cb1.selected == true) {
        str += cb1.label + "\n";
    }
    if (cb2.selected == true) {
        str += cb2.label + "\n";
    }
    if (cb3.selected == true) {
        str += cb3.label;
    }
    output.text = str;
}
```

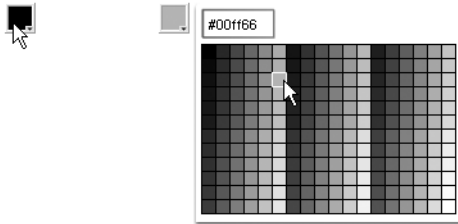
This assigns an event handler to the main timeline, listening for `Event.CHANGE` events. This event handler could have been attached to each `CheckBox` instance individually, but by doing it this way, the events of all three can be handled at the same time. When any of the three `CheckBox` instances is changed by clicking, each member of the group is checked in turn to see if it is selected. If so, the value of its label is added to a string that is ultimately sent to a text field beneath the check boxes.

## ColorPicker component

`ColorPicker` is a fun component, because nothing like it exists in the realm of HTML—at least, not without a swarm of complicated `JavaScript`!—but it’s common enough in applications like Microsoft Word, Adobe Photoshop, and even Flash itself. In a nutshell, the `ColorPicker` component is a clickable color chip that reveals an assortment of colors when



pressed (as shown in Figure 9-12). It allows the user to choose one of the presented colors or optionally to type in a hex value, at which point the chosen color is available for use.



**Figure 9-12.** The ColorPicker component lets users choose from a range of colors.

Double-clicking a ColorPicker instance inside the authoring environment makes its skins editable, and styling works the same as it does for the Button component. The palette of colors displayed by this component is also editable, but requires just a bit of ActionScript, as shown in the following code.

*Using one or more instances of the ColorPicker component in your movie will add 19 KB to the SWF if no other components share the load.*

The ColorPicker.fla file that accompanies this chapter shows this component in action.

1. Open the ColorPicker.fla file and note that the component itself has the instance name cp. The dynamic text field next to it has the instance name poem. Click into frame 1 of the scripts layer and open the Actions panel (Window ► Actions) to see the following ActionScript:

```
var fmt:TextFormat = new TextFormat();

cp.addEventListener(
    Event.CHANGE,
    function changeHandler(evt:Event):void {
        fmt.color = cp.selectedColor;
        poem.setTextFormat(fmt);
    }
);
```

Here, a variable, `fmt`, is declared and set to an instance of the `TextFormat` class. An `Event.CHANGE` event listener is assigned to the `ColorPicker` instance, `cp`—it does two things. First, it sets the `TextFormat.color` property of the `fmt` instance to the selected color of the `cp` instance. Second, it applies that format to the `poem` text field.

*See Chapter 6 for more information on the TextFormat class.*

2. Now let's determine what colors to display. Update the existing ActionScript to look like this:

```
var fmt:TextFormat = new TextFormat();

cp.colors = new Array(
    0x6E1E46,
    0xA12F1C,
    0xD47565,
    0x557A40,
    0x79A11C
);
cp.selectedColor = cp.colors[0];

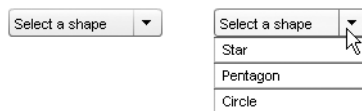
cp.addEventListener(
    Event.CHANGE,
    function changeHandler(evt:Event):void {
        fmt.color = cp.selectedColor;
        poem.setTextFormat(fmt);
    }
);
```

Specifying your own color palette couldn't be easier. Just provide the desired hexadecimal values (up to 1,024 individual colors!) as array elements to the `ColorPicker.colors` property of your component instance. To configure the color chip's initial display color, set the `ColorPicker.selectedColor` property. (Here, it's set to the first element in the colors array.)

3. Drag the `ColorPicker` instance to the lower-right corner of the stage. Test the movie to see that the pop-up color palette is smart enough to position itself to the upper left of the color chip. Note that in the `Parameters` tab of the Property inspector, the color palette's text field can be hidden by setting the `showTextField` parameter to `false`. You'll also see an alternate way to set the component's `selectedColor`.

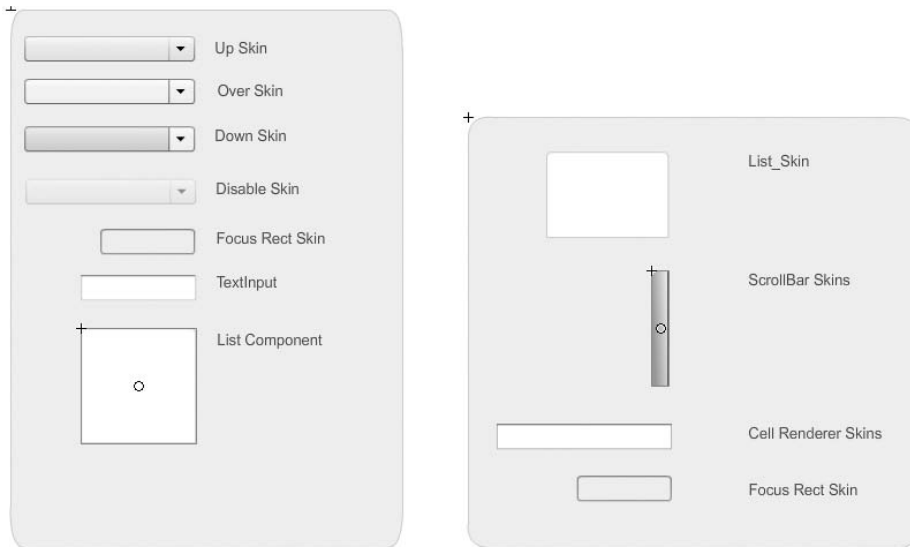
## ComboBox component

The `ComboBox` component is very much like the `<select>` element in HTML; specifically, the `<select>` element without its optional `size` and `multiple` attributes. It gives users the ability to make one selection at a time from a drop-down list (see Figure 9-13). In addition, the component can be made editable, which lets the user manually type in a custom selection.



**Figure 9-13.** ComboBox allows users to make one selection at a time from a drop-down list.

ComboBox skinning is a little more complicated than Button component skinning, but the basic approach is the same. The reason for the complexity is the ComboBox combines two other components, List and TextInput, which are described later in this chapter. Adding a ComboBox instance to your movie puts three components into your library—ComboBox, List, and TextInput—plus the Component Assets folder used by all UI components. Double-clicking a ComboBox instance in the authoring environment opens the first tier of skins (see the left image in Figure 9-14). Double-clicking the List element in this tier opens up the skins for the embedded List component (the right image in Figure 9-14).



**Figure 9-14.** ComboBox skins (left) include nested elements, such as List skins (right).

In turn, the skins for List include a third tier for scrollbars. In spite of this nesting, individual skins are nothing more than symbols, usually with 9-slice guides, such as the up and over skins for the Button component. Styling works the same as it does for the Button component. The `textFormat` style, in particular, can only be set for all instances of the ComboBox component by way of the `StyleManager.setStyle()` method.

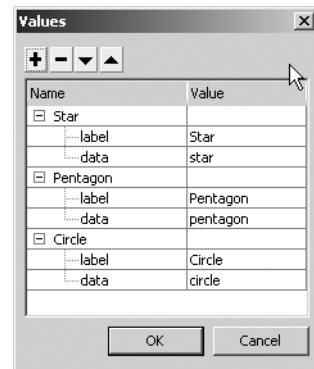
*Using one or more instances of the ComboBox component in your movie will add 35 KB to the SWF if no other components, other than the automatically included List and TextInput, share the load.*

1. Open the ComboBox.fla file that accompanies this chapter and select the ComboBox instance on the stage by clicking it once. Note that in the Parameters tab of the Property inspector some information has already been entered into the dataProvider parameter shown in Figure 9-15. This is an array of objects, each of which represents the visible portion of a drop-down choice (label) and the hidden value each label contains (data).



**Figure 9-15.** An array of objects defines the labels and data that populate a ComboBox.

2. Double-click the right column of the dataProvider row to open the Values dialog box shown in Figure 9-16.
3. Click the + button in the Values dialog box to create a new entry, which will appear below the existing Circle entry. Double-click the right column of the label row and change the existing stand-in label to Square. Double-click the right column of the data row and enter the value square. Pay attention to the capitalization. Test your movie to verify that the combo box now includes a Square choice that changes the shape to its right.



**Figure 9-16.** The Values dialog box lets you specify the content and order of a ComboBox instance.

How does this work? Let's take a look. The shapes symbol in the library contains a series of shapes drawn every few frames of its own timeline. Frame labels are provided for each shape, and it is these frame labels that are represented by the data row in the Values dialog box. Click into frame 1 of the scripts layer to see the ActionScript that pulls this off:

```
cbx.addEventListener(
    Event.CHANGE,
    function(evt:Event):void {
        shapes.gotoAndStop(cbx.selectedItem.data);
    }
);
```

The combo box is referenced by its instance name, cbx. An Event.CHANGE event triggers a function that tells the shapes instance to stop at the frame label determined by the selected item's data property of the cbx instance.

4. To populate the combo box by way of ActionScript, add the following line before or after the existing code:

```
cbx.addItem({label:"Triangle", data:"triangle"});
```

Pretty straightforward! The other parameters in the Parameters tab are just as intuitive: `editable` determines whether the user can type in a custom selection (if so, check for this value by referencing the combo box's instance name, and then the `text` property), `prompt` determines the default text (in this example, the phrase "Select a shape"), and `rowCount` determines how many selections to show in the drop-down list (if there are 15 selections and the value of `rowCount` is 5, only five will show, but the rest will be available via scrollbar).

## DataGrid component

The DataGrid is the most complex component in the UI arsenal. Its purpose falls almost entirely in the realm of übergeek interface programmers, but we're going to give you a cursory look, including a basic sample file. In short, the DataGrid component gives you a spreadsheet-like, sortable display for tabular data, as shown in Figure 9-17.



Numeric	English	German	French
1	one	eins	un
2	two	zwei	deux
3	three	drei	trois
4	four	vier	quatre
5	five	fünf	cinq

**Figure 9-17.** DataGrid displays scrollable, sortable tabular data.

*Using one or more instances of the DataGrid component in your movie will add 40 KB to the SWF if no other components share the load.*

See the `DataGrid.fla` file that accompanies this chapter for a working demonstration. Click into frame 1 of the scripts layer to see the ActionScript. Here's a bird's eye view of that code:

```
dg.addColumn("num");
dg.addColumn("eng");
dg.addColumn("ger");
dg.addColumn("fre");
```

These first lines reference the DataGrid component's instance name, `dg`, and instructs the component to add four columns. These column names are arbitrary and, here, represent a column for numbers, and then their English, German, and French equivalents.

```
dg.addItem({num:1, eng:"one", fre:"un", ger:"eins"});
dg.addItem({num:2, eng:"two", fre:"deux", ger:"zwei"});
dg.addItem({num:3, eng:"three", fre:"trois", ger:"drei"});
```

```

dg.addItem({num:4, eng:"four", fre:"quatre", ger:"vier"});
dg.addItem({num:5, eng:"five", fre:"cinq", ger:"fünf"});
dg.addItem({num:6, eng:"six", fre:"six", ger:"sechs"});
dg.addItem({num:7, eng:"seven", fre:"sept", ger:"sieben"});
dg.addItem({num:8, eng:"eight", fre:"huit", ger:"acht"});
dg.addItem({num:9, eng:"nine", fre:"neuf", ger:"neun"});
dg.addItem({num:10, eng:"ten", fre:"dix", ger:"zehn"});

```

There is not a way to populate DataGrid instances from the Parameters tab of the Property inspector, and we're sure you can see why. It's much easier to type in the data in the relatively spacious environs of the Actions panel.

```

dg.getColumnAt(0).headerText = "Numeric";
dg.getColumnAt(1).headerText = "English";
dg.getColumnAt(2).headerText = "German";
dg.getColumnAt(3).headerText = "French";

```

These lines make the header text a bit more “friendly” to the eye. Test the movie at this point to see how it all comes together. Click the headers to sort each column. When you sort the Numeric column, you'll see something odd. By default, sorting is alphabetical, which puts the numbers 1 and 10 right next to each other. To fix that for columns that contain numerical data, remove the comment (*//*) from the final line of ActionScript so that it looks like this:

```

dg.getColumnAt(0).sortOptions = Array.NUMERIC;

```

This is great for displaying data, but what about retrieving what cell has been selected? Yeah, we thought that was a good question, too. The `selectedItem` property for this component returns the contents of the whole row you click, not just the clicked cell. It is possible to return the selected cell, but it requires something called the `CellRenderer` class and more ActionScript, and frankly, it rockets way out of the atmosphere that makes this book breathable.

## Label component

Label is something of an oddball in the UI components collection. Unless you're an avid programmer, we're almost certain you'll want to forego Label in favor of a simple dynamic text field. Why? Practically speaking, from a designer's point of view, Label doesn't really *do* anything that a text field can't—and besides, by using a text field, you'll save the 14 KB that an instance of Label would have brought to the table.

Labels don't really have skins, and double-clicking an instance will tell you as much. Styling works the same as for Button, but again, trust us on this one . . . just use a dynamic text field. If you still want to see a Label component in action, check out `Label.fl` in the exercise files.

*See Chapter 6 for a full discussion on text fields in Flash.*

## List component

The List component is akin to the `<select>` element in HTML when its optional `size` and `multiple` attributes are specified. This component is basically a combo box without the drop-down aspect—it's always dropped down—and it allows multiple selections, as shown in Figure 9-18.

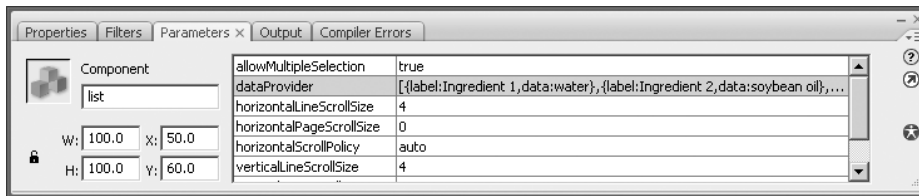


**Figure 9-18.** The List component optionally allows multiple selections.

Like `ComboBox`, the List component has nested skins, so when you double-click an instance in the authoring environment, the skins become available for editing in tiers. Styling is handled the same way as described in the “Button component” section; however, the `textFormat` style must be set using the `SelectableList.setRendererStyle()` method, as in `myList.setRendererStyle("textFormat", fmt)`.

*Using one or more instances of the List component in your movie will add 29 KB to the SWF if no other components share the load.*

The Parameters tab in the Property inspector is relatively hefty for the List component, and the Component Inspector panel comes in handy for looking over this component's settings. Most of the choices pertain to scrolling (the distance to scroll horizontally and vertically, whether scrolling should be automatic or constant, etc.), but the important parameters are `allowMultipleSelection` and `dataProvider` (see Figure 9-19).



**Figure 9-19.** The important parameters for the List component are `allowMultipleSelection` and `dataProvider`.

To populate your user's choices in a given List instance, double-click the right column of the `dataProvider` row and use the Values dialog box as described in the “ComboBox component” section. Setting `showMultipleSelection` to `true` (the default is `false`) lets your users hold down `Ctrl` (PC) or `Cmd` (Mac) while they click in order to select more than one of the listed choices. (just like the `multiple` option in HTML).

1. Open the `List.fla` file that accompanies this chapter. Note that the instance name for the List instance is `list`, which only works because ActionScript is a case-sensitive language—you couldn't call it `List`, because that's the name of the class that defines this object. In your own work, you'll want to use an instance name that describes the list's use (in this case, that might be the word *ingredients*). Note that the dynamic text field, next to the List instance, has the instance name `output`.

2. Click into frame 1 of the scripts layer and type the following ActionScript:

```
list.addEventListener(
    Event.CHANGE,
    function(evt:Event):void {
        var str:String = "The secret ingredient(s): ";
        for (var i:uint = 0; i < list.selectedItems.length; i++) {
            str += list.selectedItems[i].data;
            if (i < list.selectedItems.length - 1) {
                str += ", ";
            } else {
                str += ".";
            }
        }
        output.text = str;
    }
);
```

This one may look more complicated than it actually is, so let's break it down. As always, we're using `addEventListener()` to associate a function with an event. In this case, the event is `Event.CHANGE` and the function does three things.

First, the variable `str` holds the phrase "The secret ingredient(s): ".

```
var str:String = "The secret ingredient(s): ";
```

Next, a `for` loop repeats a particular set of actions. The duration of the loop depends on the number of selected items, based on the `Array.length` property of the `selectedItems` property, which is an array. The variable `i` starts at zero and increments at each "lap" around the loop, so that the line

```
str += list.selectedItems[i].data;
```

refers to the first selected item (item 0), and then the second selected item (item 1), and so on, of the `List` instance. The reason there's a `.data` tacked onto the end is because `List` items are made up of two parts: `label` and `data`, which are—bingo!—the elements that comprise the `dataProvider` parameter described previously.

An `if` statement adds a comma between items in the middle and a period after the item at the end. Finally, the `str` variable, which has continuously been updated by this process, is set to the `TextField.text` property of the `output` instance.

The net result is that `List` selections populate a dynamic text field with the ingredients of Kraft Cucumber Ranch dressing.

*For extra credit, add the line `list.addItem({label:"Ingredient 11", data:"natural flavor"})`; after the existing ActionScript to show that it's also possible to populate a `List` instance programmatically.*



## NumericStepper component

NumericStepper is a compact little gadget that lets the user specify a numeric value, either by typing it in or by clicking up and down arrow buttons (see Figure 9-20). You, as a designer, can specify your own desired minimum and maximum values, as well as the size of each increment (count by ones, by twos, by tens, etc.), which can be set via the Parameters tab of the Property inspector.

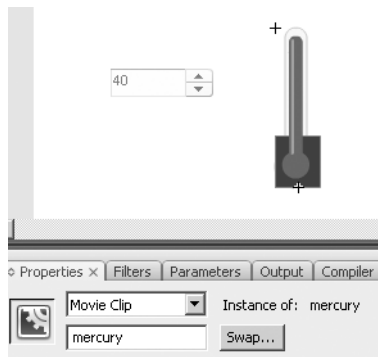


**Figure 9-20.** The NumericStepper component

NumericStepper's skins can be edited by double-clicking an instance, and styling can be applied as described in the "Button component" section. This component carries with it the TextInput component, so you'll see both in your library if you add NumericStepper to your movie.

*Using one or more instances of the NumericStepper component in your movie will add 18 KB to the SWF if no other components (other than the automatically included TextInput) share the load.*

1. Open the NumericStepper.fla file that accompanies this chapter. Note that the NumericStepper instance has the instance name `ns` and that the thermometer movieclip has the instance name `thermometer`. Double-click that movieclip to enter its timeline, and you'll see a red rectangle (masked by a green shape) with the instance name `mercury` (see Figure 9-21). You're going to set the height of this nested movieclip based on the value property of the NumericStepper instance.



**Figure 9-21.** The mercury will rise and fall in response to NumericStepper clicks.

2. Select **Edit > Edit Document** to return to the main timeline. Click into frame 1 of the scripts layer and type the following ActionScript:

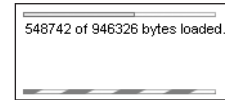
```
ns.addEventListener(
    Event.CHANGE,
    function(evt:Event):void {
        thermometer.mercury.height = ns.value;
    }
);
```

3. Test your movie and click the up and down arrow buttons to see it in action.

## ProgressBar component

Used often for preloading, the `ProgressBar` component (shown in Figure 9-22) gives you a rising thermometer-style animation to display load progress when loading files of known size, and a bar-ber pole-style animation to indicate that the user must wait (e.g., for files of unknown size to load or for processes to finish).

This component doesn't have a whole lot to skin, but you can access what's there by double-clicking a `ProgressBar` instance. Styling works as it does for the `Button` component, but `ProgressBar` doesn't even have text, so your styling choices are fairly slim. (Yes, Figure 9-22 shows text, but that's an example of the `Label` component.)



**Figure 9-22.** The `ProgressBar` component indicates load progress (top), and also presents a “waiting” animation (bottom).

*Using one or more instances of the `ProgressBar` component in your movie will add 16 KB to the SWF if no other components share the load—that means 16 KB of non-preloadable content (the preloader itself!), so don't put much else into the frame that contains the `ProgressBar` instance.*

1. Open the `ProgressBar.fla` file that accompanies this chapter. Note that a `ProgressBar` instance exists in frame 1 with the instance name `pb`, as well as a text field with the instance name `output`. In frame 5, you'll find a fairly heavy image of a homegrown onion, snapped years ago by one of the authors. In the scripts layer, there's a `stop()` action in frames 1 and 5.
2. Click into frame 1 of the scripts layer and type the following ActionScript:

```
root.loaderInfo.addEventListener(
    Event.COMPLETE,
    function(evt:Event):void {
        play();
    }
);
```

```
pb.source = root.loaderInfo;
```

So far, here's what's going on. An `Event.COMPLETE` event kicks the playhead back into gear. In other words, when the root (the movie itself) has completed loading, the playhead will play until frame 5 stops it again, revealing the onion.

The `ProgressBar` part is practically magic. Simply set the `ProgressBar.source` property to the root's `loaderInfo` property. It couldn't be simpler. In Chapter 13, you'll see additional loading examples, including one that uses the `Loader` class. In a case like that, you would set the source property of your `ProgressBar` instance to the `Loader.contentLoaderInfo` property of the `Loader` instance. Later in this chapter, you'll see an example using the `UI Loader` component, in which case it is sufficient merely to point to the `UI Loader` instance itself.

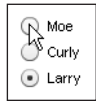
3. Now, if you also want to display a text message indicating a percent loaded, you could do something like the following. Add a few more lines below the existing code:

```
pb.addEventListener(
    ProgressEvent.PROGRESS,
    function(evt:ProgressEvent):void {
        output.text = Math.floor(pb.percentComplete).toString() + "%";
    }
);
```

The `ProgressBar` component features a `percentComplete` property, which we're using here. The `addEventListener()` method is invoked against the `pb` instance, and the function it performs sets the output text field's `text` property to a rounded-down string version of the progress percentage—with the percent sign tacked onto the end for good measure.

## RadioButton component

Radio buttons are social creatures. They belong in groups, and courteously defer to each other as each takes the spotlight. What are we talking about? We're talking about a component identical in functionality to radio buttons in HTML. Groups of these are used to let the user make a single selection from a multiple-choice set (see Figure 9-23).

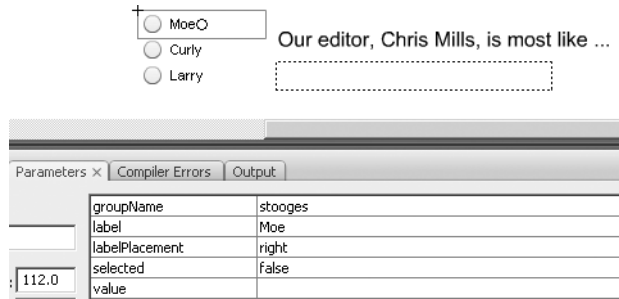


**Figure 9-23.** The `RadioButton` component lets the user make a single selection from a multiple-choice set.

Double-clicking a `RadioButton` instance provides access to its skins, which you can edit as described in the “Button component” section. Styling works the same way.

*Using one or more instances of the `RadioButton` component in your movie will add 16 KB to the SWF if no other components share the load.*

1. Open the `RadioButton.fla` file that accompanies this chapter. Because radio buttons work in groups, the `Parameters` tab of the Property inspector has a “group think” parameter we haven't seen with other components: `groupName`. Select each of the three radio buttons in turn and verify that each belongs to the same group, `stooges`, even though each has its own distinct label: `Moe`, `Curly`, and `Larry` (see Figure 9-24). Note also the empty dynamic text field whose instance name is `output`. You're about to wire up the radio buttons to that text field.



**Figure 9-24.** RadioButton instances must be associated with a group name.

2. Click into frame 1 of the scripts layer and type the following very condensed but interesting ActionScript:

```
rb1.group.addEventListener(
    Event.CHANGE,
    function(evt:Event):void {
        output.text = rb1.group.selection.label;
    }
);
```

What makes this interesting? In most of the event-handling samples in this book, you've invoked the `addEventListener()` method on an object that you personally gave an instance name. Here, that might have been `rb1`—but that's not the focal point in this case. You're not adding an event listener to a particular radio button, but rather to the *group* these buttons belong to. The `RadioButton` class provides a `group` property, which means that each instance knows what group it belongs to. It's the group dispatching the `Event.CHANGE` event, which occurs when any one of these radio buttons is clicked.

It doesn't matter which radio button's `group` property you use, because all of them point to the same `RadioButtonGroup` instance. The associated function updates the output text field by sending it the selected button in this group—in particular, that button's `label` property, which is either Moe, Curly, or Larry.

*Note that the Parameters tab gives you the option to supply a value for each radio button. This allows you to say one thing and do another, just as in the List example—except that the List choices were label and data; here, they're label and value, and the data type of value is Object (not String). The text field wants a string, so you would change that line of ActionScript to `output.text = rb1.group.selection.value.toString();`*

## ScrollPane component

The ScrollPane component lets you have eyes bigger than your stomach. If you want to display a super-large image—so large that you’ll need scrollbars—ScrollPane is your component; Figure 9-25 shows it in action.



**Figure 9-25.** ScrollPane provides optional scrollbars to accommodate oversized content.

ScrollPane has nested skins because of its scrollbars, so double-clicking an instance during authoring will open up its skin elements in tiers. Styling works the same as described in the “Button component” section, though with no text elements, most of your customization work will probably center around skins.

*Using one or more instances of the ScrollPane component in your movie will add 21 KB to the SWF if no other components share the load.*

1. In this example, there’s no need for ActionScript. Open the ScrollPane.fla file that accompanies this chapter. Select the ScrollPane instance and click the Parameters tab of the Property inspector.
2. In the Parameters tab, double-click the right column of the source row. Type Onion.jpg and test the movie. Pretty slick! The source parameter can be pointed to any file format that Flash can load dynamically, including GIFs, PNGs, and other SWFs.

## Slider component

The Slider component is conceptually the same thing as NumericStepper, except that instead of clicking buttons to advance from one number to the next, the user drags a knob along a slider, as shown in Figure 9-26. You, as designer, are responsible for setting the minimum and maximum values, and this component lets you specify whether sliding is smooth or snaps to increments specified by you.



**Figure 9-26.** Slider lets the user drag a handle back and forth to specify a value.

Slider has no text elements, so styling is fairly light. What's there works as it does for the Button component. Skinning also works as it does for Button: double-click a Slider instance in the authoring environment to change the knob and track skins.

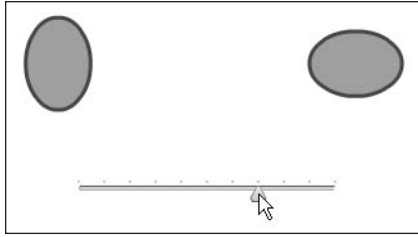
*Using one or more instances of the Slider component in your movie will add 17 KB to the SWF if no other components share the load.*

1. Open the Slider.fla file that accompanies this chapter. Note that the instance name for the Slider instance is `slider`, which only works because ActionScript is a case-sensitive language—you couldn't call it `Slider`, because that's the name of the class that defines this object. Note, also, the instance names `circle1` and `circle2` on the two circles. You're about to wire up the Slider component to adjust their width and height.
2. Click into frame 1 of the scripts layer and type the following ActionScript:

```
slider.addEventListener(
    Event.CHANGE,
    function(evt:Event):void {
        circle1.scaleX = slider.value / 100;
        circle2.scaleY = slider.value / 100;
    }
);
```

When the `Event.CHANGE` event is dispatched—this happens as the knob moves along the track—the slider's `value` property is used to update scaling properties of the `circle` movieclips. Why divide by 100? In movieclip scaling, 0% is 0 and 100% is 1. Because the Slider instance happens to have its `maximum` parameter set to 100, the division puts `value` into the desired range, as shown in Figure 9-27.

Be sure to experiment with the parameters in the Property inspector's Parameters tab. Most of them are intuitive, but `liveDragging` probably isn't. The `liveDragging` parameter tells Slider how often to update its `value` property. Change it to `false` and test again to see the circles resize less smoothly.



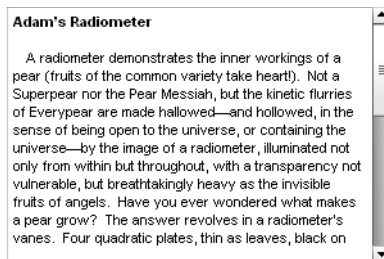
**Figure 9-27.** A single Slider instance can adjust many objects. Hey, that looks like a face!

*You may be surprised to find a direction parameter (its values are horizontal and vertical). Why not just use the Free Transform tool to rotate this slider? Well, try it. We'll wait . . . Kinda weird, right? It doesn't work. Components are a sophisticated phenomenon, even though they look so simple. What if you want a slanted slider, not horizontal or vertical? Here's a trick: select the Slider instance, convert it to a movieclip (Modify ► Convert to Symbol), and give that movieclip an instance name. When both the movieclip (here, sliderClip) and its nested Slider have instance names, you're set.*

```
sliderClip.slider.addEventListener(
    Event.CHANGE,
    function(evt:Event):void {
        circle1.scaleX = sliderClip.slider.value / 100;
        circle2.scaleY = sliderClip.slider.value / 100;
    }
);
```

## TextArea component

Chapter 6 introduced you to text fields. Consider the TextArea component a text field in a tux. It has an attractive, slightly beveled border, lets you limit how many characters can be typed into it (like input text fields), and is optionally scrollable (see Figure 9-28). This component is akin to the <textarea> element in HTML.



**Figure 9-28.** TextArea is the James Bond of text fields.

TextArea is skinnable, but its parts are few. You'll see a nested skin for the scrollbars when you double-click an instance in the authoring environment. More likely, you'll want to style its text contents, which works as described in the "Button component" section.

*Using one or more instances of the TextArea component in your movie will add 21 KB to the SWF if no other components (other than the automatically included UIScrollBar) share the load.*

See the TextArea.fla file that accompanies this chapter for an example of populating a TextArea instance with text. (We figured it would be cruel to make you type in a lengthy bit of sample text on your own.) Note that the TextArea component can display HTML text, as shown in the sample file, or plain text. Use the htmlText or text property accordingly. Note that the Parameters tab of the Property inspector only shows a text parameter for supplying text. We can't imagine anyone using that tiny space to enter more than a sentence.

## TextInput component

The TextInput component is the single-line kid brother to TextArea. For this reason, to trump it up, we'll show it displaying one of the shortest short stories in the world, attributed to Ernest Hemingway (see Figure 9-29).

For sale: baby shoes, never used.

**Figure 9-29.** TextInput features a slightly beveled look.

TextInput is primarily used to collect typed user input, such as happens in HTML-based "contact us" forms, and can even be set to display password characters as asterisks. The component is skinnable—just double-click an instance in the authoring environment—but there's not much to skin. Styling works as described in the "Button component" section.

*Using one or more instances of the TextInput component in your movie will add 15 KB to the SWF if no other components share the load.*

1. Open the TextInput.fla file that accompanies this chapter. Note the two TextInput instances, with instance names input (top) and output (bottom). Select each component in turn and look at the Parameters tab as you do. For the top TextInput instance, the displayAsPassword and editable parameters are set to true. For the bottom, both of those parameters are set to false. You're about to make the upper component reveal its password to the lower one.



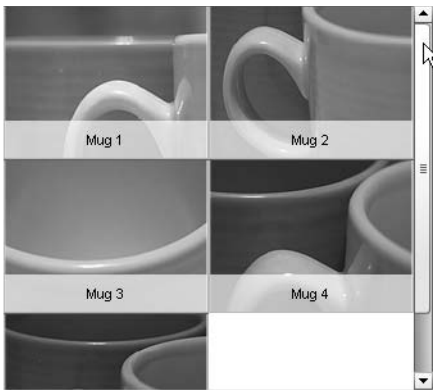
- Click into frame 1 of the scripts layer and type the following ActionScript:

```
input.addEventListener(
    Event.CHANGE,
    function(evt:Event):void {
        output.text = input.text;
    }
);
```

As text is typed into the upper TextInput instance, the Event.CHANGE event updates the lower instance's text content with that of the other. Because of the parameter settings, the text content is hidden above, but clearly displayed below.

## TileList component

TileList is not unlike the ScrollPane component. Both load files for display, optionally with scollbars, but TileList displays numerous files—JPGs, SWFs, and so on—in the tiled arrangement shown in Figure 9-30.



**Figure 9-30.** TileList displays a tiled arrangement of content, optionally scrolling as necessary.

Double-click a TileList instance to edit its skins. You'll see a second tier of skins for the scrollbars. Styling may be accomplished as described in the “Button component” section; however, the textFormat style must be set using the SelectableList.setRendererStyle() method, as in myTileList.setRendererStyle("textFormat", fmt).

*Using one or more instances of the TileList component in your movie will add 32 KB to the SWF if no other components share the load.*

There are quite a few parameters listed in the Parameters tab of the Property inspector for this component, but they're all easy to grasp. For example, there are settings for the width and number of columns, height and number of rows, direction or orientation (horizontal

or vertical), and scrolling settings (on, off, and auto, which makes scrollbars show as necessary). The `dataProvider` parameter is the most important, because that's where you define the content to show. It works the same as the `dataProvider` for `ComboBox`, except that instead of `label` and `data` properties, `TileList` expects `label` and `source`.

If you find the `Parameters` tab a bit confining, you can always use `ActionScript` to add items to `TileList` instances.

1. Open the `TileList.fla` file that accompanies this chapter. Note that the `TileList` instance has the instance name `tl`, and the dynamic text field below it has the instance name `output`.
2. Click into frame 1 of the `scripts` layer and type the following `ActionScript`:

```
tl.addItem({label:"Mug 6", source:"Mug06.jpg"});
tl.addItem({label:"Mug 7", source:"Mug07.jpg"});
tl.addItem({label:"Mug 8", source:"Mug08.jpg"});

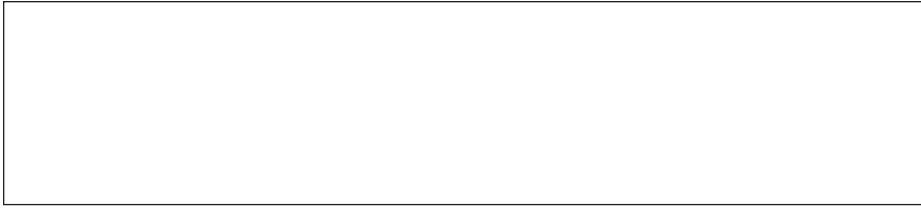
tl.addEventListener(
    Event.CHANGE,
    function(evt:Event):void {
        output.text = tl.selectedItem.label;
    }
);
```

The first three lines use practically the same approach we used in adding an additional item to the `ComboBox` instance in that section of the chapter. Here, they give us a few more mug shots (heh, mug shots—we love that joke). In the event handler, the function updates the output text field's text property with the `label` value of the tile list's selected item.

*TileList also supports multiple selections, like the List component. The sample code in the "List component" section provides the same basic mechanism you would use here, except instead of targeting the data property, you'll probably want to target `label`, as shown in the preceding single-selection sample.*

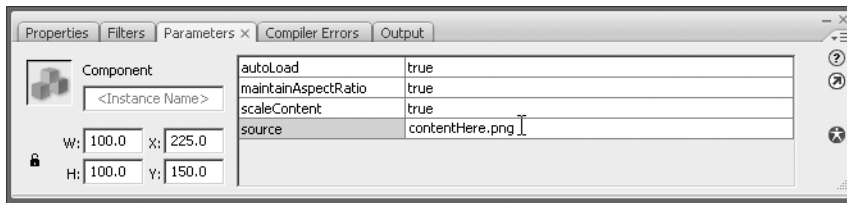
## UILoader component

If the Flash CS3 UI components all went to a Halloween party, `UILoader` would show up as the Invisible Man (see Figure 9-31).



**Figure 9-31.** Practically speaking, UI Loader has no visual elements (and yes, this figure is empty; it tickled us to include it).

So what's the point? Ah, but UI Loader is such a selfless, *giving* component! Its purpose is to load and display content other than itself. This keeps you from having to use the Loader class (described in Chapter 13)—in case the thought of ActionScript makes you feel like you just found half a worm in your apple. Simply enter a file name into the source parameter of the Property inspector's Parameters tab, and you're set (see Figure 9-32).

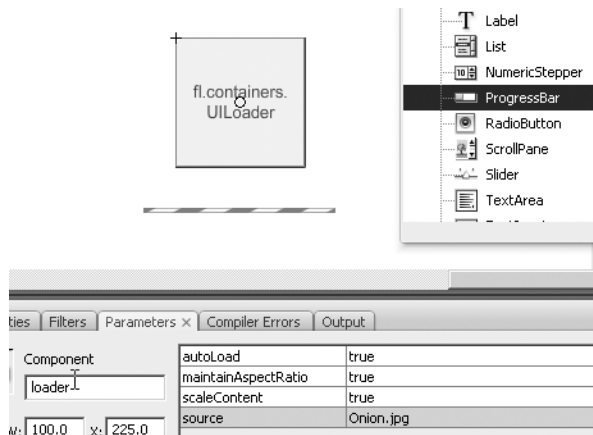


**Figure 9-32.** Just enter in the name of a supported file format, and Flash will load it.

*Using one or more instances of the UI Loader component in your movie will add 15 KB to the SWF if no other components share the load.*

1. Open the UI Loader.fla file that accompanies this chapter. Double-click the UI Loader instance if you like; you'll see message that no skins are available. Since we aren't speaking to this component with ActionScript—yet—it doesn't need an instance name. In the Parameters tab of the Property inspector, enter the file name Onion.jpg into the right column of the source row. This references a JPG file in the same folder as your FLA. Test your movie, and you'll see the onion load into its UI Loader container.
2. In the Parameters tab, change the maintainAspectRatio parameter to false and test again. This time, the onion loads a bit squished. Our personal preference is usually to maintain aspect ratio. The scaleContent parameter determines whether the loaded content is scaled or cropped in its container.

3. Our friend `ProgressBar` is about to make a cameo appearance. Drag an instance of the `ProgressBar` component to the stage below the `UI Loader` instance, and give the `UI Loader` instance the instance name `loader` (see Figure 9-33).



**Figure 9-33.** It's very easy to show the load progress of a `UI Loader` instance.

4. Select the progress bar, and in the `Parameters` tab, set its `source` parameter to `loader`—that's the instance name you just gave the `UI Loader` instance. You're associating the two. Test your movie, and then in the SWF window that opens, select `View > Simulate Download` to see some super-easy preloading action.
5. To wrap up, let's add a teensy bit of `ActionScript`. (Don't worry, that half a worm we mentioned earlier was just a centipede—half a centipede.) To make sure `ActionScript` talks to the `ProgressBar` instance, give it an instance name. We're using `pb`. Click into frame 1 of the `scripts` layer and type the following `ActionScript`:

```
pb.addEventListener(
    Event.COMPLETE,
    function(evt:Event):void {
        removeChild(pb);
    }
);
```

What does that do? That makes the progress bar disappear when loading is complete.

## UIScroller component

If you read any other sections of this chapter, you've probably already been introduced to the `UIScroller` component. This component is a humble but useful member of the team, as it allows things to have scrollbars. `UIScroller` is skinnable when you double-click any instance of it in the authoring environment. Styling doesn't make much sense, but it is possible as described in the "Button component" section.

*Using one or more instances of the UIScroller component in your movie will add 18 KB to the SWF if no other components share the load.*

So as to avoid repeating ourselves, we'll direct your attention to Chapter 6's "Scrolling Text" section to see this component in action.

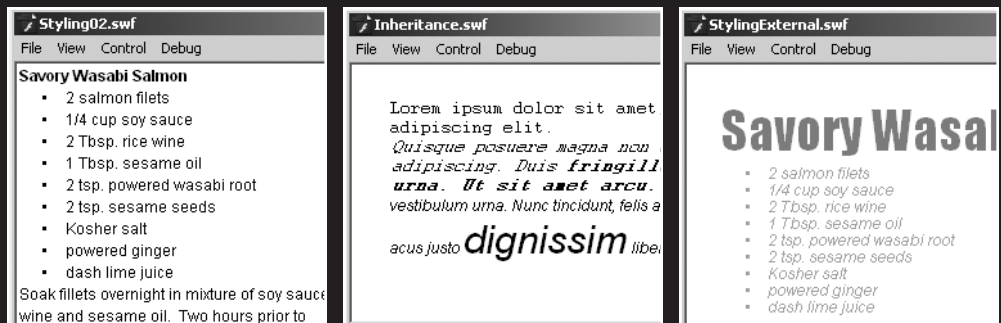
## What you've learned

- How to use the various Flash CS3 UI components
- How to write the ActionScript that controls components
- How to skin a component
- How to manage components in a Flash movie

In the next chapter, we'll show you how to use CSS to format text in a Flash movie. It isn't as dry as it sounds because you can do some pretty interesting things with CSS-formatted text in Flash CS3. What are they? Turn the page to find out.



## 10 CSS AND FLASH



Cascading Style Sheets (CSS) refers to a W3C (World Wide Web Consortium) specification that, in the W3C's own words, provides “a simple mechanism for adding style (e.g., fonts, colors, spacing) to Web documents ([www.w3.org/Style/CSS/](http://www.w3.org/Style/CSS/)).” Simple concept, sure—but, as any web developer will tell you, CSS can be a Marlboro-smokin', tumbleweed-kickin' maverick cowboy when it comes to corralling HTML. In other words, CSS is rugged, powerful, and does a great job at making HTML behave. Obviously, this is a good thing. But CSS can also be a bit hard to work with, which makes sense when you're dealing with a stubbly, saddle-sore buckaroo.

In the world of HTML, this trouble is due to the wide variety of browsers (and versions of browsers) in use by the general public. Each browser supports CSS to a varying, and often buggy, degree. In Flash, you have a lot less to worry about, even though the use of CSS requires ActionScript. Why are things easier in a SWF? The answer is mainly that Flash supports only a very small subset of the full CSS specification. This means that there are only a few cows to wrangle. As a Flash designer, you're not worried about half a dozen browsers, but merely a single Flash Player plug-in. As an extra plus, the supported CSS subset hasn't really changed since the feature was introduced in Flash MX 2004 (Flash Player 7).

*“Wait a minute, varmints,” you might be saying, “If CSS has been available since Flash Player 7, what you're really talking about are three Flash Players, versions 7 through 9. Don't try to pull a fast one on us!” Well, if we opened it up to ActionScript 2.0, you'd have a good point. Even so, the supported styles are pretty much the same; it's only the ActionScript nitty-gritty that's been updated. In any case, three plug-ins are nothing compared to a herd of browsers—and since we're only dealing with ActionScript 3.0 in this book, Flash Player 9 is the only one that counts. Yippee-ki-yay!*

What we'll cover in this chapter

- A brief overview of CSS, including what makes it so useful
- Some of the limitations of CSS in Flash
- How to generate and apply CSS in ActionScript
- The difference between element selectors and class selectors
- Custom HTML tags
- Inheritance basics
- How to style anchor tag hyperlinks
- How to embed fonts for CSS
- How to load styles from an external CSS file

Files used in this chapter:

- Styling01.fla (Chapter10/ExerciseFiles\_CH10/Exercise/Styling01.fla)
- Styling02.fla (Chapter10/ExerciseFiles\_CH10/Exercise/Styling02.fla)
- Styling03.fla (Chapter10/ExerciseFiles\_CH10/Exercise/Styling03.fla)
- Styling04.fla (Chapter10/ExerciseFiles\_CH10/Exercise/Styling04.fla)



- `ClassSelectors.fla` (Chapter10/ExerciseFiles\_CH10/Exercise/ClassSelectors.fla)
- `ElementSelectors.fla` (Chapter10/ExerciseFiles\_CH10/Exercise/ElementSelectors.fla)
- `Hyperlinks.fla` (Chapter10/ExerciseFiles\_CH10/Exercise/Hyperlinks.fla)
- `HyperlinksVaried.fla` (Chapter10/ExerciseFiles\_CH10/Exercise/HyperlinksVaried.fla)
- `Inheritance.fla` (Chapter10/ExerciseFiles\_CH10/Exercise/Inheritance.fla)
- `styles.css` (Chapter10/ExerciseFiles\_CH10/Exercise/styles.css)
- `StylingEmbeddedFonts01.fla` (Chapter10/ExerciseFiles\_CH10/Exercise/StylingEmbeddedFonts01.fla)
- `StylingEmbeddedFonts02.fla` (Chapter10/ExerciseFiles\_CH10/Exercise/StylingEmbeddedFonts02.fla)
- `StylingExternal.fla` (Chapter10/ExerciseFiles\_CH10/Exercise/StylingExternal.fla)

## The power of CSS

In a nutshell, the power of CSS is that it allows you to separate styling from informational content. In Flash, we're essentially talking about text. You'll wrap text content in HTML tags—that's one side of the coin—and you'll style those HTML tags with CSS—that's the other side. Flip that coin as you see fit: if you change your mind about how the text should look—regarding font, color, indentation, spacing, and the like—you can change the CSS without upsetting the text. The reverse is also true. Not only that, but styling can be applied to numerous text fields at once, and even managed from a convenient external file. As if that wasn't enough, this external style sheet can update a movie's styles without your having to recompile the SWF! Have we got your interest yet?

Here are the available style properties:

- `color`: This property determines the color of text, specified as a hexadecimal value preceded by the # sign, as in #FFFFFF, rather than the 0xFFFFFF you would use in ActionScript.
- `display`: This property determines how the styled object is displayed. Values include `inline` (displayed without a built-in line break), `block` (includes a built-in line break), and `none` (not displayed at all).
- `fontFamily`: This property allows you to specify fonts for text content—either a single font or comma-separated collection of fonts listed in order of desirability.
- `fontSize`: This property is used for specifying font size in pixels. Only number values are accepted (units such as pt or px are ignored).
- `fontStyle`: This property optionally displays text content in italics, if the font in use supports it. Values include `normal` and `italic`.

- **fontWeight**: This property optionally displays text content in bold, if the font in use supports it. Values include `normal` and `bold`.
- **kerning**: This property, if specified as `true`, allows embedded fonts to be rendered with kerning, if the fonts support it. **Kerning**, the removal of a bit of space between letters, is only applied in SWF files generated in the Windows version of Flash. Once the SWF is published, the kerning is visible both in Windows and Mac.
- **leading**: This property determines the amount of space between lines of text. Negative values, which are allowed, condense lines. Only number values are accepted (units such as `pt` or `px` are ignored).
- **letterSpacing**: Not to be confused with kerning, this property determines the amount of space distributed evenly between characters. Only number values are accepted (units such as `pt` or `px` are ignored).
- **marginLeft** and **marginRight**: These properties add marginal padding by the specified amount in pixels to the left and right. Only number values are accepted (units such as `pt` or `px` are ignored).
- **textAlign**: This property aligns text. Values include `left`, `center`, `right`, and `justify`.
- **textDecoration**: This property adds or removes underscoring by way of the `underline` and `none` values.
- **textIndent**: This property indents a text field by the specified amount in pixels—only number values are accepted (units such as `pt` or `px` are ignored).

Now let's roll up our sleeves and use some of these properties:

1. Open the `Styling01.fla` file that accompanies this chapter. There are a few things already in place for you. Note the two dynamic text fields, side by side, with instance names `unstyled` and `styled`. There's also a bit of ActionScript in frame 1 of the `scripts` layer, which does nothing more than build a string of HTML tags and apply that string to the `TextField.htmlText` property of the two text fields.
2. Test the movie to see two identical copies of the wasabi salmon recipe shown in Figure 10-1 (yup, it's a real recipe).

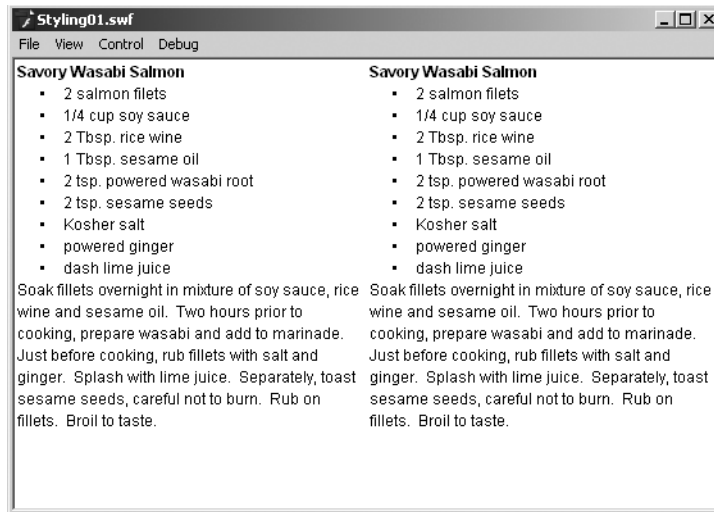


Figure 10-1. CSS is about to save you a lot of effort.

3. Click into frame 1 of the scripts layer and open the Actions panel. The first thing you need to do is import the `StyleSheet` class, otherwise none of this is going to work. Put your cursor in line 1, in front of the word `var`, and then press the Enter/Return key a couple times to make room. Type the following code in line 1 (see Figure 10-2):

```
import flash.text.StyleSheet;
```

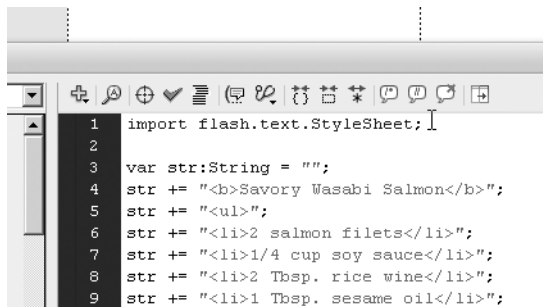


Figure 10-2. The `StyleSheet` class must be imported.

4. The thing about CSS in Flash is that styling must be applied to a text field before any text is added to it. We're going to leave the unstyled text field as is, in order to have a running comparison. The CSS that formats the styled text field will have to appear before the last line of ActionScript, because the last line actually provides the HTML text. Put your cursor in front of the last line of code and press Enter/Return three times. This is where the new ActionScript will go. Now, hold that thought.

How is this CSS thing going to work? That's a good question, and thankfully, the answer isn't especially hard, even though the process takes a few steps. First, you're going to create an instance of the `StyleSheet` class. Next, you'll decide on a handful of style properties. You'll repeatedly use the `StyleSheet.setStyle()` method to associate those properties with an HTML tag. Finally, you'll associate the `StyleSheet` instance itself with a given text field and add HTML content to that text field.

The crafty thing is that there are a number of ways to handle the `setStyle()` part. We're going to step you through a wordy approach first, because we think it best summarizes, on a conceptual level, what's going on. When you've seen that, we'll steer you toward a more compact approach, which will eventually lead toward an external CSS file, which is the most versatile way to handle styling in Flash.

5. OK, still holding the thought? Good. Put your cursor into the second of the three blank lines that precede the last line of code. Type the following ActionScript:

```
var css:StyleSheet = new StyleSheet();
var condensed:Object = new Object();

condensed.fontStyle = "italic";
condensed.color = "#A2A2A2";
condensed.leading = "-2";

css.setStyle("li", condensed);
styled.styleSheet = css;
```

Let's review what you've done so far. The first line declares a variable, `css`, that points to an instance of the `StyleSheet` class. The second line declares another variable, `condensed`, that points to an instance of the generic `Object` class—that's right, this is an `Object` object—and the next three lines set arbitrary properties of this new object; namely, `fontStyle`, `color`, and `leading`, each of which is set to a string value. The second-to-last line refers again to the `css` instance, using that instance to invoke `StyleSheet.setStyle()` with two parameters: an HTML tag to style and the object to style it with. Quite simply, this line says, "Any `<li>` tags in the house? If so, you're about to get comfy with the `condensed` object, whose instructions are to render you in italics, in the color `#A2A2A2`, and at a leading of `-2`." Finally, a text field whose instance name is `styled` has its `styleSheet` property set to the `css` instance.

6. Test the movie so far to see a change to all the `<li>` content, as shown in Figure 10-3. You can save and close the movie if you wish.

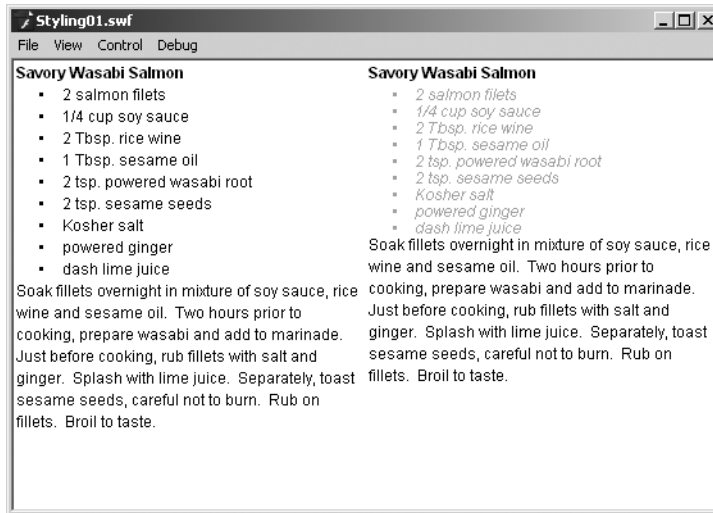


Figure 10-3. CSS styling applied to a series of <li> tags.

Pretty nifty! Now, in case you thought that ActionScript was a lot to type, keep in mind that what you've seen is the gabbiest of the styling approaches. It's possible to collapse five of those lines into one, which we'll do in just a moment. First, let's take a look at how this might have happened without CSS—because once you see that catastrophe, even this version will seem a welcome relief.

Taking just the first <li> tag's content, how would you apply italics? Easy enough; you'll remember from Chapter 6 that this happens with the <i> tag. So far, then, we've got one nested pair of tags:

```
<li><i>2 salmon filets</i></li>
```

What about the coloring? That's the <font> tag. Combined, that makes

```
<li><i><font color="#A2A2A2">2 salmon filets</font></i></li>
```

Almost done! The final style property is leading (the spacing between lines). In the HTML-only realm, that requires the Flash-specific <textformat> tag. This brings the combined total to the following example of spaghetti code:

```
<li><i><font color="#A2A2A2"><textformat leading="-2">2 salmon  
filets</textformat></font></i></li>
```

Multiply that by the nine bullet points in this recipe, and you've got carpal tunnel syndrome just waiting to happen! If you decide later to change the text color, you'll have to revisit all nine nested <font> tags and either edit or remove them. It's a mess. Definitely, the CSS styling mechanism is the nicer pick. All the more so if we can reduce the lines of ActionScript.

In order to accomplish that, we're going to rely on a shortcut in creating our Object instance, involving the use of the {} characters. Our `setStyle()` line will continue to use "li" as the first parameter, but the second parameter will be composed of a single shortcut object that holds all three styling properties at once, as shown in Figure 10-4.

```
var condensed:Object = new Object();

condensed.fontStyle = "italic";
condensed.color = "#A2A2A2";
condensed.leading = "-2";

css.setStyle("li", condensed);
```

**Figure 10-4.** These lines can be folded into a single object reference.

The actual ActionScript looks like this:

```
css.setStyle("li", {fontStyle: "italic", color: "#A2A2A2",
leading: "-2"});
```

This brings the full ActionScript styling portion to a mere three lines:

```
var css:StyleSheet = new StyleSheet();
css.setStyle("li", {fontStyle: "italic", color: "#A2A2A2",
leading: "-2"});
styled.styleSheet = css;
```

Following suit, let's style up a few more HTML tags:

1. Open the `Styling02.fla` file that accompanies this chapter. This file picks up where we left off. The same text fields are in place and some styling has already been applied (see the scripts layer). What's there uses the shortened code version we just looked at.
2. Next, you'll style all the `<p>` tags. Position your cursor after the `setStyle()` line and press Enter/Return to make room for the new code. Update your ActionScript so that it includes the following new code (shown in bold):

```
var css:StyleSheet = new StyleSheet();
css.setStyle("li", {fontStyle: "italic", color: "#A2A2A2",
leading: "-2"});
css.setStyle("p", {textAlign: "justify", leading: "6"});
styled.styleSheet = css;
```

3. Test your movie to see the new formatting—justified and with a taller line height—below the bullet points at the bottom right (see Figure 10-5).

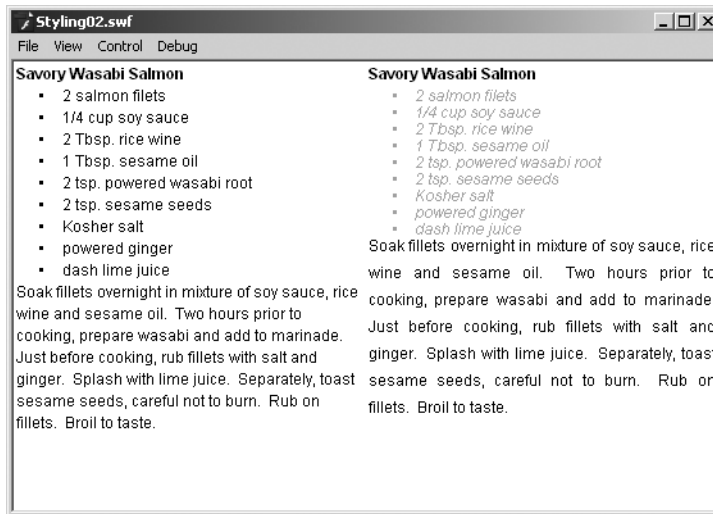


Figure 10-5. After the first style is in place, additional styles are a snap.

Say, this is encouraging! Let's keep right on going. There really isn't enough space between the bullet points and the text below them, so let's pad the bottom of the `<ul>` tag a bit. We'll also want the recipe's title to stand out more.

4. Enter the scripts layer again and update the styling ActionScript so that it includes the following new code (shown in bold):

```
var css:StyleSheet = new StyleSheet();
css.setStyle("li", {fontStyle: "italic", color: "#A2A2A2", ➤
leading: "-2"});
css.setStyle("p", {textAlign: "justify", leading: "6"});
css.setStyle("ul", {leading: "6"});
css.setStyle("b", {fontFamily: "Impact", fontSize: "14", ➤
color: "#339966"});
styled.styleSheet = css;
```

5. Test the movie to see the new styling . . . or part of it. Whoops. There's now space after the bullets—that's the additional 6 pixels of leading we wanted—but the title (the `<b>` content) hasn't changed at all! What's going on? It is a matter of selectors, which we'll deal with in the next section.
6. Feel free to save the file or close it without saving the changes.

*We're going to go off on a sizable tangent here, but don't worry. It all eventually leads back to the salmon.*

## Element selectors vs. class selectors

To this point, we've limited our view to something called **element selectors**. These refer to HTML tags—also called HTML elements—and they apply their styling, in one swoop, to all tags of a specified kind. Want to format all `<p>` content? Write a `p` element selector. Need to style a bunch of list items (`<li>`)? Write an `li` element selector. Pretty easy procedure. At the end of the previous section, though, we saw that it doesn't always work. This is one of the limitations of Flash CSS, and it's an important one to note.

In HTML documents, practically all HTML elements can be styled by way of an element selector. In Flash, the list is vastly reduced. According to the Flash CS3 documentation, the following tags comprise the meager list: `<body>`, `<p>`, `<li>`, and `<a>`. The interesting thing about this list is that `<body>` doesn't appear as one of the supported tags noted in the ActionScript 3.0 Language and Components Reference entry for `TextField.htmlText`. Then again, `<ul>` doesn't appear in that list either, and yet we saw that tag implement a leading style. So . . . is there a method to this madness? Is there some easy way to keep track of which tags can be styled with element selectors and which can't?

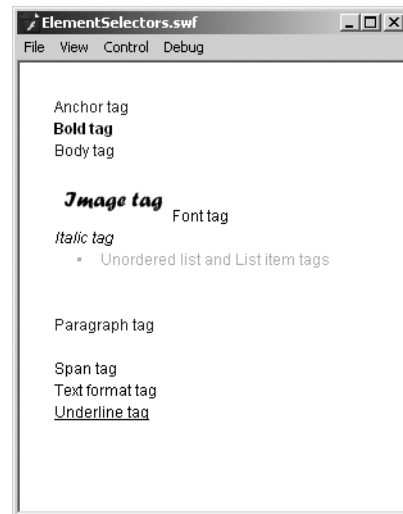
The authors spent a bit of time studying the tea leaves, and this is what we discovered: officially documented or not, the tags that support element selectors are all block elements, with the exception of the anchor tag (`<a>`). In other words, the rule of thumb is this: if the tag carries with it a built-in line break, then an element selector will do the trick. The special case is hyperlinks, which we'll cover in detail later in the chapter (hyperlinks are a special case in several ways).

For your reference, let's take a quick look at a “proof is in the pudding” sample file:

1. Open the `ElementSelectors.fla` file that accompanies this chapter. You'll find a text field with the instance name `out`. The ActionScript in the `scripts` layer shouldn't be any trouble for you by now: the `StyleSheet` class is imported, a string of HTML is created, element selectors are defined and then assigned to the `StyleSheet` instance, and finally, the HTML is supplied to the text field. Test the movie to see the result shown in Figure 10-6.

The output may not look all that interesting, but it is, because it spells out a few additional “gotchas” while verifying the block element principle.

2. Click into frame 1 of the `scripts` layer and take a look at the ActionScript in the Actions panel. Each line of HTML ends in a break tag (`<br />`), just to keep things visually neat. The `<a>` tag is not a block element, so it does not display an additional, built-in line break as with later tags; but



**Figure 10-6.** Only block elements—and one exception, anchor tags—support element selectors.



as the exception to the rule in question, it does pick up the blue color from its element selector. The `<body>` and `<p>` tag contents carry their own additional line breaks—these are block elements—and both display the expected element selector color styling. The `<ul>` and `<li>` tags' content is combined. These are also block elements and therefore display a combined pair of extra line breaks—and the expected element selector styling.

Comment out the `body` and `li` element selectors in the `ActionScript` by preceding those lines with double slashes (`//`), as shown in Figure 10-7.



```

16 var css:StyleSheet = new StyleSheet();
17 css.setStyle("a", {color: "#0000FF"});
18 css.setStyle("b", {color: "#00FF00"});
19 //css.setStyle("body", {color: "#0000FF"});
20 css.setStyle("img", {color: "#00FF00"});
21 css.setStyle("font", {color: "#0000FF"});
22 css.setStyle("i", {color: "#00FF00"});
23 css.setStyle("ul", {color: "#0000FF"});
24 //css.setStyle("li", {color: "#00FF00"});
25 css.setStyle("p", {color: "#0000FF"});
26 css.setStyle("span", {color: "#00FF00"});
27 css.setStyle("textformat", {color: "#0000FF"});
28 css.setStyle("u", {color: "#00FF00"});

```

**Figure 10-7.** Commenting out the `body` and `li` selectors leads to a line-spacing quirk and the idea of inheritance.

3. Test the movie again. It should come as no surprise that the `<body>` tag content is no longer styled. What may raise your eyebrows is that the extra line break is missing. This is a quirk involving only the `<body>` tag, and will raise its head again in the “Custom tags” section of this chapter, which follows. The other thing to notice is that the `<ul>/<li>` content has changed color. The reason for this is that a distinct color style was applied to each tag (green for `<ul>` and blue for `<li>`), and the blue won the wrestling match earlier because of a CSS concept called inheritance (covered in the “Style inheritance” section later in the chapter).
4. As a final test, uncomment the `body` element selector by removing the double slashes from that line. Instead, comment out the `p` element selector. Test the movie, and you’ll see that the `<p>` content is still blue. Why? Again, this is an example of inheritance, but in a really twisted way. Under normal circumstances, HTML documents feature most of their content inside a `<body>` tag. If a style is applied to the body, it will “trickle down” to tags inside that body if those inner tags happen to support the style properties at hand. Here in this Flash file, the `<p>` content is clearly not inside the `<body>` content, and yet some phantom inheritance seems to still hold sway. Comment out the `body` element selector one last time, and the `<p>` content finally turns black.
5. Close the file without saving the changes.

*Every development platform has its quirks, and these are a few of the ones that belong to Flash. Being aware of these, even if they aren’t burned into your neurons, might just save your hide when something about CSS styling surprises you.*

Now you've had some experience with block elements and the anchor tag, with the understanding that anchor tags still hold a bit of mystery, yet to be unfolded. Meanwhile, what remains of the other supported HTML tags? What's the opposite of a block element, and how can one be styled?

In Flash, if it is not a block HTML element, it is an inline element. All that means is that you don't carry your own line break with you. Examples include the `<b>` and `<i>` tags, which apply their own innate formatting—bold and italic, respectively—without otherwise interrupting the flow of text. As you've seen, inline elements in Flash do not support element selectors. Is there another option, then? You bet your spurs, podner. But it only goes so far.

Not to be confused with the classes discussed in Chapter 4, CSS features **class selectors**, which differ from element selectors in a significant way. Rather than apply their style to all tags of a specified type, class selectors only look for tags that have a `class` attribute whose value is set to the name of the class in question. We'll see an example of this in just a moment. In HTML documents, just about any tag can be given a `class` attribute, but this isn't the case in Flash. Actually, nothing *stops* you from giving an HTML tag such an attribute in Flash, but Flash only applies class selector styling to a few tags—and only one of those as an inline element.

Here's another “proof is in the pudding” sample file, which should make everything clear:

1. Open the `ClassSelectors.fla` file that accompanies this chapter. At first glance, this file may look identical to `ElementSelectors.fla`, but click into frame 1 of the scripts layer to lay eyes on a different hunk of code. To wit, every HTML tag now has a `class` attribute, set either to `blue` or `green`, and the number of selectors has been reduced to two—the selfsame `blue` and `green` styles. Now, how can you tell that these are class selectors and not element selectors? The giveaway, which is easy to miss if you aren't looking for it, is the dot (`.`) in front of the style names (see Figure 10-8).

```

7  str += "<img class='green' src='sampleImage' /><br />";
8  str += "<font class='blue'>Font tag</font><br />";
9  str += "<i class='green'>Italic tag</i><br />";
10 str += "<ul class='blue'><li class='green'>Unordered list and List item tags</li></ul><br />";
11 str += "<p class='blue'>Paragraph tag</p><br />";
12 str += "<span class='green'>Span tag</span><br />";
13 str += "<textformat class='blue'>Text format tag</textformat><br />";
14 str += "<u class='green'>Underline tag</u>";
15
16 var css:StyleSheet = new StyleSheet();
17 css.setStyle(".blue", {color: "#0000FF"});
18 css.setStyle(".green", {color: "#00FF00"});
19
20 output.styleSheet = css;
21 output.htmlText = str;

```

**Figure 10-8.** Class selectors are much more selective than element selectors. You can spot them by their dot prefixes.

Those dots change everything, because at this point, CSS doesn't care what tag it's dealing with—it only cares if that tag has a `class` attribute set to `blue`, `green`, or whatever the style's name is.

*Be careful where you put your dots! They only belong in the `setStyle()` method, never in the `class` attribute of any tag.*

2. Test the movie to see the result. Remember, in the “real world” outside of Flash, every one of these tags would be affected by the relevant style. In the SWF, only the following tags do anything: `<a>`, `<li>`, `<p>`, and `<span>`. Unfortunately, we haven’t found as neat a way to memorize this list as the other, but if you can remember the block elements that go with element selectors, you need only swap the `<body>` tag for the `<span>` tag and drop `<ul>` to know the block and inline elements that go with class selectors. (Yeah, we agree, it’s not especially intuitive.)
3. For the sake of completeness, comment out the `.green` class selector and test the movie to verify. The `<ul>/<li>` content turns black because class selectors don’t apply to `<ul>` tags in Flash.
4. Close the movie without saving the changes.

## Custom tags

Ready to head back to the wasabi salmon? When we abandoned it to venture out on our educational tangent, most of our styling had taken—all of it had, in fact, except the `<b>` content, and now we know why. The `<b>` tag is not a block element, which means it simply doesn’t support element selectors. In any case, element selectors affect all tags of a given type, and for the sake of illustration, let’s say we only want this recipe’s title to stand out, not all content that happens to be set in bold. An obvious solution, then, based on your current knowledge, would be to swap the `<b>` tag for something that supports class selectors.

1. Open the `Styling03.fla` file to see an example of just that approach. The key changes in the ActionScript from `Styling01.fla` are shown in bold in the following code:

```
var str:String = "";
str += "<p class='heading'>Savory Wasabi Salmon</p>";
str += "<ul>";
...
css.setStyle("ul", {leading: "6"});
css.setStyle(".heading", {fontFamily: "Impact", fontSize: "14",
color: "#339966"});
styled.styleSheet = css;
```

This mix-and-match approach is perfectly valid. In fact, it’s a good basic methodology: use element selectors to sweep through the styling for most tags, and then cover the exceptions with class selectors. Or, you can use custom tags, which provide a kind of hybrid mechanism. They save you from having to type `class='someStyleName'` throughout your HTML content, and the best part is, you can use familiar, genuine HTML tags from the “real world,” if you like (think along the lines of `<h1>`, `<h2>`, `<strong>`, etc.). Flash happily accepts these as “custom” tags, because in its skimpy repertoire, they are.

2. Open the Styling04.fla file to see a custom tag in action. Once again, this file is virtually identical to the previous one, except for the parts shown in bold:

```
var str:String = "";
str += "<strong>Savory Wasabi Salmon</strong>";
str += "<ul>";
...
css.setStyle("ul", {leading: "6"});
css.setStyle("strong", {fontFamily: "Impact", fontSize: "14",
color: "#339966"});
styled.styleSheet = css;
```

Note the *absence* of a dot preceding the strong element selector: this is *not* a class selector! If you put 50 <strong> tags full of content in your SWF, all 50 occurrences will pick up the style from this `setStyle()` method. That said—and we can't stress this enough—please understand that this is not a magical, undocumented way to squeeze additional tags out of Flash's limited HTML support. Flash has no idea what a <strong> tag is, much less that most browsers treat it like a <b> tag. This is nothing more than a convenient hook for CSS, an excuse to dodge class selectors if you happen not to like them. In fact, to prove it, and to reveal a limitation of the custom tag approach, proceed to step 3.

3. Replace the <strong> tag in the highlighted ActionScript with the completely made-up <citrus> tag. There is no such tag in any of the W3C specifications (we looked). Your code will only change in three places:

```
var str:String = "";
str += "<citrus>Savory Wasabi Salmon</citrus>";
str += "<ul>";
...
css.setStyle("ul", {leading: "6"});
css.setStyle("citrus", {fontFamily: "Impact", fontSize: "14",
color: "#339966"});
styled.styleSheet = css;
```

4. In addition, find the word "lime" in the bulleted list and wrap it with this new <citrus> tag:

```
str += "<li>powered ginger</li>";
str += "<li>dash <citrus>lime</citrus> juice</li>";
str += "</ul>";
```

5. Now test the movie and take a look. You should see the styling shown in Figure 10-9.

Danger, Will Robinson! What do we learn from the broken dash lime juice line? A valuable lesson, that's what. The recipe's title is fine, but that's because it stands on its own. The lime line breaks because custom tags become block elements when styled. In this case, the word juice has even been pushed past the extra line height given earlier to the <ul> tag.

**Savory Wasabi Salmon**

- 2 salmon filets
- 1/4 cup soy sauce
- 2 Tbsp. rice wine
- 1 Tbsp. sesame oil
- 2 tsp. powdered wasabi root
- 2 tsp. sesame seeds
- Kosher salt
- powdered ginger
- dash *lime*
- juice

Soak filets overnight in mixture of soy sauce, rice wine, and sesame oil. Two hours before cooking, prepare wasabi and add to mixture. Just before cooking rub filets with

**Figure 10-9.** Whoops, something isn't right with the lime.

We've spent the last several miles mulling over some pretty arcane rules and even hazier exceptions to them. CSS was supposed to be easier in Flash, right? If your head is spinning, take a sip from the canteen and rest for a spell. While we wait, one of the authors will hum an old, lonely cowboy tune. The lyrics go something like this: "To get the biggest bang for your buck, use element selectors first, then custom tags for headings and other short or specific blocks, and finally class selectors for special cases." (Hey, no one said it had to rhyme, and the melody really is pretty.)

## Style inheritance

In moving from Object instances to the object shortcut characters ({}), earlier in the chapter, we saw one way to trim CSS into a more compact form. There's another way to compact things even further, but it's more conceptual than syntactical. The concept is called **inheritance**, and it basically means that styles applied "up the creek" tend to eventually flow down to lower waters. A concrete example will spell this out quicker than an explanation.

1. Open the Inheritance.fla file that accompanies this chapter. You'll see a text field with the instance name output.
2. Click into frame 1 of the scripts layer to view the ActionScript. As with the other samples in this chapter, the code begins by building an HTML string. In this case, the structure of the HTML tags is important. Stripping out the text content, the tag hierarchy looks structurally like this:

```
<body>
  <p></p>
  <outer>
    <mid>
      <inner><span class='big'></span></inner>
    </mid>
  </outer>
</body>
```

Styling is applied to the <body> tag, which sets its font to Courier. The tags nested inside this tag, <p> through <mid>, gain the same font thanks to inheritance. The custom <inner> tag would also inherit Courier, except that this particular tag bucks the trend by specifying

its own font, Arial, which overrides the inherited Courier and sets up its own new inheritance. Note that the `<span>` tag within `<inner>` displays Arial, like its parent, as shown in Figure 10-10.

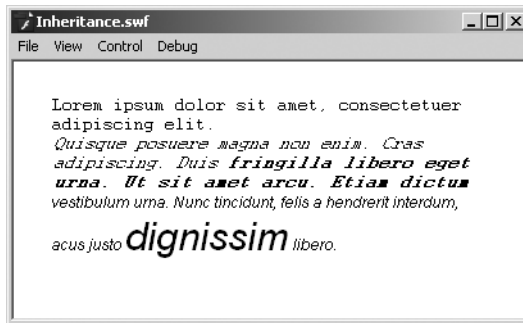


Figure 10-10. CSS inheritance at work

This sort of procedure can get fairly sophisticated. For example, the custom `<outer>` tag adds italics to the mix.

```
css.setStyle("outer", {fontStyle: "italic"});
```

Because the flow goes downhill, `<mid>`, `<inner>`, and `<span>` inherit not only the font of `<outer>`'s parent, but also its italics, while sibling tags (`<p>`) and parent tags (`<body>`) do not. And honestly, that makes good sense.

In the same vein, the custom `<mid>` tag introduces bold:

```
css.setStyle("mid", {fontWeight: "bold"});
```

Now, unopposed, `<inner>` and `<span>` would inherit that bold styling as well, but `<inner>` purposefully overrides that by setting `fontWeight` to `normal` in its own element selector:

```
css.setStyle("inner", {fontFamily: "Arial", fontWeight: "normal"});
```

In turn, this causes `<span>` to inherit the override, as it too ignores the bold. Note, however, that `<span>` does inherit the italics, which were not overridden by a parent tag.

Use this inheritance phenomenon to your advantage. It saves you keystrokes, for one—there's absolutely no need to specify font families for whole groups of related tags, for example—and in addition, it gives you the opportunity to make sweeping changes from relatively few locations.

## Styling hyperlinks

Anchor tags are fun to style because of something called pseudo-classes. In CSS talk, a **pseudo-class** corresponds to various possible states of an HTML element and is indicated by a colon (`:`) prefix. In Flash, the only supported pseudo-classes are associated with the anchor tag (`<a>`) and correspond to the following states: `:link` (an anchor tag that specifically

contains an href attribute), :hover (triggered by a mouse rollover), and :active (triggered by a mouse click). The long and short of this is that you have the tools you need to create different anchor tag styles that update as the mouse moves and clicks your hyperlinks. Note that Flash does not support the :visited pseudo-class, which in normal CSS indicates that a hyperlink has already been clicked.

Think of pseudo-classes as a second tier of styles, not separated by hierarchy, as shown in the “Style inheritance” section, but separated by time or events.

1. Open the Hyperlinks.fla file to see an example in action. The ActionScript begins, as always, by establishing an HTML string:

```
var str:String = "";
str += "<ul>";
str += "<li><a href='http://www.apress.com/'>Hyperlink 1</a></li>";
str += "<li><a href='event:someFunction'>Hyperlink 2</a></li>";
str += "<li><a href='http://www.friendsofed.com/'>
Hyperlink 3</a></li>";
str += "</ul>";
```

These anchor tags happen to be nested within list items, but they needn't be. The important part is that anchor tags exist that have href attributes actively in use. In these next three lines, the element selectors provide a style of anchor tags in any state—that's the first highlighted line—followed by distinct styles for the :hover and :active pseudo-classes.

```
var css:StyleSheet = new StyleSheet();
css.setStyle("li", {leading: "12"});
css.setStyle("a", {textDecoration: "none"});
css.setStyle("a:hover", {fontStyle: "italic"});
css.setStyle("a:active", {fontStyle: "italic",
text-decoration: "underline", color: "#FF0000"});
output.styleSheet = css;
```

2. Test this movie to verify that hovering over hyperlinks puts them temporarily in italics, and that clicking additionally displays an underline and new color. Note that the italic style isn't inherited by :active because :active is not a child of :hover; they have a sibling relationship.

What if you'd like more than one style for your hyperlinks? Answer: Use a class selector.

3. Open HyperlinksVaried.fla for an example. First, here's the new HTML:

```
var str:String = "";
str += "<ul>";
str += "<li><a href='http://www.apress.com/'>Hyperlink 1</a></li>";
str += "<li><a href='event:someFunction'>Hyperlink 2</a></li>";
str += "<li><a href='http://www.friendsofed.com/'>
Hyperlink 3</a></li>";
str += "</ul>";
str += "<ul>";
str += "<li><a class='oddball' href='http://www.apress.com/'>
Hyperlink 4</a></li>";
```

```
str += "<li><a class='oddball' href='event:someFunction'>▶
Hyperlink 5</a></li>";
str += "<li><a class='oddball' href='http://www.friendsofed.com/'>▶
Hyperlink 6</a></li>";
str += "</ul>";
```

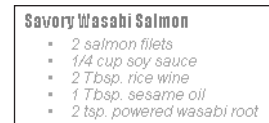
Unfortunately, it isn't possible to create unique pseudo-classes for the oddball-specific anchor tags, but the following new class selector at least separates the new batch of hyperlinks in their default state:

```
var css:StyleSheet = new StyleSheet();
css.setStyle("li", {leading: "12"});
css.setStyle("a", {textDecoration: "none"});
css.setStyle("a:hover", {fontStyle: "italic"});
css.setStyle("a:active", {fontStyle: "italic"}, ▶
textDecoration: "underline", color: "#FF0000"});
css.setStyle(".oddball", {color: "#00FF00"});
output.styleSheet = css;
```

4. Close the open files, and let's now look at embedding fonts.

## Embedded fonts

Before we take what we've learned and nudge it all toward an external CSS file, let's make a quick stop at the Last Chance Saloon to talk about embedded fonts. CSS in Flash requires HTML, which in turn requires a dynamic text field. As you learned in Chapter 6, only static text fields embed font outlines by default. What this means is that unless you purposefully embed your fonts—and the choice is yours—CSS-enhanced SWFs tend to have that jagged, non-font-embedded look (shown in Figure 10-11).



**Figure 10-11.** Text can look a bit choppy if fonts aren't embedded.

Font symbols were introduced in Chapter 6, but there's a new twist in how they're used with CSS. To recap, the font embedding process is as follows:

1. Add a font symbol to the library and associate it with the desired font on your system.
2. Enable the font symbol's linkage by exporting the symbol for ActionScript.
3. Set the text field's `embedFonts` property to `true`.
4. Reference the font symbol's linkage class name—this has nothing to do with CSS class selectors; it's just coincidentally the same term—in place of the font's actual name in any relevant styles.

The trouble is that the process doesn't work as advertised. Here's how to fix it.



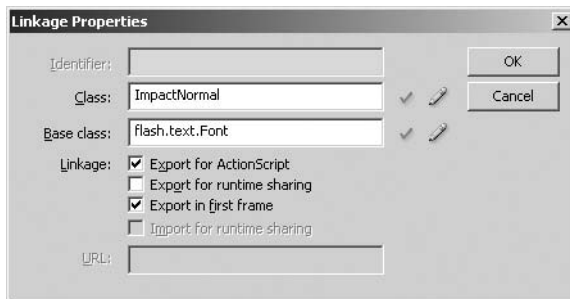
1. Open the `StylingEmbeddedFonts01.fla` file that accompanies this chapter. Test the movie and you'll see a blank SWF. This is odd because with the inclusion of the ActionScript, this movie should presumably work just fine.
2. Click into frame 1 of the scripts layer and note the following pertinent lines of code:

```
css.setStyle("strong", {fontFamily: "ImpactNormal", fontSize: 14, color: "#339966"});
```

```
output.embedFonts = true;
```

In the first line, notice the reference to a font named `ImpactNormal`. This name refers to the linkage class name of a font symbol in the library, which we'll see in just a moment. The second line turns on font embedding for the output text field instance.

3. Right-click (PC) or Ctrl-click (Mac) the `ImpactNormal` font symbol in the library. Choose `Linkage` from the context menu and verify that the font is exported for ActionScript (see Figure 10-12), and has a `Class` name of `ImpactNormal`.



**Figure 10-12.** The font symbol has been given what at first glance appears to be valid linkage properties.

Flash simply doesn't like something in this file as it stands. The trouble—and this is a bona fide bug—is due to the arbitrary class name given to the font symbol. In the Chapter 6 sample file, this didn't matter because the text field's font was specified via the Property inspector rather than ActionScript. In an ideal world, the linkage class name provides the necessary “link” to this library asset, but in this case, we'll have to roll with Flash's whim.

4. Update the ActionScript to reference the font's actual name, as shown in the font list of the Property inspector:

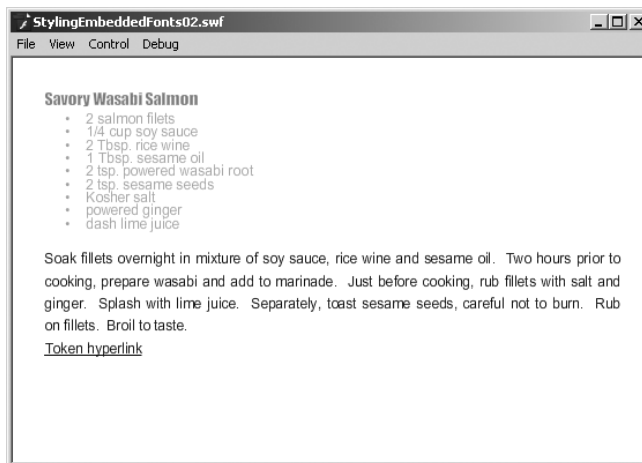
```
css.setStyle("strong", {fontFamily: "Impact", fontSize: "14", color: "#339966"});
```

Note that it makes no difference what the font symbol's library name or linkage class is. In this case, the library name is still `ImpactNormal`. Test your movie, and the text shown in Figure 10-13 magically appears. Hey, that's an improvement! Not only does the recipe's title show up, but the lettering is smooth. Now for the rest of the text.



**Figure 10-13.** The ImpactNormal font is now showing, without the jaggies.

5. Even though it isn't explicitly done, this text field is being asked to display more than one font. Sure, the only font mentioned in the `css` instance is Impact, but the Property inspector happens to show Arial for the text field, which means the default font is Arial. The text field's `embedFonts` property has been set to `true`, which means only embedded fonts can be displayed. Two fonts are being summoned, so two fonts need to be embedded.
6. Using the technique described in Chapter 6, add an Arial font symbol to your library and name the symbol whatever you like. Make sure to export it for ActionScript. To prove that neither the library name nor linkage class name matters—because ActionScript does not reference this second font directly—compare your work with `StylingEmbeddedFonts02.fla`, whose embedded Arial is named `HornyToads` in the library. Test your movie (see Figure 10-14). Bingo! All of the text content shows.



**Figure 10-14.** All the text is accounted for, and none of it suffers from the jaggies.

## Loading external CSS

If we had to pick our favorite aspect of CSS in Flash, it would undoubtedly be the fact that CSS styling can be loaded from an external file. The existence of this feature brings the concept of separating style from informational content to its logical conclusion. Given all you've learned so far in this chapter, you'll be happy to find that loading external CSS is a piece of cake. There's really only one snare to be aware of: some of the style properties we showed you in the beginning of the chapter are spelled just a tad differently when they appear in an external file. Single-word properties, such as `color` or `display`, are identical. Multi-word properties, such as `fontFamily` or `fontSize`, are split into hyphenated parts: `font-family`, `font-size`, and so on.

1. Open the `StylingExternal.fla` file that accompanies this chapter. You'll see a single text file with the instance name `output`. Click into frame 1 of the `scripts` layer and take a quick look at the `ActionScript`. By now, the HTML portion will be old hat—it's a resurrection of the salmon recipe one last time, with a token hyperlink at the bottom. The new stuff is just below:

```
var css:StyleSheet = new StyleSheet();
var loader:URLLoader = new URLLoader();
loader.load(new URLRequest("styles.css"));
loader.addEventListener(
    Event.COMPLETE,
    function(evt:Event):void {
        css.parseCSS(URLLoader(evt.target).data);
        output.styleSheet = css;
        output.htmlText = str;
    }
);
```

The first line creates our familiar `StyleSheet` instance. The next . . . aha, that's the nifty one! A variable, `loader`, is declared and set to an instance of the `URLLoader` class. This differs from the `Loader` class (which you'll see in Chapter 13), which loads images or SWFs. What makes `URLLoader` different is that it not only loads files, but actually reads them, which is essential when the goal is to sift through external CSS.

The `URLLoader.load()` method is invoked on the `loader` instance with the expression `new URLRequest("styles.css")` as the parameter. This is a shortcut way of converting the raw string `"styles.css"` into the format needed. Finally, the `Event.COMPLETE` event is handled with a function that performs three straightforward tasks: parse the loaded CSS, set the text field's `styleSheet` property to the `css` instance, and set its `htmlText` property to the prepared HTML string. You already know how the last two work, so let's pick apart the first line of this function. The `StyleSheet.parseCSS()` method takes a single parameter, which in this case is the expression `URLLoader(evt.target).data`. That may look like a mouthful, but it's nothing more than a compact way of getting at the CSS styles themselves. The `evt.target` part refers to the file loaded by `loader`, and is wrapped in a `URLLoader()` function to make it presentable. Consider that its calling card, or that `URLLoader()` has run a comb through `evt.target`'s hair. The `data` part refers to the data inside said file.

2. Open the `styles.css` file in Dreamweaver CS3 or any simple text editor such as NotePad on the PC or TextEdit on the Mac.

*CSS files, though they serve a special styling purpose, are really just text files with a .css file extension).*

The contents should be easily recognizable to you:

```
li {
  font-style: italic; color: #A2A2A2; leading: -2;
}

p {
  text-align: justify; leading: 6;
}

ul {
  leading: 6;
}

strong {
  font-family: Impact; font-size: 14; color: #339966;
}

a {
  font-family: Courier; font-weight: bold;
}

a:hover {
  color: #FF00FF;
}
```

Besides the hyphenated style properties and a few minor syntactical differences, these selectors represent the same styling approach you've seen throughout this chapter. The syntax differences to look out for are as follows: in this version, neither property names nor values are wrapped in quotation marks, as they are in ActionScript (e.g., `font-style: italic` instead of `fontStyle: "italic"`); and properties are separated by semicolons rather than commas, like this:

```
li { fontStyle: italic; color: #A2A2A2; leading: -2; }
```

instead of this:

```
css.setStyle("li", { fontStyle: "italic", color: "#A2A2A2",
leading: "-2" });
```

By the way, thanks to this punctuation, you have some leeway in how you arrange the properties, both in ActionScript and the CSS file; single-line or spread over several lines—it doesn't matter. As long as the required parts are present, Flash can figure out what you mean. So go ahead and suit your fancy. For example, this

```
li { fontStyle: italic, color: #A2A2A2, leading: -2 }
```

is the same as

```
li {
  fontStyle: italic;
  color: #A2A2A2;
  leading: -2;
}
```

3. And now we've arrived at the punch line. Test the movie to generate a SWF file, which should look something like Figure 10-15. Now close Flash. That's right, shut down the application. The rest is a matter between you, a SWF, and a CSS file.

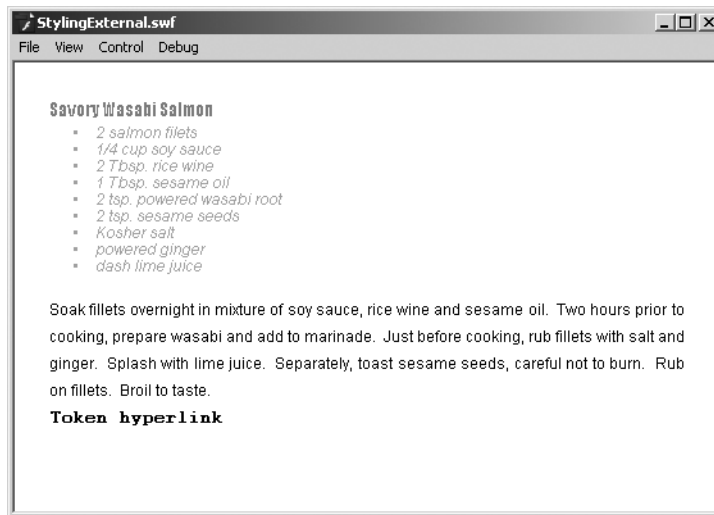


Figure 10-15. CSS styles pulled from an external CSS file

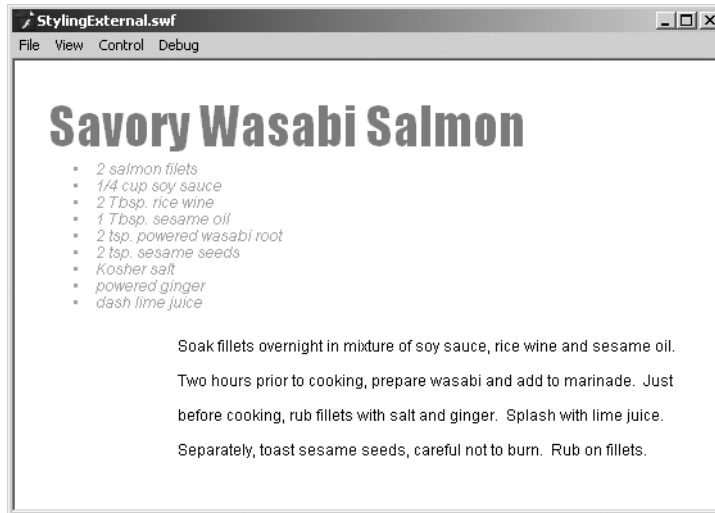
10

4. Double-click your newly created `StylingExternal.swf` file to give it one last look. This is a bit like making sure the magician has nothing up either sleeve. Now change a few of the style properties in `styles.css`. As a suggestion, update the `p` and `strong` styles as follows:

```
p {
  margin-left: 100px; leading: 12px;
}

strong {
  font-family: Impact; font-size: 40px; color: #339966;
}
```

5. When you make your changes, save the document.
6. Close `StylingExternal.swf` and double-click it again to launch the SWF. Notice how you can change the look of a movie without having to change the code (see Figure 10-16).



**Figure 10-16.** Look, Ma—style changes without re-creating the SWF!

Without republishing the SWF, you've updated its formatting! That's no small feat. Hey, did you catch something missing? What happened to that hyperlink? The increased leading in the p style has pushed it off the stage! In fact, the phrase Broil to taste has also been shoved aside. No problem. Just readjust the leading property or decrease the strong style's font-size property until everything fits. This sort of tweaking is what CSS was made for.

*At the time this chapter was written, one of the authors had recently completed a Flash-based training presentation for a U.S. government agency that featured over 250 slides. At one point, the author needed to change the color of one of the heading styles to a slightly different orange. Guess who was happy that day because he had used CSS in his SWFs?*

## What you've learned

Apart from learning what to serve the authors at your next barbecue, you have discovered that the CSS techniques widely employed in the HTML universe are just as applicable to your Flash efforts. As you moved through this chapter, you learned the following:

- How to apply CSS styling through ActionScript
- The difference between an element selector and a class selector
- That you can create your own custom tags

- How to use the concept of inheritance to your advantage
- That there is a pesky bug—and a workaround—when it comes to using embedded fonts in the library
- How to use an external CSS style sheet in Flash

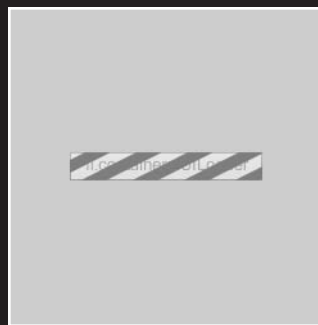
If there is one major theme running through this chapter, it is this: your CSS skills put a powerful tool in your arsenal. Speaking of powerful tools, XML's relationship with Flash just got a power boost. Turn the page to find out.





# 11 DYNAMIC DATA (XML) AND FLASH

```
Output x
<film title="Strong to the Fir
29, 1934" runningTime="7">
  <cast>
    <character>Popeye</charact
    <character>Olive Oyl</char
    <character>Orphan Childrer
  </cast>
</film>
<film title="Little Swee' Pea'
25, 1936" runningTime="7">
  <cast>
    <character>Popeye</charact
    <character>Olive Oyl</char
    <character>Swee' Pea</char
```



Flash is a social creature. Not only does it rub elbows with HTML, coexisting happily with text, JavaScript, images, audio, video, Cascading Style Sheets (CSS), and more, but it can also reach out past its own SWF boundaries to collaborate with data hosted on a server.

In the hands of an experienced programmer, Flash can interact with database applications by way of the `URLLoader` and `URLVariables` classes, perform web service and Flash remot-ing calls, and even slap a secret handshake with Ajax, thanks to the `ExternalInterface` class. All this from a browser plug-in that began its life as a way to improve on animated GIFs! It's easy to see why Flash has become a widespread phenomenon, and its versatility makes equally social creatures of the countless designers and developers who end up warming their diverse mitts around the same campfire because of it.

This book isn't here to make programmers out of artists. We don't have the page count to delve into most of the concepts just mentioned, but we are going to introduce you to a markup language called XML that, with a bit of help from ActionScript, can make your SWFs dynamic.

What we'll cover in this chapter:

- An overview of XML
- How to retrieve and filter XML data using E4X syntax
- How to build a dynamic slideshow driven by XML

Files used in this chapter:

- `LoadXML.fla` (Chapter11/ExerciseFiles\_CH01/Exercise/LoadXML.fla)
- `LoadXML-E4XFiltering.fla` (Chapter11/ExerciseFiles\_CH01/Exercise/LoadXML-E4XFiltering.fla)
- `LoadXML-E4XBonusRound.fla` (Chapter11/ExerciseFiles\_CH01/Exercise/LoadXML-E4XBonusRound.fla)
- `popeye.xml` (Chapter11/ExerciseFiles\_CH01/Exercise/popeye.xml)
- `slideshow.xml` (Chapter11/ExerciseFiles\_CH01/Exercise/slideshow.xml)
- `Slideshow.fla` (Chapter11/ExerciseFiles\_CH01/Exercise/Slideshow.fla)
- `SlideshowXML.fla` (Chapter11/ExerciseFiles\_CH01/Exercise/SlideshowXML.fla)
- `geocache01.jpg` (Chapter11/ExerciseFiles\_CH01/Exercise/geocache01.jpg)
- `geocache02.jpg` (Chapter11/ExerciseFiles\_CH01/Exercise/geocache02.jpg)
- `geocache03.jpg` (Chapter11/ExerciseFiles\_CH01/Exercise/geocache03.jpg)
- `geocache04.jpg` (Chapter11/ExerciseFiles\_CH01/Exercise/geocache04.jpg)
- `geocache05.jpg` (Chapter11/ExerciseFiles\_CH01/Exercise/geocache05.jpg)
- `geocache06.jpg` (Chapter11/ExerciseFiles\_CH01/Exercise/geocache06.jpg)

## The power of XML

To start, let's take a quick survey of what XML is. If you haven't already worked with XML, well . . . we bet our next single malt Scotch you've at least heard of it. The letters stand for eXtensible Markup Language, and extensibility is almost certainly the reason XML has become a towering champ in data communication. Countless markup languages and file formats are based on XML, including SMIL, RSS, XAML, MXML, RDF, WAP, SVG, SOAP, WSDL, OpenDocument, XHTML, and truly more than would fit on this page. We'll leave the letter combinations to a Scrabble master.

"That's fine and dandy," you might be saying, "But guys—*what is XML?*" Fair enough. The remarkable thing about this language is that it can basically be whatever you want it to, provided you stick by its rules. The main purpose of XML is to expedite the sharing of data. In fact, XML is so flexible that newcomers are often baffled on where to even begin. On paper—or rather, on the screen—XML looks a lot like HTML, except instead of predetermined tags and attributes, you organize your content into descriptive tags of your own design. HTML formats data, and XML describes data. The combination of familiar, hierarchical format and completely custom tags generally makes XML content easy to read, both to computers and humans. By separating your data from the movie, you give yourself the opportunity to change content from the outside, affecting SWFs without having to republish them.

## Writing XML

Let's say you've been tasked with organizing a collection of vintage Popeye cartoons. You've got five short films on your list: *I Yam What I Yam*; *Strong to the Finich*; *Beware of Barnacle Bill*; *Vim, Vigor, and Vitaliky*; and *Little Swee' Pea*. Each cartoon has its own release date, running time, and cast of characters. Where to begin? Let's take a look.

Every XML document must have at least one tag, which constitutes its **root element**. The root element should describe the document's contents. In this case, we're dealing with cartoons, so let's make that our root:

```
<cartoons></cartoons>
```

*Looks kinda crazy, doesn't it? Almost like you're getting away with something. After all, a technology that facilitates stock market transactions and configures user preferences should be . . . somehow . . . serious—right? Hey, if you don't think Popeye is seriously funny, you haven't spent enough time with Descartes. I think, therefore I yam. (Thank you! We'll be here all week.)*

The rest of our elements will layer themselves inside this first one. Every cartoon is its own film, so we'll add five `<film>` elements:

```

<cartoons>
  <film></film>
  <film></film>
  <film></film>
  <film></film>
  <film></film>
</cartoons>

```

Each film has a title, so the next step seems obvious enough:

```

<cartoons>
  <film>
    <title>I Yam What I Yam</title>
  </film>
  <film>
    <title>Strong to the Finich</title>
  </film>
  <film>
    <title>Beware of Barnacle Bill</title>
  </film>
  <film>
    <title>Vim, Vigor, and Vitaliky</title>
  </film>
  <film>
    <title>Little Swee' Pea</title>
  </film>
</cartoons>

```

You get the idea. It doesn't take much effort to connect the rest of the dots. An excerpt of the completed document might look something like this:

```

<cartoons>
  <film>
    <title>I Yam What I Yam</title>
    <releaseDate>September 29, 1933</releaseDate>
    <runningTime>6 min</runningTime>
  </film>
  . . .
</cartoons>

```

Actually, that's isn't complete after all, is it? The cast of characters is missing. For that, another tier of elements is in order:

```

<cartoons>
  <film>
    <title>I Yam What I Yam</title>
    <releaseDate>September 29, 1933</releaseDate>
    <runningTime>6 min</runningTime>

```

```

    <cast>
      <character>Popeye</character>
      <character>Olive Oyl</character>
      <character>Wimpy</character>
      <character>Big Chief</character>
    </cast>
  </film>
  . . .
</cartoons>

```

That would certainly do it. The tag names are meaningful, which is handy when it comes time to retrieve the data. The nested structure organizes each concept into a hierarchy that makes sense: characters belong to a cast, which is one aspect, along with title, release date, and running time, of a film. Nicely done—though in a sizable collection, this particular arrangement might come across as bulky. Is there a way to trim it down? Sure thing. Remember, XML allows you to create your own attributes, so you have the option of rearranging the furniture along these lines:

```

<cartoons>
  <film title="I Yam What I Yam" releaseDate="September 29, 1933" ▶
    runningTime="6 min">
    <cast>
      <character name="Popeye" />
      <character name="Olive Oyl" />
      <character name="Wimpy" />
      <character name="Big Chief" />
    </cast>
  </film>
  . . .
</cartoons>

```

The exact same information is conveyed. The only difference now is how it would be retrieved, which you'll see in the "E4X" section of this chapter. Which approach is better? Honestly, the choice is yours. It's not so much a question of better as it is what best matches your sense of orderliness. Ironically, this open-ended quality, which is one of XML's strongest assets, is the very thing that seems to scare off so many XML freshmen.

*Working with and structuring an XML document follows the first principle of web development: "Nobody cares how you did it. They just care that it works." Find what works best for you, because in the final analysis, your client will never pick up the phone and say, "Dude, that was one sweetly structured XML document you put together."*

Folks, this is a bit like an artistic ceramics class. As long as you're careful around the kiln, nobody can tell you whose vase is art and whose isn't. Just work the clay between your fingers, let a number of shapes mull around your noggin, and then form what you've got into a structure that appeals to you. While you're at it, keep a few rules in mind:

- If you open a tag, close it (<tag></tag>).
- If a tag doesn't come in two parts—that is, if it only contains attributes, or nothing at all—make sure it closes itself (<tag />).
- Close nested tags in reciprocating order (<a><b><c /></b></a> is correct; <a><b><c /></a></b> lights your pants on fire).
- Wrap attribute values in quotation marks (<tag done="right" />, <tag done=wrong />).

The Popeye example just shown would be saved as a simple text file with the .xml file extension—for example, popeye.xml—and that's that.

Now that our introductions have been made, let's get social.

*Feel free to use a text editor such as NotePad on the PC or TextEdit on the Mac. Just be sure you add the .xml extension to the file's name. If you have Dreamweaver CS3, that's even better, because it will offer code completion suggestions to speed up your workflow.*

## Loading an XML file

The ActionScript required for loading an XML document isn't especially involved. You'll need an instance of the XML and URLLoader classes, and then, of course, an XML document. In our case, the document will always be an actual XML file, although XML documents can be built from scratch with ActionScript.

Open the LoadXML.fla file that accompanies this chapter. Click into frame 1 of the scripts layer and open the Actions panel to see the following code:

```
var xml:XML = new XML();
var loader:URLLoader = new URLLoader();
loader.load(new URLRequest("popeye.xml"));
loader.addEventListener(
    Event.COMPLETE,
    function(evt:Event):void {
        xml = XML(evt.target.data);
        trace(xml);
    }
);
```

Let's break it down. The first two lines declare a pair of variables, xml and loader, that point to instances of the XML and URLLoader classes, respectively.

Line 3 then invokes the URLLoader.load() method on the loader instance, specifying the expression new URLRequest("popeye.xml") as the parameter. In your own projects, you will of course replace popeye.xml with the name of your own XML files. This procedure

starts the load process, but the data isn't available until the XML document has fully arrived from the server. For this reason, the final block attaches an `Event.COMPLETE` handler to the loader instance.

In response, the handler function sets the `xml` instance to the data property of the target of the `evt` parameter passed into the function. That's a mouthful, but it basically means that the `xml` instance is associated with the actual text content of the `popeye.xml` file. At this point, the text file's XML tags become a "living XML object" in the SWF, accessible via that `xml` variable. To prove it with this sample, a `trace()` function sends the full set of Popeye elements to the Output panel. Test the movie and compare the Output panel's content to the `popeye.xml` file itself, which you can open with Dreamweaver CS3 or any simple text editor.

The preceding sample code will serve as the basis for all loading for the rest of the chapter. It's really that simple. Even better, ActionScript 3.0 makes it just as easy to actually use XML, so let's jump right in.

## E4X

In ActionScript 2.0, interacting with an XML class instance was a bit like groping in the dark for matching socks (and it's hard enough to sort laundry with the lights on!). The reason for this is because of the way XML elements used to be accessed once loaded, which wasn't by the practical tag names we supplied earlier in the chapter.

Until Flash CS3 arrived on the scene, XML in Flash was not up there on the list of "cool things I really need to do." In fact, many designers and developers (one of the authors among them) regarded the use of XML as being similar to the long walk to the principal's office in grade school. The walk was so painful because you just knew your parents were about to be involved and a world of grief was to be opened on you. Readers familiar with Flash XML prior to CS3 will doubtless groan to remember obtuse expressions such as, for example, `xmlInstance.firstChild.firstChild.childNodes[2]`. Flash developers used properties like `firstChild` and `childNodes` because they had to, not because it was fun. Then there was the "now defunct" `XMLConnector` component, which complicated our lives more than simplifying the process. ActionScript 3.0 does away with this groping, thanks to something called E4X. Hey, hear that whooshing noise? That's the sound of everyone dashing to meet this new kid in the neighborhood with the really cool bike.

What is E4X, and what makes it so good? Seemingly named after an extra from a George Lucas feature, those three characters form a cutesy abbreviation of ECMAScript for XML, a specification that provides a completely new, simplified way to access data in an XML instance. In E4X, elements are referenced by the name you give them. Paths to nested elements and attributes are easily expressed by a neatly compact syntax of dots (`.`) and *at* symbols (`@`), which closely matches the dot-notation pathing you're already familiar with from the Twinkie example in Chapter 4.

Here's how it works:

1. If you haven't already, open the LoadXML.fla file that accompanies this chapter. Click into frame 1 of the scripts layer and open the Actions panel to reveal the ActionScript. The `trace()` function at line 8 is about to illustrate a number of dynamic E4X features.
2. Testing the movie as it stands puts the full XML document's contents into the Output panel. So far, so good; but if you don't care about the root element, `<cartoons>`, and simply want to see the `<film>` elements, update the `trace()` line to read `trace(xml.film)`; Once you do that, test the movie again. This time, the `<cartoons>` tag doesn't show, because you're only accessing its children.

To view `<film>` elements individually, use the array access operator, `[]`, and specify the desired element, starting your count with 0 (zero):

```
trace(xml.film[0]);
// displays the first <film> element (I Yam What I Yam)
// and its children

trace(xml.film[1]);
// displays the second <film> element (Strong to the Finch)
// and its children
```

3. Now, what about attributes? To see those, just precede an attribute's name with the `@` symbol as part of your dot-notation path reference. For example, if you want to see the `title` attribute of the first `<film>` element, type the following:

```
trace(xml.film[0].@title);
```

To see the second `<film>` element's title, substitute 0 with 1; to see the third, substitute 1 with 2; and so on. Based on this pattern, the last element's title attribute would be `xml.film[4].@title`—but we only know to use the number 4 because we're aware how many `<film>` elements there are. What if we don't know?

In this case, it helps to understand exactly what you're getting back from these E4X results. What you're getting are instances of the `XMLElementList` class, which means you can invoke any of the methods that class provides on these expressions.

For example, we've already seen that the expression `xml.film` returns a list of all the `<film>` elements. That expression *is* a bona fide `XMLElementList` instance, so by appending an `XMLElementList` method—say, `length()`—to the expression, you get something useful (in this case, the length of the list, which is 5). We know that in this context, counting starts with 0, so to see the title attribute of the last `<film>` element, put the following somewhat complex expression inside the array access operator (`[]`):

```
trace(xml.film[xml.film.length() - 1].@title);
```

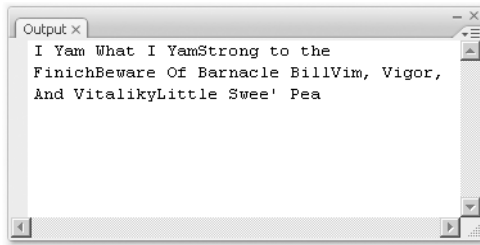
It may look a little scary, but it isn't when you reduce it to its parts. The expression `xml.film.length() - 1` evaluates to the number 4, so what you're seeing is as good as actually *using* the number 4.

To see the title attribute of all `<film>` elements, drop the array access operator altogether:

```
trace(xml.film.@title);
```



In the Output panel, you'll see that the combined results run together (as shown in Figure 11-1). The reason is because these attributes don't have any innate formatting; they aren't elements in a nested hierarchy, they are just individual strings. Let's fix that.



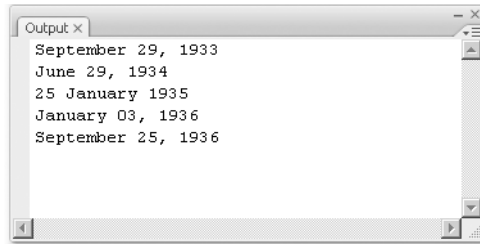
**Figure 11-1.** Unless they have their own line breaks, attributes will run together.

4. In this situation, another XMLList method can help you out. To make each title appear on its own line, append `toXMLString()` to the existing expression:

```
trace(xml.film.@title.toXMLString());
```

5. Swap title for releaseDate to see release dates instead, as shown in Figure 11-2:

```
trace(xml.film.@releaseDate.toXMLString());
```



**Figure 11-2.** Any element's attributes can be retrieved.

6. What about looking at a list of the cast members? Viewing individual cast members is just as easy. Update the `trace()` function to look like this:

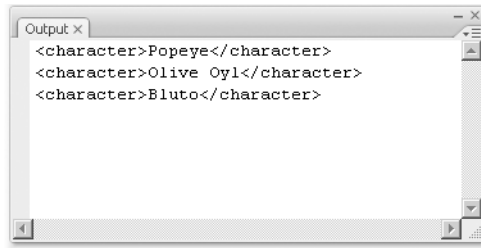
```
trace(xml.film[0].cast.character[1]);
```

That instructs Flash to look at the first `<film>` element's cast element and pull out the second of its character elements, which happens to be Olive Oyl. For fun, and to see how easy E4X makes things for you, contrast the preceding intuitive reference with its ActionScript 2.0 equivalent: `xml.firstChild.firstChild.firstChild.childNodes[1]`. Which would you rather use?

Moving back to the kinder, gentler world of ActionScript 3.0, update the `trace()` function as follows to see the whole cast of the third film:

```
trace(xml.film[2].cast.character);
```

Interesting—this time you get actual elements, complete with their tags, as shown in Figure 11-3.



**Figure 11-3.** Accessing elements selects the elements themselves, as well as their children.

This happens because of a characteristic of XML that isn't especially obvious. Once you know it, though, you're set. The characteristic goes like this: in the expression `<character>Popeye</character>`, you're not just looking at one element, you're *really* looking at two.

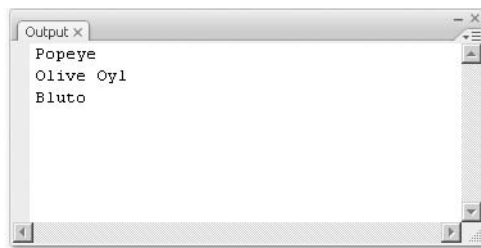
Both the tag (`<character>`) and its content (Popeye, in this case) are considered elements. In XML proper, these are also known as nodes, and there are a dozen node types. Flash only supports two of them, `ELEMENT_NODE` and `TEXT_NODE`, which is a relief—and knowing those two lets you easily pull out a tag's content. As before, an `XMLList` method, `descendants()`, comes to the rescue:

```
trace(xml.film[2].cast.character.descendants());
```

This gives you run-on results, because, like attributes, these text elements don't have any inherent formatting. All you need to do is just slap the `toXMLString()` method onto the end of the code line:

```
trace(xml.film[2].cast.character.descendants().toXMLString());
```

The result is exactly the sort of thing you might use to populate a text field (as shown in Figure 11-4).



**Figure 11-4.** Like attributes, text-only elements are nothing more than strings.

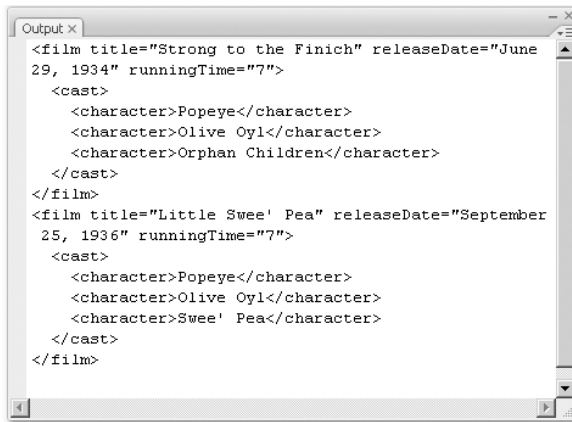
*Remember, we are dealing with text in this example, so although the results may look rather plain, you can format and manipulate them in a number of ways, as outlined in Chapters 6 and 10.*

7. All right, we'll give you one more illustration of E4X (we've saved the best for last). The popeye.xml file included with this chapter has slightly different runningTime attributes from those shown earlier in the chapter. Instead of a whole phrase, such as "6 min," these attributes show only numbers. Why? Because E4X allows you to make comparisons, which lets you filter content based on specific criteria.

Let's say you want to know which films have a running time longer than 6 minutes. Return again to our humble trace() function and update its parameter to the following:

```
trace(xml.film.(@runningTime > 6));
```

The result is a list of the <film> elements, and all their children whose runningTime attribute is greater than 6 (see Figure 11-5).



```

Output x
<film title="Strong to the Finish" releaseDate="June
29, 1934" runningTime="7">
  <cast>
    <character>Popeye</character>
    <character>Olive Oyl</character>
    <character>Orphan Children</character>
  </cast>
</film>
<film title="Little Sweet Pea" releaseDate="September
25, 1936" runningTime="7">
  <cast>
    <character>Popeye</character>
    <character>Olive Oyl</character>
    <character>Sweet Pea</character>
  </cast>
</film>

```

**Figure 11-5.** E4X allows filtering by way of comparison operators.

The parentheses tell Flash you're intending to filter the XMLList instance you get back. Inside the parentheses, the expression is a simple comparison, @runningTime > 6, which in plain English would be, "If you would be so kind, please tell us which <film> elements' runningTime attributes match this criterion." The reason Flash searches every <film> element in the bunch is because nothing appears between the word film and the dot that begins the next expression.

What if you want only the title of these films? Try this:

```
trace(xml.film.(@runningTime > 6).@title.toXMLString());
```

The trick, as always, is to break each expression into its parts. On its own, each concept is usually easy enough to understand. Concepts—expressions—are separated by dots. A blow-by-blow account of the preceding trace() goes like this:

trace(xml.film traces all <film> elements in the xml instance whose runningTime attribute is greater than the value 6—. (@runningTime > 6). This returns an XMLList instance that comprises a list of <film> elements, which the remaining expressions now reference. .@title shows the title attribute of <film> elements in the filtered XMLList instance, and .toXMLString() invokes the XMLList.toXMLString() method to clean up the results. Finally, ); closes the trace() function.

See the `LoadXML-E4XFiltering.fla` file for a working example of the preceding E4X filtering.

## E4X bonus round

True, we already said “one more illustration of E4X,” and that’s the preceding one. If you’re in a hurry to dispense with all this theory and jump head-first into a practical application, we doff our hats and invite you to make a beeline for the next section—but we figure at least a handful of you are wondering if it’s possible to return film titles based on who appears in the cartoon. Let’s pop open a can of spinach and take a gulp.

Open the `LoadXML-E4XBonusRound.fla` file that accompanies this chapter, and click into frame 1 of the scripts layer. Most of the ActionScript should look familiar. The important part appears in lines 9 through 11:

```
for each (var node:XML in xml.film.cast.character.
(descendants() == "Bluto")) {
    trace(node.parent().parent().@title);
}
```

Yup, this is new. The `for each..in` statement was introduced to ActionScript 3.0 thanks to the E4X specification. There has been a similar statement in ActionScript for quite some time, `for..in`, that works almost the same way, which is this: you point `for..in` at an object, and it loops through that object’s properties, however many properties there happen to be. What it loops on, however, are the properties’ *names*, rather than the properties themselves. This is either nifty or frustrating, depending on your needs. In contrast, the new `for each..in` statement loops on an object’s actual properties, which is great for the need we have in this particular endeavor.

To understand the mechanism of this E4X filtering, let’s start with a skeleton and slowly build up to the skin.

```
for each (someProperty in someObject) {
    // do something
}
```

The `someObject` in question is the hardest part of this equation, but based on what you’ve seen, it shouldn’t be impenetrable. The object is an `XMLList` instance determined by the expression `xml.film.cast.character.(descendants() == "Bluto")`. Stepping through the subexpressions dot by dot, we get the following:

- `xml.film`: All `<film>` elements in the `xml` instance.
- `.cast`: All `<cast>` elements of those films (each `<film>` element happens to only have one).

- `.character`: All `<character>` elements of those cast elements (each film's cast happens to have its own particular number).
- `.(descendants() == "Bluto")`: A comparison of the `TEXT_NODE` descendants of each `<character>` element—these could arguably be called the *value* of each `<character>` element—against the string "Bluto", which returns the `XMLList` instance. It's this `XMLList` instance that is the "someObject" of our skeleton.

That gives us the following:

```
for each (someProperty in xml.film.cast.character.
(descendants() == "Bluto")) {
    // do something
}
```

The replacement for our stand-in "someProperty" is an XML instance, stored in an arbitrarily named variable, `node`.

```
for each (var node:XML in xml.film.cast.character.
(descendants() == "Bluto")) {
    // do something
}
```

All this means is that `for each..in` is going to update the value of `node` to the latest XML object it finds in the `XMLList` instance as it steps through that list. The `node` variable effectively *is* the XML object in question, which means that you can apply your recently acquired E4X magic to it. Remember, at this point you're dealing with a text element, `Bluto`, inside the `<character>` element of a `<cast>` element of some `<film>` element. The text element's parent is `<character>`, whose parent is `<cast>`, which puts our point of view inside a `<film>` element. This outermost element has a `title` attribute, and we reference that with the `@` symbol:

```
for each (var node:XML in xml.film.cast.character.
(descendants() == "Bluto")) {
    trace(node.parent().parent().@title);
}
```

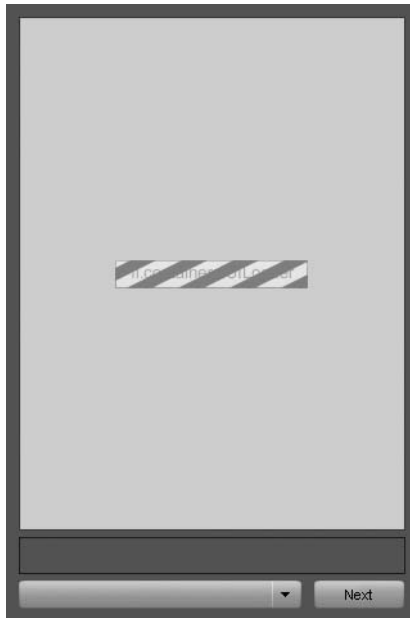
## Your turn: Using XML to build a slideshow

The popularity of websites like Flickr and Photobucket prove that people like to share photos. Of course, this was true even before the Internet, but modern technology makes it easier than ever to whip out that tumbling, unfolding wallet and proudly show off all the kids, aunts, uncles, cousin Eds, and Fidos, not only to friends, but to every continent on the planet. At the rate most people take pictures, if you were to make a photo slideshow in Flash, you'd want to be sure it was easy to update. With XML, that goal is closer than you may think.

To start, we're going to walk you through a self-contained, "hard-wired" movie that displays a small collection of external JPGs and their captions. The number of JPGs, and the order in which they appear, are "baked into" the SWF, which means that the movie must be edited and republished to accommodate new images. This slideshow features a ComboBox and Button component to let people choose which JPGs they want to see, and it even uses the UILoader and ProgressBar components to load the images, so this will be something of a cumulative exercise.

Once the test model is complete, we'll free the photo-specific data from its dungeon and move it to an XML file, where it can leap free in the fields like a shorn sheep. Here we go!

1. Start a new Flash document and save it as `Slideshow.fla` in the `Slides` folder that contains the six JPGs accompanying this chapter. Set the movie's dimensions to 320×480 and set the background color to whatever you like (we chose #336699, which is light blue).
2. Create the following five layers: scripts, progress bar, loader, caption, and nav. Now it's time to populate this framework.
3. Open the Components panel (Window ► Components) and drag an instance of the ProgressBar component to the progress bar layer. Use the Property inspector to set its width to 150, height to 22, x position to 85, and y position to 200. Give it the instance name `pb`.
4. Drag an instance of the UILoader component to the loader layer. In the Property inspector, set its width to 300, height to 400, x position to 10, and y position to 10. Give it the instance name `loader`.
5. Captions will be displayed with a text field. Use the Text tool to create a dynamic text field in the caption layer. Switch to the Selection tool and set the text field's width to 300, height to 28, x position to 10, and y position to 416. Give this text field the instance name `caption`. Using the Property inspector, set the following values for the text box:
  - Font: `_sans`
  - Size: 18pt
  - Color: #FFFFFF (white), so that it shows over the blue background
6. Drag an instance of the ComboBox and Button components to the nav layer.
7. In the Property inspector, set the combo box's width to 220, height to 22, x position to 10, and y position to 450. Give it the instance name `images`. For the button, set its width to 70, height to 22, x position 240, and y position to 450. Give it the instance name `next`.
8. With the button selected, click the Parameters tab and set the button's Label parameter to `Next`. At this point, you have something like the scaffolding shown in Figure 11-6.



**Figure 11-6.** The parts are in place; time for the ActionScript.

Now it's time to bring these parts to life. For the most part, it's a matter of handling events for the components and populating the combo box. We only have a handful of steps, here, and we'll have something to test.

1. Click into frame 1 of the scripts layer and open the Actions panel (Window > Actions). Here's the first chunk of code:

```
import fl.data.DataProvider;

// Set up image file info and captions
var imageData:Array = new Array(
    {label:"Geocaching Photo 1", data:"geocache01.jpg", caption:➤
    "I have the GPS; you follow me."},
    {label:"Geocaching Photo 2", data:"geocache02.jpg", caption:➤
    "This says three paces ahead."},
    {label:"Geocaching Photo 3", data:"geocache03.jpg", caption:➤
    "Cool! Fingerpuppet treasure!"},
    {label:"Geocaching Photo 4", data:"geocache04.jpg", caption:➤
    "I found a pretty pony!"},
    {label:"Geocaching Photo 5", data:"geocache05.jpg", caption:➤
    "Treasure hunting with my cousins."},
    {label:"Geocaching Photo 6", data:"geocache06.jpg", caption:➤
    "May I have the stickers?"}
);
```

The first line imports the `DataProvider` class, which is needed later when it's time to populate the combo box. After that, an arbitrarily named variable, `imageData`, is set to an instance of the `Array` class. Arrays are lists of whatever you put in them. You can use the `Array.push()` method on an instance to add elements to that instance, but you can also pass in the whole collection at once, which we've done here. This array has six items, and each item is an instance of the generic `Object` class with three properties. What, no new `Object()` statement? How are these objects being created? The curly braces (`{}`) take care of that. It's a shortcut, and we're taking it. You'll remember from Chapter 9 that `ComboBox` instances can be supplied with label and data information, so that explains what those properties are. The `caption` property is a custom addition.

2. Press the Enter/Return key a couple times and type in the following:

```
// Keep track of current image
var currentImage:Number = 0;

// Picture changing function
function changePicture(pict:Number):void {
    pb.visible = true;
    caption.text = imageData[pict].caption;
    loader.load(new URLRequest(imageData[pict].data));
}
changePicture(0);
```

The first line after the comment declares a variable, `currentImage`, and sets it to 0. This number will keep track of which image is being looked at. The next several lines declare a custom function, `changePicture()`, that accepts a single parameter, `pict`. This function does the following three things:

- Makes the `ProgressBar` instance visible (yes, it's already visible at this point, but later code turns off its visibility when an image finishes loading).
- Makes the text field display the current caption. The incoming `pict` parameter determines which element to retrieve from the `imageData` array, and that element's `caption` property is consulted.
- Makes the `URLoader` instance load the current image. Here, again, the `imageData` array is consulted, but this time from the relevant item's `data` property.

Immediately after its declaration, the `changePicture()` function is called, with 0 as its parameter. Why start at 0? Because that's where arrays start counting. We're displaying the first image and its caption.

Now we just have to hook up the components.

3. Press Enter/Return a couple times, and type in the following:

```
// Wire up progress bar
pb.source = loader;

// Handle progress bar loading completion
```



```
pb.addEventListener(
    Event.COMPLETE,
    function(evt:Event):void {
        pb.visible = false;
    }
);
```

The first line after the comment associates the `ProgressBar` instance with the `UILoader` instance. As the `UILoader` component loads images, the progress bar will *automagically* know what to do. The next little block makes the progress bar invisible when loading is complete.

4. Press Enter/Return a couple times, and type in the following:

```
// Wire up combo box data provider
images.dataProvider = new DataProvider(imageData);

// Handle combo box changes
images.addEventListener(
    Event.CHANGE,
    function(evt:Event):void {
        changePicture(images.selectedIndex);
    }
);
```

The first line after the comment populates the combo box by setting its `dataProvider` property to a new `DataProvider` instance (this is why we need the `import` statement at the top). All the `DataProvider` instance needs is an array whose elements have `label` and `data` properties, which is exactly what we have. The `caption` properties are extra, but they don't hurt anything. This one line—`images.dataProvider = new DataProvider(imageData)`;—shoves the whole `imageData` array's content into the combo box in one swoop. Next, the `Event.CHANGE` event is handled for the combo box. The handler function calls the custom `changePicture()` function and feeds it a number determined by the combo box's current selection.

5. There's just one thing left—the button. Press Enter/Return a couple times and type in the following:

```
// Handle button clicks
next.addEventListener(
    MouseEvent.CLICK,
    function(evt:MouseEvent):void {
        currentImage++;
        if (currentImage == imageData.length) {
            currentImage = 0;
        }
        images.selectedIndex = currentImage;
        changePicture(currentImage);
    }
);
```

Here, the `MouseEvent.CLICK` event is handled for the button. The handler function does the following:

- Increments the `currentImage` variable by one.
  - Checks to see if `currentImage` shares the same value as the expression `imageData.length` (the number of items in the `imageData` array). If so, it sets `currentImage` back to 0.
  - Sets the combo box's current selection to `currentImage`.
  - Calls the custom `changePicture()` function and passes it `currentImage` as its parameter.
6. Test the movie. You'll be treated to a mini-geocaching excursion, led by a young enthusiast, as shown in Figure 11-7. Click the `Next` button to flip through the pictures in sequence, or use the combo box to skip around. To simulate image downloads, so you can see the progress bar in action, select `View > Simulate Download` from the SWF window.



**Figure 11-7.** A few quick components and a bit of ActionScript, and you're off!

As it turns out, geocaching photos make a decent metaphor for this chapter, because after all this careful, plodding double-checking of coordinates, we're about to uncover some treasure—only a few more paces in a westerly direction.

1. Save your file to keep everything safe, and then select File ► Save As and save a copy as SlideshowXML.fla into the same folder.
2. Click back into frame 1 of the scripts layer to make a few changes. Here's the first chunk of code, with the revisions shown in bold:

```
import fl.data.DataProvider;

// Pull in image information from XML
var xml:XML = new XML();
var xmlLoader:URLLoader = new URLLoader();
xmlLoader.load(new URLRequest("slideshow.xml"));
xmlLoader.addEventListener(
    Event.COMPLETE,
    function(evt:Event):void {
        xml = XML(evt.target.data);
        images.dataProvider = new DataProvider(xml);
        changePicture(0);
    }
);

// Keep track of current image
var currentImage:Number = 0;
```

The `imageData` array is gone completely. In its place stands our trusty XML loading formula. The only difference here is that the instance name for the `URLLoader` instance has been changed to `xmlLoader`, because `loader` is already in use as the instance name for the `UIloader` component. This time, we're loading the file `slideshow.xml`, and that's where the former `imageData` content now resides. Translated into XML, it looks like this:

```
<slideshow>
  <slide label="Geocaching Photo 1" data="geocache01.jpg" ➤
caption="I have the GPS; you follow me." />
  <slide label="Geocaching Photo 2" data="geocache02.jpg" ➤
caption="This says three paces ahead." />
  <slide label="Geocaching Photo 3" data="geocache03.jpg" ➤
caption="Cool! Fingerpuppet treasure!" />
  <slide label="Geocaching Photo 4" data="geocache04.jpg" ➤
caption="I found a pretty pony!" />
  <slide label="Geocaching Photo 5" data="geocache05.jpg" ➤
caption="Treasure hunting with my cousins." />
  <slide label="Geocaching Photo 6" data="geocache06.jpg" ➤
caption="May I have the stickers?" />
</slideshow>
```

As you can see, this XML document closely mirrors the original `imageData` array.

Let's take another look at the `Event.COMPLETE` event handler for the `xmlLoader` instance. The function runs as follows:

```
function(evt:Event):void {
    xml = XML(evt.target.data);
    images.dataProvider = new DataProvider(xml);
    changePicture(0);
}
```

There are a couple of important things to note. First, the `DataProvider` goings-on have been moved here from their former position next to the combo box `Event.CHANGE` handler. Why? Because under the circumstances, the combo box can't be populated until the XML has loaded. Next, the `changePicture()` call has been removed from its original place and relocated here. Why? Same reason—until the XML loads, the `changePicture()` function has no reference for what image to summon.

Two more paces!

3. At or near line 20, you'll find the `changeFunction()` declaration. You'll need to tweak two lines (the changed portions are shown in bold in the following code):

```
// Picture changing function
function changePicture(pict:Number):void {
    pb.visible = true;
    caption.text = xml.slide[pict].@caption;
    loader.load(new URLRequest(xml.slide[pict].@data));
}
```

*Note that the `changePicture(0)` line after this function has been removed. Instead of pulling from the old `imageData` array, the text field and `XMLLoader` component now draw their information from the `xml` instance, using the E4X syntax to specify the relevant `<slide>` element attributes. Here, the function's incoming `pict` parameter serves the same purpose it did before: it specifies which `<slide>` element to consult. Don't forget to delete what used to be the last line in this chunk—that `changePicture(0)`; call is now inside the `Event.COMPLETE` event handler for the `xmlLoader` instance.*

Treasure's in sight!

4. Here are the last touch-ups. First, delete the data provider line:

```
images.dataProvider = new DataProvider(imageData);
```

which has since been moved to the `Event.COMPLETE` handler. Finally, change one reference in the button's event handler:

```
// Handle button clicks
next.addEventListener(
    MouseEvent.CLICK,
    function(evt:MouseEvent):void {
        currentImage++;
    }
);
```

```

    if (currentImage == xml.slide.length()) {
        currentImage = 0;
    }
    images.selectedIndex = currentImage;
    changePicture(currentImage);
}
);

```

Since `imageData` is no more, the `if()` statement needs to look to the number of `<slide>` elements, instead.

#### 5. Test the movie and watch the show again.

If you think you missed a step, compare your work to the `SlideshowXML.fla` file in the Complete folder. Now that the movie has become *XML-ified*, you can have some fun editing `slideshow.xml` and running the SWF to see the changes.

For example, delete the first three `<slide>` elements and test the movie again. Like magic, only the three remaining slides and captions display. Change the wording of one of the captions and run the SWF again. Change the order of the `<slide>` elements. Every time, the SWF takes these changes effortlessly in stride.

## What you've learned

In this chapter, we gave you the absolute basics of XML use in Flash. Though on the surface it may not seem like much, what we have presented in this chapter forms the foundation for complex Flash projects ranging from video pickers, MP3 players, and portfolio sites to e-commerce applications. In this chapter, you have discovered

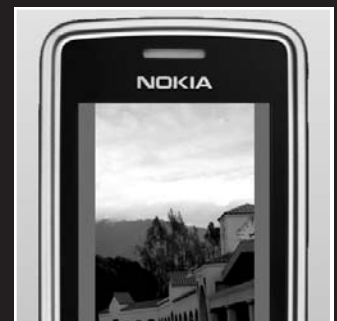
- The relationship between an XML document and Flash CS3
- How to retrieve and filter XML data using E4X syntax
- How to build a dynamic slideshow driven by XML

The most important point you need to take away from this chapter is the sheer flexibility of XML in your Flash design and development efforts. You can make your movies expand or contract effortlessly by simply adding to or subtracting from the XML document being used by the movie. This is the true meaning of *dynamic*.

Speaking of dynamic, one of the hottest and most dynamic aspects of our industry is the recent emergence of cell phones as yet another medium for our work. Turn the page to discover how we can pull our Flash work off of the computer and make it mobile.



# 12 GOING MOBILE IN FLASH



The first thing you need to know about mobile is this: the question to be asked is not *if* you will develop a mobile Flash application but *when* you will develop a Flash mobile application. In this chapter, our intention is not to turn you into a mobile developer. Instead, it is to introduce you the tools you will use: Device Central and Flash CS3.

As you are reading this, Flash is rapidly becoming the de facto standard for content delivery through cell phones in Asia and Europe. Recent announcements regarding the North American market indicate this geographic region is also “getting in the game.” In fact, things are happening so quickly in North America that the question we asked in the previous paragraph—When will you start developing mobile applications?—is one you will have to answer much sooner than you may think.

What we’ll cover in this chapter:

- Navigating around Device Central
- Creating device sets
- Creating mobile Flash documents using Device Central
- Using Flash Lite 2, Flash, and Device Central to test a simple movie
- Using ActionScript 2.0 to control a mobile application

Files used in this chapter:

- `OjaiAdventure.fla` (Chapter12/ExerciseFiles\_CH12/OjaiAdventure.fla)

## Flash and devices

It should not be news to discover that Flash has gone mobile. The sheer number of wireless handsets and other wireless devices is driving a demand for rich content that makes the text-based solutions for cell phones and PDAs that were all the rage a couple of years ago look like charcoal scrawls on a cave wall. The reason is the mobile market discovered what Flash developers have known for years: Flash content results in small, fast-loading files.

It isn’t only corporations that are driving the demand for mobile content. Educational institutions ranging from K-12 to postsecondary across North America are seriously examining how these devices can be used to deliver educational content and are developing courses that teach their students how to produce applications for these devices. Whether it is a Flash developer friend of ours using his cell phone to track his progress along the Pacific Coast Highway in California or pedestrians hooking into the traffic cameras in New York City, it is difficult for most to understand this technology is still in its infancy. To get a real sense of what you can do, start with this chapter, and then point your browser to the Adobe Mobile & Devices Developer Center at [www.adobe.com/devnet/devices/](http://www.adobe.com/devnet/devices/).



*We would be remiss in not mentioning that one of the best Flash mobile books out there is *Foundation Flash Applications for Mobile Devices* by Richard Leggett et al. (friends of ED, 2006).*

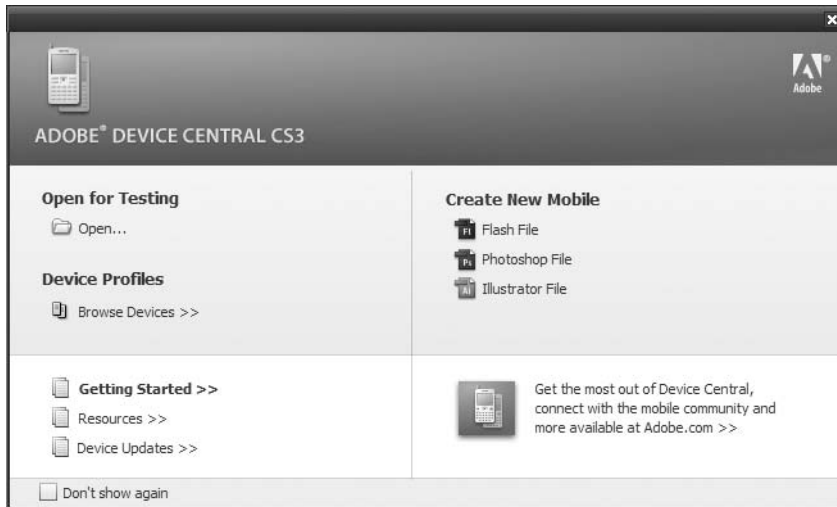
The first thing you need to know about developing Flash applications for mobile devices is: *Not one thing you have learned about Flash CS3 and ActionScript 3.0 in this book applies to devices.* Everything you have done to this point in the book has relied on Flash Player 9. Flash mobile uses *Flash Lite 2.0*, which is roughly equivalent to Flash Player 7. This means that every scrap of ActionScript 3.0 code and every component, filter, or blend effect added to a new ActionScript 3.0 document simply can't be used in a device.

We tell you this not to scare you off but to make sure you enter this fascinating and emerging field with your eyes wide open. Other things you need to know include the following:

- **The phone is not a computer:** There is no mouse for user input. You are limited to the up, down, and select keys on a handset along with the number keys and \* and # keys on the phone's keypad.
- **Forget about fonts:** You are dealing with a screen that could be around 170 pixels square. This means text must be both legible and readable. Though you can embed fonts, they add to file size.
- **Small is a very good thing:** If something has no purpose other than to add weight . . . throw it overboard. Flash content in devices is the only content you will deliver that people have to pay for. Data downloads are charged on the user's cell bill by having the user "pay by the K." For example, one of the authors has a plan that charges 3 cents per kilobyte downloaded. This can quickly add up. Keep the code to a minimum, and, if you must use bitmaps, use them at their final size and use compressed bitmaps. Avoid vectors if you can, and substitute them with JPG images. (Yes, this isn't what you would expect to hear while authoring Flash content—devices are different!)
- **Be aware of the device:** No two handsets are the same, and this includes screen real estate. Device Central will become your most important resource for this aspect of Flash development.
- **Test, test, test, test, test . . .** : Did we mention test everything? That includes testing both in the emulator in Device Central and on the actual handset.

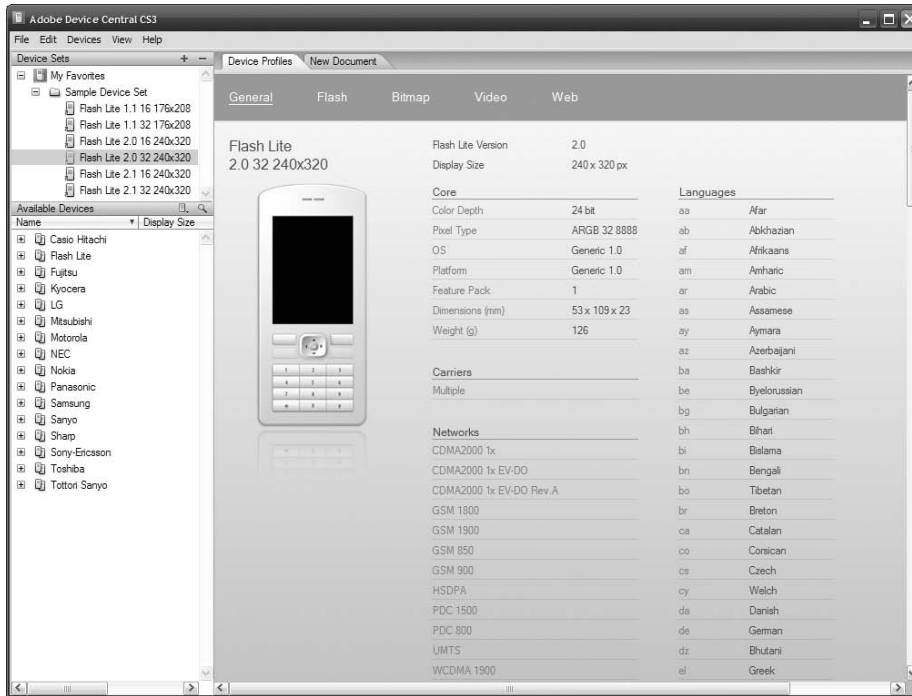
## Device Central CS3

New to the CS3 Studio is Device Central. Though aimed primarily at Flash designers and developers, direct access to Device Central is also available in Dreamweaver CS3, Illustrator CS3, and Photoshop CS3. In fact, when you launch the application, the Start page, shown in Figure 12-1, gives you the option of creating Illustrator, Photoshop, or Flash documents. Let's go wander around Device Central and get comfortable with it.



**Figure 12-1.** Welcome to Device Central.

1. Launch Device Central, and in the Start page, under Create New Mobile, select Flash File.
2. When Device Central opens, click the Flash Lite 2.0 32 240X320 device in the Device Sets area. If it isn't selected, click the Device Profiles tab. The Device Profile screen that opens will present you with a view of a generic phone (see Figure 12-2). Along with the phone you will be given a lot of information, ranging from generic properties such as Dimensions and Weight to the languages available in phones that use Flash Lite Player. Selecting the Flash, Bitmap, Video, and Web links will also give you extra information regarding what types of FSCommands can be used in Flash, the types of images you can prepare, the video formats that work on the device, and the coding languages you can use when developing web pages for the device. Now let's get more specific and actually choose a device.



**Figure 12-2.** You can start with generic information regarding a particular player.

*FSCommands? These were quite common and quite mysterious in ActionScript 1.0 and Flash Player 3. These commands are used to allow Flash Lite Player to communicate with the device's operating system.*

3. Along the bottom-left side of Device Central is a listing of all the phones for which profiles are currently available in Device Central. As the next few months and years pass, we suspect this will become a rather crowded list. For now, click the + sign beside the Motorola link and select the Motorola RAZR V3m device. As soon as you do this, the image in the Device Profiles panel will change, as you see in Figure 12-3, as will the general information for the selected device.

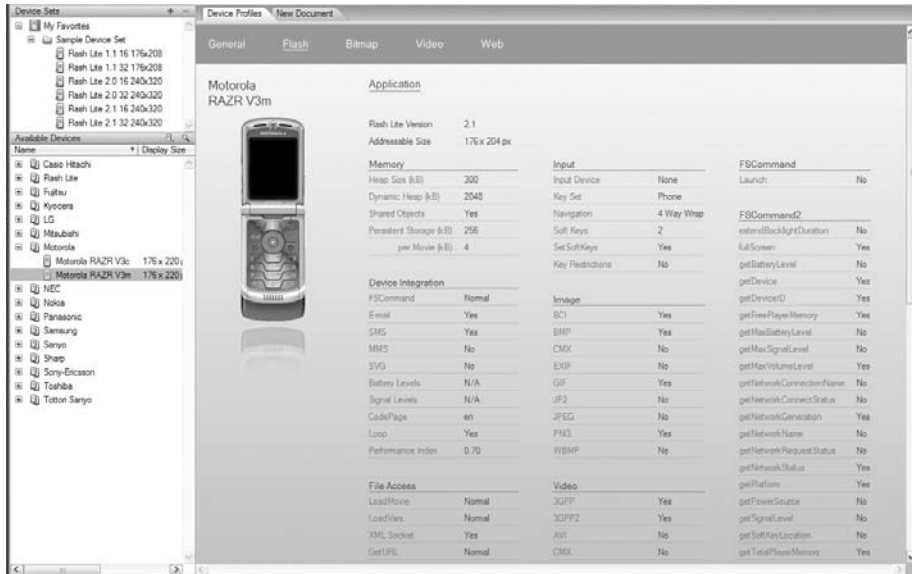


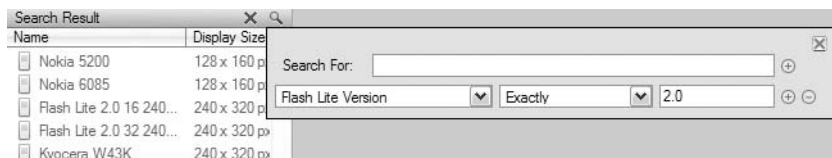
Figure 12-3. You can select a specific device.

4. There will also be occasions when you can't remember the name of a device, but you know who made it and you know what it looks like. To see all of the Nokia devices, for example, click Nokia. All of the Nokia phones in the category will appear along the top of the Device Profiles panel, and the various features of the phone will be listed under the phone's image. Twirl down the various sections, as shown in Figure 12-4, to obtain even more specific information regarding a specific model of handset.
5. What if you know the name or the model number but can't remember the manufacturer? Click the Search Devices icon on the Available Devices bar—it is the Magnifying Glass—to open the Search dialog box. Enter the number 6 in the Search For text input box, and all of the devices with that number in the model name will appear. Add a 0 to the search criteria, and the list is winnowed even further to all devices with 60 in the model name.



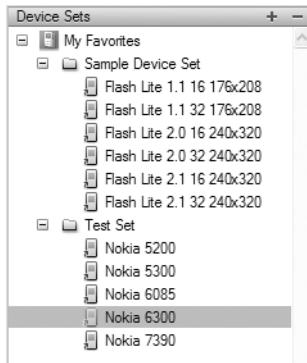
**Figure 12-4.** You can view an entire collection of devices and compare feature sets.

- You can also do very specific searches. For example, say you are curious which devices use only Flash Lite 2.0 Player. Open the Search dialog box. Leave the Search For area empty, but click the Add Search Criteria button to the right of the Search For input box. When the menu expands, select Flash Lite Version from the drop-down menu on the left, select Exactly from the middle drop-down menu, and enter 2.0 into the input box as shown in Figure 12-5. You will see a list of the devices that use this player.



**Figure 12-5.** Device Central contains a blazingly fast search engine.

7. You don't always have to use the search engine to group the devices. Click the Group By button (the icon to the left of the Magnifying Glass) and select Flash Lite Version from the drop-down menu. The devices will all be grouped by version. If you open the Flash Lite 2.0 grouping, all of the devices that use this player will be listed.
8. Another scenario you might encounter is your constantly having to develop for the same various handsets. In this instance, click the + sign (New Device Sets button) in the Device Sets bar. Name the folder that appears as Test Set. Select the Nokia phones listed in the 2.0 listing of the Available Devices area and drag them to the folder. If you expand the Test Set folder, as in Figure 12-6, you will see you have created a test set of all the Nokia phones that use Flash Lite 2.0.



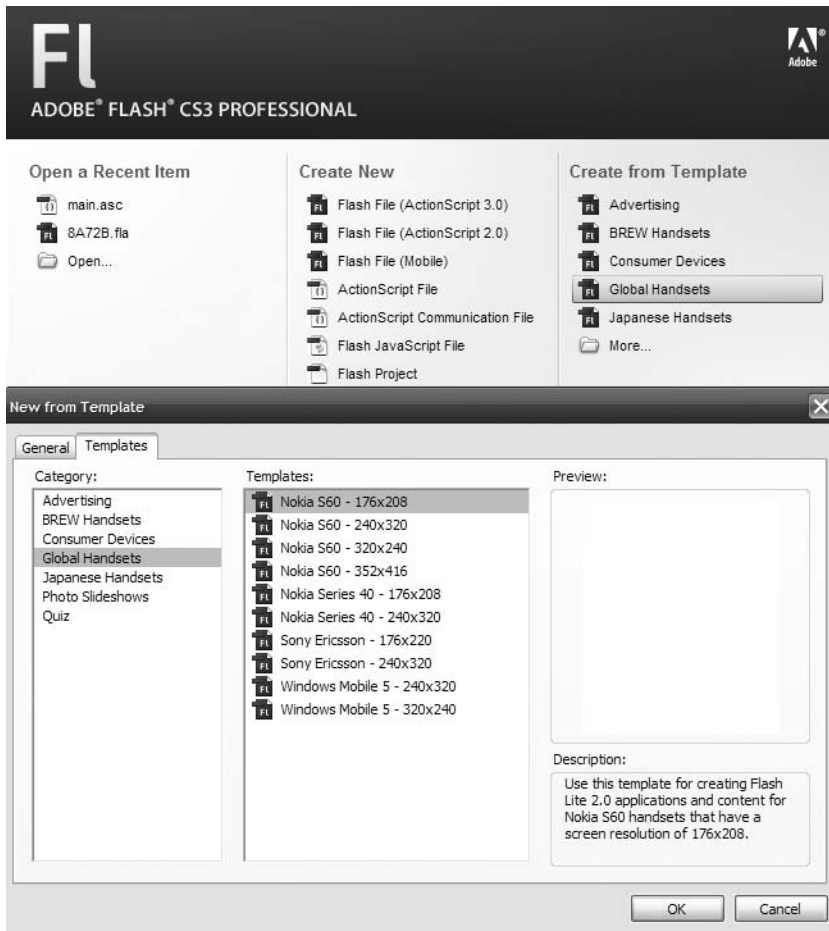
**Figure 12-6.** You can create custom groupings, called “sets,” of the devices you will design for.

*To remove a set, select its name in the Device Sets panel and press the Delete key.*

## Creating a new Flash document using Device Central

When you launch Flash CS3, you will notice one of the new document options in the Start page is Flash File (Mobile). Clicking this will launch Device Central, and the New Document tab will be selected. This is how you start creating Flash movies for mobile devices.

An interesting aspect of this path to Device Central is contained on the right side of the Flash Start page in the Create from Template area. If you click the Global Handsets selection, the New from Template dialog box opens and a preselected list of Flash Lite 2.0 templates will appear (see Figure 12-7). Click one of those choices, and Device Central will launch. The bottom line is it is irrelevant whether you start in Flash or start in Device Central because you are going to wind up in Device Central no matter which route you take. Now that you know how to get there, let's look at how to create a mobile document.



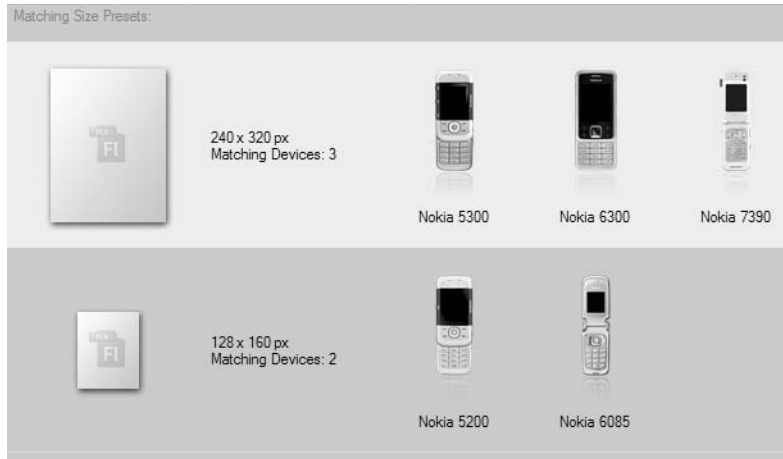
**Figure 12-7.** How to get to Device Central from Flash

*A lesser-known path to Device Central is through Adobe Bridge. With Bridge open, you can either select an item and select File ► Test in Device Central, or right-click (PC) or Ctrl-click (Mac) an item and select Test in Device Central from the context menu.*

12

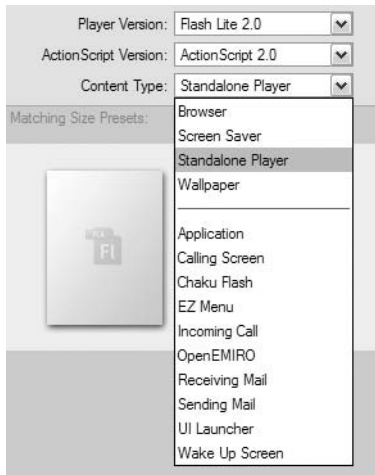
1. Launch Device Central and select the Nokia 6300 device in your set or in the Available Devices area.
2. Click the New Document tab, and the phone and a Flash document icon will appear in the New Document panel.

3. Click the Test Set folder created in the previous section, and all of the devices and Flash page sizes will appear (see Figure 12-8). This is how you can select multiple devices. From this you can instantly determine that you may need to create two presentations, because the devices in the set have differing screen sizes. Let's not get complicated. Reselect the Nokia 6300 device.



**Figure 12-8.** You can create content for a single device or for a device set.

4. In the New Document panel, select Flash Lite 2.0 in the Player Version drop-down menu and ActionScript 2.0 from the ActionScript Version drop-down menu. In the Content Type drop-down menu, shown in Figure 12-9, select Standalone Player.



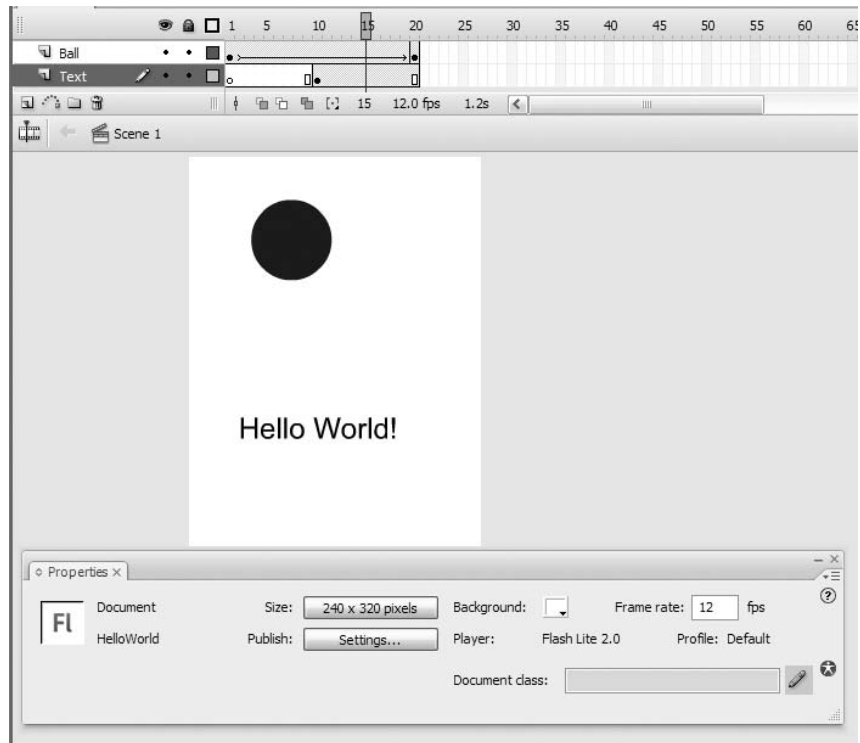
**Figure 12-9.** Choosing a content type



You are probably looking at that list in the Content Type drop-down menu and wondering “What the . . . ?” The good news is not all devices support everything in the list. If you were to select Calling Screen from the list, the device would disappear. This is a visual clue the Nokia 6300 doesn’t support this content type.

Still, it is important that you at least have a vague understanding of the content types. Here’s a brief description of each one:

- **Browser:** Uses the player to render Flash content embedded in a mobile web page and viewed in the device’s browser.
  - **Screen Saver:** Uses Flash Lite to show the device’s screen saver.
  - **Standalone Player:** Turns Flash Lite into a stand-alone application. This means the user can open the SWF anytime and view the movie without having to launch a browser.
  - **Wallpaper:** Turns the Flash movie into the screen’s wallpaper.
  - **Application:** Defines the movie as a stand-alone Flash application.
  - **Calling Screen:** Uses Flash Lite to play an animation when the user receives or makes a call.
  - **Chaku Flash:** Uses the SWF as the ring tone.
  - **EZ Menu:** Uses the SWF as the device’s menu.
  - **OpenEMIRO:** Plays the SWF while the device is waking up from standby mode.
  - **Receiving Mail:** Plays the SWF animation when an e-mail is received.
  - **Sending Mail:** Plays the SWF animation while the e-mail message is being sent.
  - **UI Launcher:** Defines Flash Lite as the device’s application launcher and to display the device’s launcher application.
  - **Wake Up Screen:** Plays the SWF as the phone is starting.
5. Click the **Create** button in the bottom-right corner of the **New Document** panel. This will launch Flash CS3 and, when the document opens, note the **Flash Player** chosen in **Device Central**—Flash Lite 2.0—appears in the **Property inspector**.
  6. Add a new layer, and name **Layer 1** as **Text** and **Layer 2** as **Ball**.
  7. Click the **Ball** layer to select and draw a circle on the stage. Convert the shape to a movieclip named **Ball**.
  8. Move the movieclip to the bottom of the stage and add a keyframe in frame 20 of the **Ball** layer. Move the ball to the top of the stage and insert a motion tween.
  9. Add a keyframe in frame 10 of the **Text** layer. Select the **Text** tool, click the stage, and enter **Hello World**. Use **\_sans** as the font and set the size to 24 pixels.
  10. Add a frame to frame 20 of the **Text** layer, as shown in Figure 12-10, and save the movie to your **Exercise** folder.



**Figure 12-10.** The Flash movie is ready to go mobile.

## Testing a mobile movie

Normally your final instruction would be to test the movie. Not just yet. We want to save the best for last. Unlike what you have done to this point in the book, when you test a movie, you don't play the movie in Flash Player. When it comes to mobile, you test your movie in the device chosen. Here's how:

1. Test the movie. What happens next is Device Central opens and sprouts a new Emulator tab, and the movie starts playing in the device chosen as shown in Figure 12-11.



**Figure 12-11.** Welcome to the world of mobile Flash content.

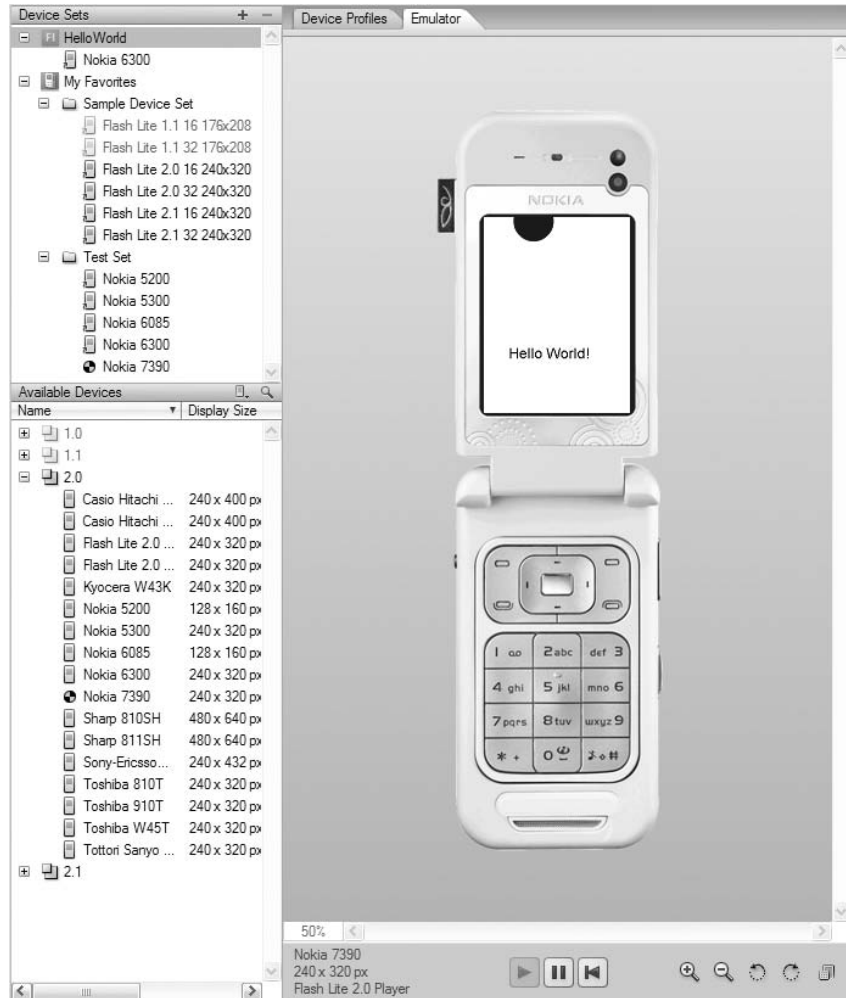
There are some aspects of the emulator of which you need to be aware.

The first item you should notice is your Flash movie is now at the top of the device list, and you see the device being used to test the movie. The next thing to notice is a change to the icon beside Nokia 6300 under Test Set. This icon is telling you this is the device being used in the emulator. The Emulator panel, where the movie is actually playing, gives you some information regarding the phone, the screen size, and the player version being used. The three buttons allow you to play, stop, and rewind the movie. You can also zoom in or zoom out on the device. The button on the right is the Toggle Detached View button. This mode is designed for devices such as clamshell devices that don't allow you to view the content at 100% view.

The panels on the right side of the emulator are the “jewels” of Device Central. These are called the **testing panels** and, in many respects, can be regarded as the “Flash Bandwidth Profiler on Steroids.” For example, many devices allow you to control screen brightness. What would be the effect of reducing the brightness on how the movie looks while it is playing? Let's find out:

2. Twirl down the Display panel and drag the Backlight, Gamma, and Contrast sliders to the right or left. Notice how the changes affect the “look” of the screen in the emulator.

3. What would the movie look like in the Nokia 7390? Double-click the device in the Test Set. The emulator will change, as shown in Figure 12-12, to the device chosen.



**Figure 12-12.** Switch devices by double-clicking the name of the new device.

*Here's a little-known trick regarding Device Central. If you were to move the HelloWorld.fla file to a Mac and open it in Flash CS3, you will see it is a Flash Lite 2.0 device. If you test the movie, Device Central will open, and the movie will start playing in a Nokia 6300 emulator.*

## Publishing a mobile movie

You have done all of the emulator stuff and are happy with the movie. The final step in the process is actually publishing the SWF file for the Nokia 6300. Here's how you do that:

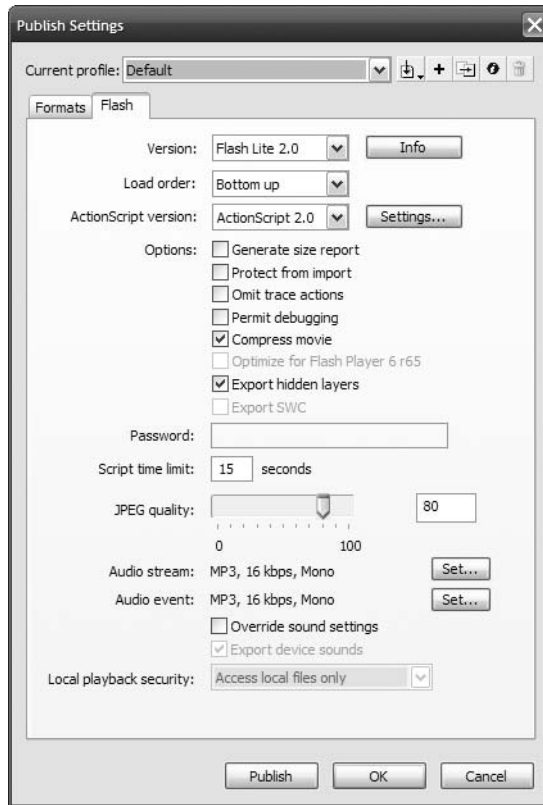
1. In Device Central, select File ► Return to Flash. Alternatively, you could press Ctrl+Shift+O (PC) or, as shown in Figure 12-13, press Cmd+Shift+O (Mac).



Figure 12-13. From Device Central back to Flash to publish the SWF

*If you open a new Flash document in Device Central, the Return to Flash menu item will change to Jump to Flash. Select this, and Flash CS3 will launch.*

2. When you return to Flash, select File ► Publish Settings to open the Publish Settings dialog box. As this is a stand-alone player, HTML is not necessary. Deselect HTML in the Formats panel.
3. Click the Flash tab to open the Publish Settings dialog box. Notice, as you see in Figure 12-14, that Flash Lite 2.0 is the player version selected. Click the Publish button.



**Figure 12-14.** Publishing the SWF for device playback

At this point in the process, we step aside because how content moves from the PC to the device is dependent upon the software provided by the handset manufacturer. In the case of our test device, it would be the **Nokia PC Suite** software. Getting the SWF file from the desktop to the phone is as easy as dragging the file from the Chapter 12 Completed folder into the folder targeted in the Nokia software (see Figure 12-15). From this point on, you would test the software and make any changes needed.



**Figure 12-15.** It is as simple as a drag-and-drop operation to get the SWF from the computer into the phone.

## Constructing a mobile application

In this exercise, you are going to construct a small application that presents a series of images from a recent trip to California. The purpose here is to get you sensitized to the fact that constructing mobile presentations is a lot different from building a web page in Dreamweaver CS3 or a Flash presentation in Flash Professional CS3.

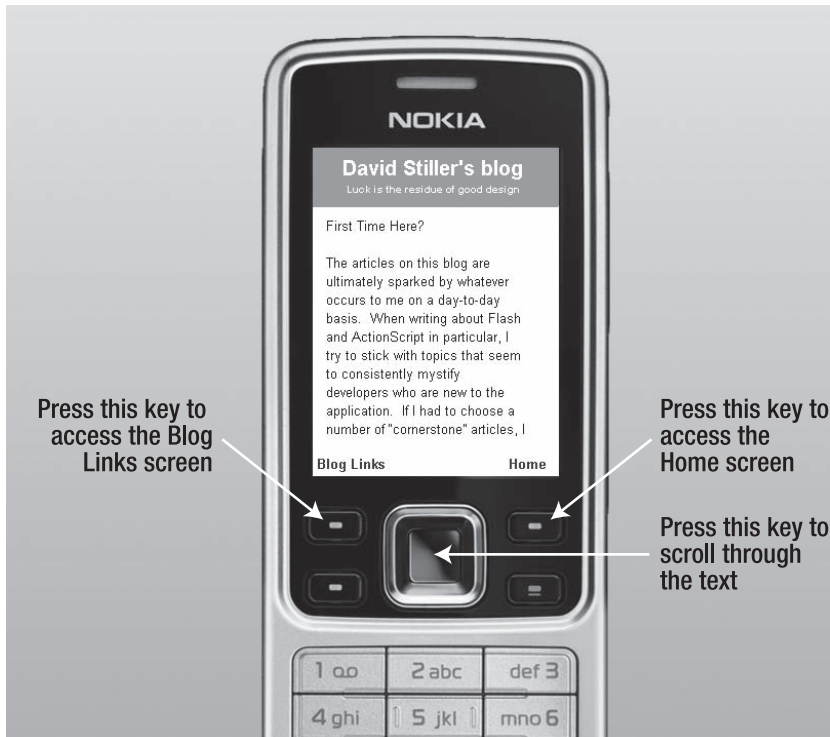
The first thing you must understand is simply moving an HTML site onto a mobile site is somewhat similar to trying to stuff 10 pounds of leaves into a bag that only holds 5 pounds of leaves. You can put a lot of information into a space that has a resolution of 1440×900 pixels or more of screen real estate. When you try to present the same amount of information in a screen that is 320×240, the information hierarchy becomes paramount, and design, in many cases, takes a back seat. To understand what we are talking about, let's look at the entry to David Stiller's blog shown in Figure 12-16.

There is a lot of information on the page. There is the banner at the top, the text block, the search box, and links to all of David's postings on the right side of the page. For all of this information to move into the mobile space, David would have to redesign the page and actually make it a series of screens on a mobile device. For example, the banner at the top could be common to each screen, but the post would need to be scrolling text, as shown in Figure 12-17, and the right and left soft keys—they are the ones to the right and left of the navigation buttons in the center—would need to be coded to allow the user to press the keys to either go to the main blog screen or navigate to the links screen.



**Figure 12-16.** You can't simply "port" a site from the web space to the mobile space.





**Figure 12-17.** There is no mouse in the mobile space. There are only soft keys.

As well, the up and down keys found in the four-directional pad would need to be coded to allow the user to scroll through the post.

Now that you are aware that developing for mobile requires you to “think differently,” let’s create an application:

1. Open the `OjaiAdventure.fla` file in the Chapter 12 Exercise folder. The bits and pieces needed to construct this application can be found in the library. Other than that, you are going to pull this project together and test it on the Nokia 6300 we have been using throughout this chapter.

*What’s with the fixation on the Nokia 6300? Nothing really. We want to be able to use ActionScript 2.0, which many of you are familiar with, and Flash Lite 2.0 Player. The device we are using simply looks “sexy” and makes for great screen shots.*

2. Add seven layers to the movie and name them, from the top down, as follows:

- Labels
- Actions
- Images
- Thumbs
- Open
- NavButtons
- SoftKeyID
- Head

3. Add a frame at frame 25 for all of your layers.

*By the time this project is finished, you will look at the timeline and think, “Shoot, I could have done this in three frames.” You would be quite correct with this statement, but we are using the extra frames to give you an opportunity to “see” how the various features of this presentation are constructed on the timeline.*

4. Select the Head layer. Select the Text tool, click the stage, and enter Excellent Ojai Adventure. In the Property inspector, apply the following formatting:

- Text Type: Static Text
- Width: 205
- X: 18.9
- H: 9.4
- Font: Times or \_serif
- Size: 30 pixels
- Color: #E6E1AF (yellow)
- Alignment: Centered
- Aliasing: Use device fonts

5. Select the keyframe in the Labels layer, and in the Property inspector, change the Frame label to Main.

6. Select the SoftKeyID layer. Select the Text tool, click the stage, and enter the word Main. Use the following settings:

- Text Type: Static Text
- X: 5
- Y: 300
- Font: \_sans
- Size: 12
- Color: #E6E1AF (yellow)

7. Repeat Step 5 and enter the word Exit. The only difference will be the text will be placed at the X and Y coordinates of 198 and 300.

The text created is a major job to do. It will be used to inform the user which keys need to be pressed to get out of the presentation and to return to the main screen. In fact, you are going to have the Home button do double duty later on in the movie.

8. Select the NavButtons layer and open the InterfaceElements folder in the library.
9. Drag the Call movieclip to the stage and place it on the horizontal guide on the stage and between the two vertical guides. In the Property inspector, give it the instance name of Call\_btn. As you may have guessed, this button will be a “speed-dial” button.
10. Drag the Photos movieclip to the stage and place it directly above the Call button on the NavButtons layer. Give this button the instance name of Photos\_btn.

*Yeah, yeah. We know. Why is one button a button symbol and the other a movieclip? Either symbol type can be used, so we decided to use one of each in this exercise. In fact, avoiding button symbols in mobile interfaces is a good thing in certain cases. Button symbols change when they are used, but that assumes a mouse action. In the mobile space, a button, if you decide it is necessary, will put a border around the object to give the user a visual clue that it is live. This explains why the button only has an up state.*

11. Select the Open layer and drag the OpenImage symbol from the library to the stage. Using the Property inspector, set its X and Y coordinates to 16.4 and 91. Save the project. As shown in Figure 12-18, you have created the opening screen of this movie.



**Figure 12-18.** The main screen has been assembled.

You could also test the app at this point to see how the interface looks in the device.

## Adding the gallery

With the main screen in place, we can turn our attention to the gallery of images that will display. Follow these steps to put it in place:

1. Add a keyframe in frame 10 of the Labels layer. Name the label Gallery and insert a keyframe in frame 10 of the Thumbs layer.
2. You aren't going to need to see the buttons or the start image. To remove the image, add a blank keyframe to frame 2 of the Open layer and another blank keyframe to frame 9 of the NavButtons layer.
3. Open the InterfaceElements folder in the library and drag a copy of the Thumbs movieclip to frame 10 of the Thumbs layer.

If you open the Thumbs movieclip in the Symbol Editor, as shown in Figure 12-19, you will see that each image is, in fact, a movieclip. Each movieclip is also given an instance name because the instance will be used to show a larger version of the image. The individual images can be found in the ThumbImages folder in the library.



**Figure 12-19.** The primary navigation tool will be a series of movieclips.

4. Add a blank keyframe to frame 11 of the Thumbs layer.
5. Add a keyframe to frame 20 of the Labels layer and name the label Images.
6. Add a keyframe to frame 20 of the Images layer and drag a copy of the ImageStrip movieclip to the stage. Line up this movieclip with the top of the stage and against the left guide. Give the movieclip the instance name of mcImages.
7. Finally, add a blank keyframe to frame 20 of the Head layer. Save the project.

Now would be a good time to review the rules regarding the use of bitmaps and vectors on mobile devices. The thing to keep in mind is you are dealing with a small screen area and a device whose computing power and memory are a fraction of those of a laptop computer. Though Flash Lite can render both types of graphics, there are a number of factors to consider when adding graphics to a movie destined for mobile playback.

Though Flash adores vectors and their use will reduce file size, they have their problems. Compared to bitmaps, vector graphics require more processing power to render, especially vector graphics that have many complex shapes and fills. Consequently, heavy use of vector shapes could slow things down. On the other hand, bitmap graphics do not require as much processing time to render as vector graphics and just might be a better choice for such things as a route map meant to be animated and scrolled on a mobile phone.

When authoring mobile presentations that use graphics, keep the following in the back of your mind:

- Avoid using outlines or strokes on vector shapes. Outlines have an inner and outer edge (fills have only one) and are twice the work to render.
- Corners are simpler to render than curves. When possible, use flat edges, especially with very small vector shapes.
- Optimization is especially helpful with small vector shapes such as icons. Complex icons may lose their details upon rendering, and the work of rendering the details is wasted.
- Import bitmap graphics at the correct size; don't import large graphics and scale them down in Flash, because this wastes file size and runtime memory. All of the graphics used in this exercise were created using the batch processing feature of Fireworks CS3 to allow us to import them into Flash size rather than scaling them.
- Flash Lite Player does not support bitmap smoothing. If a bitmap is scaled or rotated, it will pixelate. If it is necessary to scale or rotate a bitmap, consider using a vector graphic instead.
- Text is essentially just a shape. When text is needed, avoid animating it or placing it over an animation. Consider using text as a bitmap.
- Minimize, if not avoid altogether, the use of transparency in PNG files; Flash must calculate redraws even for the transparent portions of the bitmap. This will slow things down big time.

## “Wiring it up” with ActionScript

With the assets in place, you can now turn your attention to writing the ActionScript that brings this presentation to life. As we said at the start of this exercise, you can only use ActionScript 2.0 with the Flash Lite Player version we are using. As well, we are going to need to use some specific code to ensure the soft keys return us to the main screen or quit the presentation.

1. Add a keyframe to frame 1 of the Actions layer. Open the ActionScript editor and enter the following code:

```
stop();

_focusrect = true;

fscommand2("SetSoftKeys", "Options", "Exit");
fscommand2("SetQuality", "high");
fscommand2("Fullscreen", "true");
```

There is a lot of new stuff here, so we'll explain it so you understand what you are doing. The first line stops the playback head on frame 1, and the remaining lines do some housekeeping.

When an object is selected on a device, a yellow border appears around it. This border is called the **focus rectangle** and is moved around the screen using the up or down keys on the four-directional pad on the device. The `_focusrect` property turns this rectangle on (true) or off (false). The default value is true.

The three `fscommand2()` functions are how your movie communicates with the device's operating system. The first function relabels the two soft keys to Options and Exit. This means when the key is pressed, the function associated with that key is executed. You added the two labels at the bottom because this movie will be displayed in full-screen mode. When you do this, you have to manually add the labels.

The next two functions set the rendering quality of Flash Player to high and forces the player to display the movie in the full screen.

2. Press Enter (PC) or Return (Mac) and add the following code:

```
if (selectedItem == null) {
    Selection.setFocus(Photos_btn);
}
else {
    Selection.setFocus(selectedItem)
}
```

This conditional statement makes sure something—the `Photos_btn`—is always selected on the stage. It is always a good idea to give your user a visual clue that something is selected on the stage.

3. Press Enter (PC) or Return (Mac) twice and enter the following:

```
Photos_btn.onPress = function() :Void {
    selectedItem = this;
    gotoAndStop("Gallery");
};

Call_btn.onPress = function():Void{
    selectedItem = this;
    getURL("tel: 4165552365");
};
```

These are the button functions. The first one scoots the playback head to the frame labeled “Gallery.” The second one uses a `getURL` method to actually dial the phone. You can’t just toss in a number. You must use the `tel:` protocol. Now you know how to add a speed-dial function to a mobile movie.

4. Press Enter (PC) or Return (Mac) and enter the following code, which creates the listeners for the soft key events:

```
Key.removeListener(myListener);

var myListener:Object = new Object();

myListener.onKeyDown = function() :Void {
    var keyCode = Key.getCode();
    if(keyCode == ExtendedKey.SOFT1) {
        gotoAndStop("Main");
    }
    else if(keyCode == ExtendedKey.SOFT2) {
        fscommand2("Quit");
    }
};

Key.addListener(myListener);
```

You first remove any listeners that may be in place and create the Listener object. When the user presses the left soft key, the playback head is sent to the frame labeled “main,” and if the right soft key is pressed, the application is closed.

5. Check your syntax, and then save and test the movie. When Device Central opens, the Photos button is surrounded by a focus rectangle, and if you mouse over the keys, they will change color to show you they are active (see Figure 12-20).



**Figure 12-20.** The focus rectangle is visible, and the keys are “hot.”

6. The next bit of code makes the buttons in the Thumbs movieclip active. Add a keyframe in frame 10 of the Actions layer and open the ActionScript Editor. Enter the following code:

```
stop();

_focusrect = true;

fscommand2("SetSoftKeys", "Options", "Exit");
fscommand2("setquality", "high");
fscommand2("fullscreen", "true");
```



```

if (selectedItem == null) {
    Selection.setFocus(Photos_btn);
}
else {
    Selection.setFocus(selectedItem)
}

Photos_btn.onPress = function():Void {
    selectedItem = this;
    gotoAndStop("Gallery");
};

Call_btn.onPress = function() :Void {
    selectedItem = this;
    getURL("tel: 4165552365");
};

// remove any previously assigned listeners
Key.removeListener(myListener);

// create key listener object and assign onKeyDown handlers to it
var myListener:Object = new Object();

// listener actions
myListener.onKeyDown = function() :Void {
    var keyCode = Key.getCode();
    if(keyCode == ExtendedKey.SOFT1) {
        gotoAndStop("Main");
    }
    else if(keyCode == ExtendedKey.SOFT2) {
        fscommand2("Quit");
    }
};

Key.addListener(myListener);

function showImage(num:Number):Void {
    selectedItem = this;
    gotoAndStop("Images");
    mcImages.gotoAndStop(num);
}

for (var i:Number = 0; i < 10; i++) {
    mcImageSet["Button0" + i].onPress = function():Void {
        showImage(this._name.substr(7, 1));
    }
}

```

We know that was a lot of typing. In fact, you could have copied and pasted the code from frame 1 into this frame, and then entered the function at the end. It checks to see which movieclip has been selected and uses an array for the instance names of the clips that are the buttons to be clicked. The array access operator ( [ ] ) converts that string into an actual object reference.

The next code block makes use of the `String.substr()` method, which takes a given string and lets you give it a haircut. In this case, the `this._name` expression refers to the `MovieClip._name` property of the thumbnail movieclip “buttons.” In a case-by-case basis, inside the for loop, `this._name` returns “Button01”, “Button02”, and so on, and these values are the actual instance names of the buttons. Applying the `substr()` method to that string, you’re basically saying, “Go seven characters in and give me one character from that point.” That snips out the final character, which is a number—and that number gets passed to the `showImage()` function.

The beauty of this approach is that instead of writing nine separate button functions, you bundle the button functions into one function.

Each button function tosses the focus rectangle around the selected button—`selectedItem = this`—and the rest of the function scoots the playback head to the `Images` frame and then to the frame where the image is located in the `mcImages` movieclip.

*If you are “old school,” then be our guest and write nine separate button functions. The first one would be*

```
mcImageSet.Button01.onPress = function():Void {
    selectedItem = this;
    gotoAndStop("Images");
    mcImages.gotoAndStop(1);
}
```

*All you need to do for the remaining eight buttons is to copy and paste this code into the Script pane and change the button number and the frame number in the last line.*

- 7.** Check your syntax to be sure there are no errors. If there are no errors, copy all of the code to line 41—`Key.addListener(myListener);`—to the clipboard.
- 8.** Save and test the movie in Device Central. You will start in the main Screen, and the Photos button will be highlighted. Click the center button in the four-directional pad, and you will be taken to the Gallery frame. Press the right and left buttons on the four-directional pad, and each button in the interface will be given a focus rectangle as shown in Figure 12-21. Click the left soft key—Main—and you are returned to the main screen.



**Figure 12-21.** The keys on the keypad allow you to move around the interface and select the image to be viewed.

There is one last issue to address. Though the movie is fully functional, you may have noticed that when you pressed the Main soft key, you weren't taken back to the Gallery page. You went right back to the start. Obviously, this is not going to result in a good user experience. Let's fix that right now:

9. Add a keyframe in frame 20 of the Actions layer and open the ActionScript Editor. When it opens, paste the code on the clipboard into the Script pane. You are going to make one small change to the code.
10. Scroll down to the Listener object and change the word Main to Gallery. The Listener should now look as follows:

```
myListener.onKeyDown = function() {
    var keyCode = Key.getCode();
    if(keyCode == ExtendedKey.SOFT1) {
        gotoAndStop("Gallery");
    }
}
```

```
        else if(keyCode == ExtendedKey.SOFT2) {  
            fscommand2("Quit");  
        }  
    };
```

11. The last thing to do is to change the key label from Main to Back. Add keyframes in frames 19 and 20 of the SoftKeyID layer. Select the word Main in frame 20 and change it to Back.
12. Save and test the movie. Click an image button, and as shown in Figure 12-22, the image appears and the soft key ID is changed to Back. Click the left soft key, and you will be returned to the gallery.

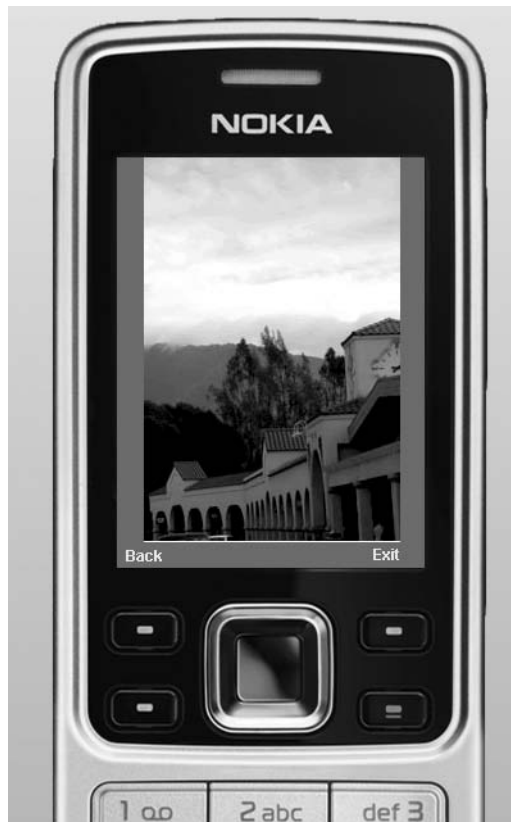


Figure 12-22. The project is complete.

## What you've learned

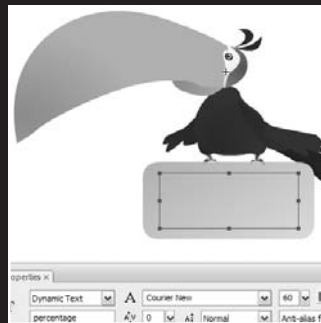
- How to use Device Central
- The process used to create content for a mobile application
- The process for testing a mobile application in Flash CS3
- How to plan and execute a mobile application
- The basic ActionScript to control a mobile application

This chapter only scratched the surface of the mobile market that is rapidly opening up to Flash developers around the world. Device Central is “Ground Zero” if you are at all serious about mobile, and we think you can see how important it is to your efforts. Mobile is a totally different space from that which you use every day, and the one thing to keep in mind is that mobile content is the only content you will create that the user actually pays for through network access charges. This is why it is so vitally important that you squeeze out every extra bit and kilobyte from the application before it goes live.

Speaking of squeezing and small, the next chapter shows you how to turbo charge your Flash movies. See you there.



## 13 OPTIMIZING FLASH MOVIES



One of the most common user experiences, when it comes to Flash on the Web, is sitting around waiting for the movie to start. From your perspective, as the developer who designed the site, this is an odd situation to encounter because, when you tested the movie, it was seriously fast and played flawlessly. What happened? To be succinct: the Web happened. Your movie may indeed be cool, but you made a fundamental mistake: you fell in love with the technology, not the user.

What we'll cover in this chapter:

- How Flash movies are streamed to a web page
- Using the Bandwidth Profiler to turbo charge movies
- Optimizing Flash movies
- Creating a simple preloader
- Converting a Flash movie to a QuickTime video

Files used in this chapter:

- YawningParrot.flc (Chapter13/ExerciseFiles\_CH13/Exercise/YawningParrot.flc)
- BandwidthTest.flc (Chapter13/ExerciseFiles\_CH13/Exercise/BandwidthTest.flc)
- BandwidthTest1.flc (Chapter13/ExerciseFiles\_CH13/Exercise/BandwidthTest1.flc)
- Chill.mp3 (Chapter13/ExerciseFiles\_CH13/Exercise/Chill.mp3)
- Loading.flc (Chapter13/ExerciseFiles\_CH13/Exercise/Loading.flc)
- LoadingJPG.flc (Chapter13/ExerciseFiles\_CH13/Complete/Exercise/LoadingJPG.flc)
- LoadingSWF.flc (Chapter13/ExerciseFiles\_CH13/Complete/Exercise/LoadingSWF.flc)
- PreloadEX.flc (Chapter13/ExerciseFiles\_CH13/Exercise/PreloadEX.flc)
- YawningParrotPreloader.flc (Chapter13/ExerciseFiles\_CH13/Exercise/YawningParrotPreloader.flc)
- BubblingLettersFinal.mov (Chapter13/ExerciseFiles\_CH13/Exercise/BubblingLettersFinal.mov)

## Flash's “love-hate” Internet relationship

Back in the early days of Flash, when we really didn't know better, Flash designers would prepare these really “cool” intros to a site, which played while the rest of the site loaded. The problem was they were overly long, and in many cases the intro seemed to take almost as long to load as the site. The solution was the infamous Skip Intro button. A couple of seconds after the intro started playing, the Skip Intro button would appear, and users would click it only to discover the site hadn't quite loaded, so they would sit there



drumming their fingers on their desk. It became so ludicrous, users would see the button not as a Skip Intro button but as a “Skip Site” warning. This resulted in Flash gaining a rather nasty reputation for “bloat,” which it still hasn’t shaken.

*If you are interested, the Flash community does have quite a sense of humor, and one of the more popular Flash sites of the time was named “Skip Intro” (see Figure 13-1). You can still visit it at [www.skipintro.nl/skipintro/skipintro98.htm](http://www.skipintro.nl/skipintro/skipintro98.htm).*



**Figure 13-1.** Welcome to “Skip Intro” hell.

To deal with the bloat issue, it is critical that you comprehend the underlying technology behind your Flash movie. This means you need to understand what the Web really is and become familiar with many of the terms commonly used in the Flash designer and developer community.

## This “Internet” thing

The Internet’s roots go back to the U.S. Department of Defense’s need to create a bullet-proof means of maintaining communications between computers. This involved such things as file transfers, messaging, and so on. At the time, computers were a virtual Tower of Babel, which meant different computer types and operating systems rarely, if ever, could talk to each other. As well, in battle conditions, the system needed would have to carry on even if a piece of it was knocked out, and it had to be accessible to everything from portable computers to the big, honking mainframes in “clean rooms” around the world.

The solution was an enabling technology called **Transmission Control Protocol/Internet Protocol**, though we know it by a far sexier name, **TCP/IP**. This is how data moves from your computer to our computers or from your web server to our computers, and, as you may have guessed, the slash indicates it comes in two parts.

**Internet Protocol (IP)** is how data gets from here to there by using an address called the **IP address**. This address is a unique number used to identify any computer currently on the Internet. What it does is to create little bundles of information, called **packets**, that can then be shot out through the Internet to your computer. Obviously the route is not a straight line. The packets pass through special computers called **routers**, and their job is to point those packets to your computer. Depending on the distance traveled, there could be any number of routers that check your packets and send them either directly to your computer or the next router along the line.

The **Transmission Control Protocol (TCP)** part of the process is the technology that verifies all the data packets got to your computer. The thing is, the IP portion of the trip couldn't care less if packet 10 arrives at your computer before packet 1, or that it even got there at all. This is where TCP comes in. Its job is to ensure that all of the packets get to where they are supposed to go.

Once all of the kinks got worked out, the military had quite the communications system on its hands.

## Enter the World Wide Web

The Web is a network of networks. It came about because people realized straight data transmission was interesting, but once the cool factor wore out its welcome, they started wondering how it would be possible to use this communication network to access files containing images, audio, and video.

The solution was the World Wide Web, which is commonly seen as web pages and hyperlinks. A web page is a simple text file that uses HTML—a system of tags and text—to define how a page should look and behave. This is important because your Flash movies are always found in an HTML wrapper.

*If you are a history buff, the concept of hyperlinks and hypertext was around long before the Internet. The gentleman who managed the atomic bomb project for the U.S. during World War II, Vannevar Bush, wrote an article for the Atlantic Monthly in July 1945 that proposed a system of linking all information with all other information. The article was entitled “As We May Think,” and you can still read it at [www.theatlantic.com/doc/194507/bush](http://www.theatlantic.com/doc/194507/bush).*

An HTML page may only be a text file, but it can contain links to other assets such as your Flash SWF. These links take the form of a **Uniform Resource Locator (URL)** and specify the location of the assets. When Firefox or Internet Explorer translates the page, those addresses are used to load the assets and display them on the computer screen. Thus the Web is really composed of two parts: browsers and servers, which are the computers that hold the assets and shoot them to your computer when the browser asks for them.

As you can see, the movement of your SWF file from here to there is a rather uncomplicated process. Where your pain and headache comes into play is through something called **bandwidth**.

## Bandwidth

In the early days of Flash, around the year 1999, one of the authors read an article written by a New York Flash designer—Hillman Curtis—and one phrase leaped out of the article and has been glued to the front of his cerebral cortex ever since. The phrase? “Keep an eye on the pipe.”

The “pipe” is bandwidth. Bandwidth is a measure of how much data will move along a given path at a given time or how much information can be downloaded through a modem and how fast. One of the authors, when speaking to this topic at conferences or in classes, uses a rather amusing graphic analogy that will help you to understand this topic. Imagine trying to push the amount of data contained in your favorite TV show through a modem that is connected to a phone. Trying to push that amount of data through a dial-up modem is no different from “trying to push a watermelon through a worm.”

Bandwidth is measured in bits (usually kilobits) per second. A bit is either a 1 or a 0, so bandwidth is a measure of how many 1s and 0s can be fed through a modem each second. The higher the number, the greater the bandwidth, and the faster things get from here to there. The thing is, bandwidth is not constant. It requires more bandwidth to move a video from here to there than this page of text that you are reading. The issue is not “here to there.” The issue is the modem’s capacity to move the data. This is the “pipe.” Users with 56K dial-up modems have a pipe that has the diameter of a garden hose. Users with cable modems have a pipe the diameter of a fire hose. Connect the garden hose to the fire hydrant in front of your house, and you will get a graphic demonstration of data flow and the “pipe” when you turn on the hydrant.

As we pointed out earlier, the data packets sent to your computer get there, eventually, and the route is never a straight line. Over time, the TCP/IP model nudges the transmission rate toward a relatively predictable rate, but this is technology we’re dealing with here, and it is the prudent Flash designer who approaches technology with a dose of pragmatism and does not assume a constant flow. This has implications behind your design efforts, and we will get into those shortly.

You need to regard the pipe and data transmission in much the same manner you regard your local highway. It may have six lanes for traffic and a posted speed limit of 60 mph or 100 kph, but that all becomes irrelevant in rush hour. Traffic moves at the pace of the slowest car. It is no different with the Internet. Servers can become overloaded.

The best example of that is the infamous tragedy often referred to as 9/11. On that day, the Internet essentially ground to halt as, it seemed, every computer on the planet was attempting to get the latest information. What people overlooked on that day was that a server is only a computer and can only reply to requests for information at a set rate. If the browser can’t get the information, it will eventually assume the assets are not there, and the requested page will either not be displayed or will be displayed with information missing. It got so bad on that day for CNN and the BBC, for example, they were forced to post a message that essentially told people “come back later.” People lucky enough to get connected

experienced pauses in the download and disconnection, which are the hallmarks of an overloaded server.

What you need to take away from this “horror story” is that the time it takes to download and play your Flash movie is totally dependent on the contents of your Flash movie and traffic flow on the Internet. This means you need to concentrate on not only what is “in” your movie, but also who wants to access it. This is where you fall in love with the user and not the technology.

## So who are these folks we call users?

The Flash community is an odd-ball collection of people ranging from those who ride skateboards for entertainment to the classic “nerd” working in a corporate cubicle farm. This disparity, which actually is the strength of the Flash community, has resulted in a bit of a split between those who use supercharged computers to develop their content and take a “sucks to be you” attitude if you can’t revel in their work and the corporate types who work within the strict standard set by their IT department. This standard is usually in the form of the following Commandment: *Thou shalt develop to a Flash Player 7 standard, and may whatever god you worship have mercy upon your miserable soul if you step outside of this stricture.*

Which begs the question, What do you need to know before putting your work out there?

Here are some general guidelines:

- Small means fast. Studies show you have 15 seconds to hook the user. If nothing is happening or is appealing to the user, that user is gone. Small SWFs mean fast download. The days of eye candy at the start of a Flash movie are over. If the content users see within that 15-second window is not relevant to the site or the experience, they will be gone.
- If a bleeding edge Flash site isn’t viewable on a two-year-old computer with a standard operating system and hardware, it is time to go back to the drawing board.
- For a commercial site, give yourself another year and go back three years. Corporations are relatively slow to upgrade because of the significant cost incurred to do so.
- If your target audience is urban and in a developed country, assume they have, at minimum, a cable connection.
- If your audience is the world, develop to the lowest common denominator, which is a dial-up modem.

Now that we have inundated you with “techie talk,” let’s look at how your Flash file gets from here to there. How that happens is through a concept we talked about in Chapter 8—*streaming*.

## Streaming

As you have discovered by this point in the book, simply tossing a bunch of audio, images, and video into your movie is not a good thing. They take an inordinate amount of time to download. In fact, toss all of that content into frame 1, and you can kiss your “15-second window of opportunity” goodbye.

It is the prudent Flash designer or developer who realizes this and structures a movie in such a way that it starts playing before the content has finished loading. Like video, your Flash movie uses a progressive download process. This means when enough data has been downloaded to start something happening, that 15 seconds is a godsend.

Streaming doesn't make things faster. What it does is intelligently organize things to start the movie playing in very short order. Used wisely, streaming can ensure that everything in the Flash movie is downloaded before it is needed. The result is a Flash movie that seems to start playing almost immediately and plays “as smooth as the hair on a frog's back.”

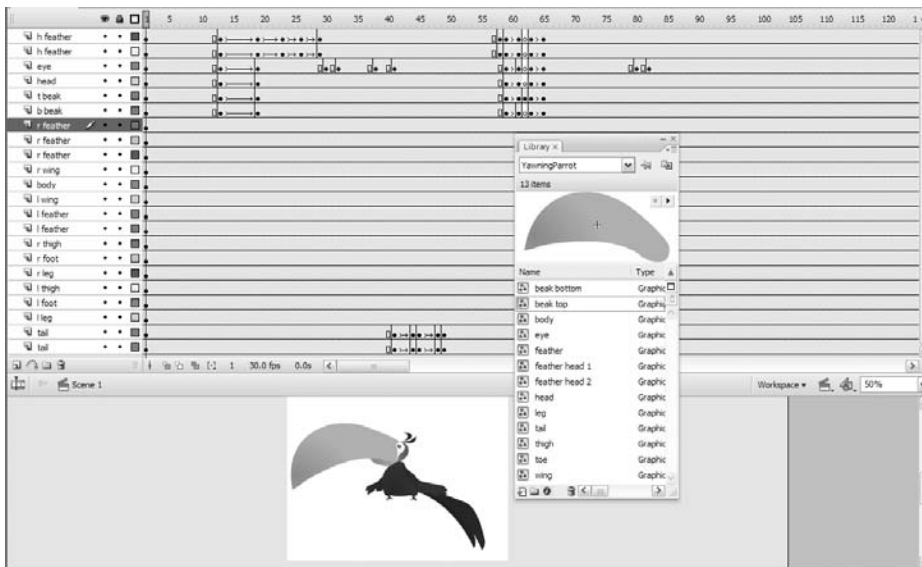
So what happens when a web page requests your movie? Two things are sent to the browser:

- The movie's timeline, including ActionScript and stuff not in the library, such as text and shapes that haven't been converted to symbols
- Library items, including audio, video, images, and symbols—if they appear in the timeline or are set to export for ActionScript

You need to clearly understand this: when your Flash movie is shot through the Internet to the user's browser, the movie is sent in frame order. If the movie is split into scenes, a relatively rare practice today, the scenes will be sent in the order in which they appear. The library is also sent, but the library items are not sent in the order in which they appear in the library. They are sent in the sequence in which they appear on the timeline. To reinforce what we have just said, let's take a look at a typical file:

1. Open the `YawningParrot.fla` file in your Chapter 13 Exercise folder.

The timeline, shown in Figure 13-2, is linear, but you will notice there are a lot of layers. Your first reaction could be, “Man that is going to take a while to load.” Not really.



**Figure 13-2.** Streaming plays a movie in frame order and loads the content in the library in the order in which it appears on the timeline.

2. Open the library. You will notice there is a lot less content in the library than there are layers. This is because the symbols in the library are reused and repurposed. All of the feather layers use the feather symbol.

When this movie loads, the parrot is constructed, in frame 1, of all of the objects in the library. The thing about each of the library objects is they are physically small. This means they are also relatively small in file size, and the result is not a lot of bandwidth is required to load them and get the movie playing.

To visualize how this movie will stream, you will need to add an extra, imaginary playhead to the timeline. When the movie starts, they are both in frame 1, but the imaginary playhead—let’s call it *streamhead*—moves ahead of the actual playhead. The streamhead’s position on the timeline indicates how much of the movie has been downloaded. The playhead will stay put because its purpose is to indicate the frame currently playing.

Now let’s assume that you toss in a movieclip containing a 3-second FLV file that has been embedded into its timeline, and this movieclip is added to frame 10. The odds are really good that the streamhead will stay put on frame 10 and the playhead will “catch up” to arrive at frame 10 and also stay put. What will happen is the movie will essentially stop until the movieclip’s inner timeline, including the FLV, is loaded, at which point the streamhead restarts its journey along the timeline.

To avoid this nastiness, Flash designers and developers give the streamhead a head start by creating a **streaming buffer**. This could be in the form of a preloader (we’ll get into this subject later) or any other technique that keeps the playhead behind in “second place” in order to let the streamhead do its job and load content.

Though you may be having difficulty visualizing two heads on the timeline, Flash has a tool that lets you see how these two heads work and how the pipe can affect the delivery of your Flash movie to the browser. Where is this tool? Let’s go look at the Bandwidth Profiler.

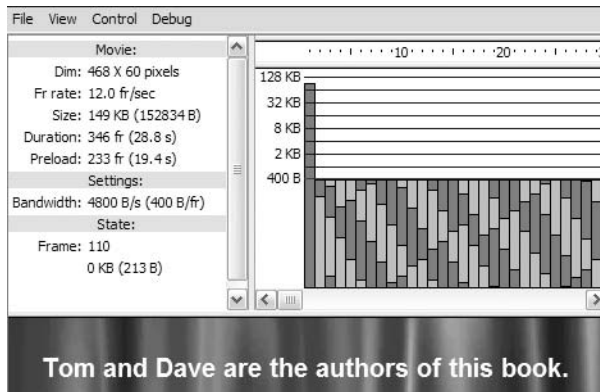
## The Bandwidth Profiler

In many respects, the Bandwidth Profiler is quite similar to what you see in Device Central when you test a mobile movie. The movie opens up in a device that emulates the performance of your movie in the chosen device. The Bandwidth Profiler emulates how your movie will behave when it downloads to the user’s machine.

Though the Bandwidth Profiler is an extremely useful tool, keep in mind it is nothing more than an emulator. It won’t mimic real life completely, mainly because it assumes a steady transfer rate into the browser, in contrast to an Internet universe that actually ebbs and pulses based on traffic.

Regardless, the Bandwidth Profiler can give you a good idea of where streaming bottlenecks are likely to occur. This can be an invaluable aid in relieving “data jam” and solving a problem before it becomes a major one. Let’s take a look at the Bandwidth Profiler:

1. Open the `BandwidthTest.fla` file in Flash. When the movie opens, you will see we have placed an audio file on the timeline, embedded an FLV into a movieclip, and placed it in frame 1, and if you scrub over to frame 2, you will see we have added some text to the stage. If you open the library, you will see the text is actually a graphic symbol. Just by looking at the timeline, you can see that the movieclip with the FLV and the audio file will be the first pieces of content to load, and then the text will load.
2. Test the movie. This time when the SWF opens, select `View > Bandwidth Profiler`. Take a look at the graph, shown in Figure 13-3, that suddenly appears above the movie.



**Figure 13-3.** The Bandwidth Profiler

On the left side are three headings: `Movie`, `Settings`, and `State`. To the right is a frame-by-frame representation of the data downloading into each frame. Notice the spike in frame 1. This is understandable because the audio file and the FLV need to load in this frame. Under the settings, you will see something like `Bandwidth:4800B/s(400 B/frame)`. Now look at where that red line is sitting. The values match. This red line is the **bandwidth limit**, which represents the maximum throughput a modem can handle. Bars under the line are handled quickly. Bars rising above the line indicate potential bottlenecks.

3. Select `View > Download Settings`. The drop-down menu that appears (see Figure 13-4) allows you to choose a particular modem speed.
4. Select `DSL (32.6 KB/s)` from the drop-down menu and scroll back to the start of the movie. You will notice the bandwidth limit has increased from 400 bytes to 2.78 KB and that the markings on the graph have changed to reflect your selection.
5. You are most likely looking at that spike in the first frame and thinking, “Yeah, so? What’s the deal?” Rather than having us explain it, we are going to let you experience it. Change back to the 56K modem choice and this time select `View > Simulate Download`.

Let’s guess. You sat around for about 25 seconds waiting for the movie to start? What you have just experienced is the other, and most important, half of the Bandwidth Profiler. You just sat through what a person with a 56K modem will experience. Let’s take a minute and talk about this.

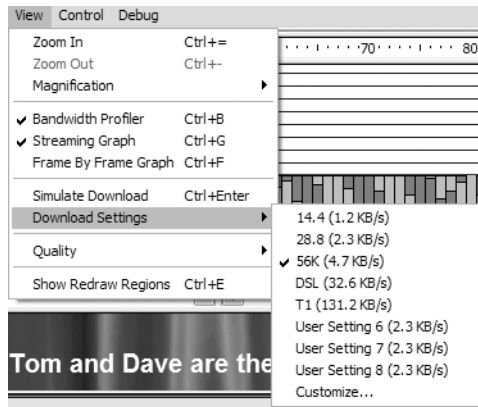


Figure 13-4. You can change the modem speed.

When you selected Simulate Download, you essentially emulated how the movie will load into a 56K modem. The other thing that happened is the Bandwidth Profiler developed a green bar at the top of the graph, which stayed put until the movie started to play (see Figure 13-5).

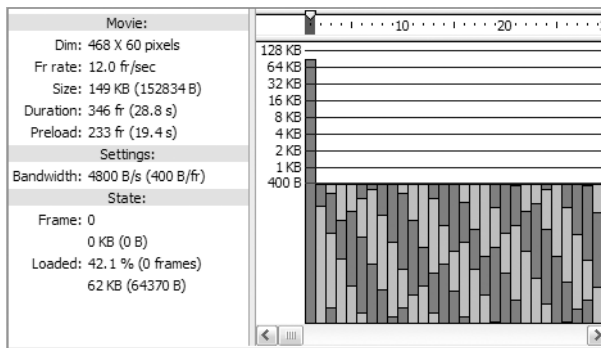


Figure 13-5. You can experience the issue with the first frame when you simulate the download.

The green bar is important. At the start of this section, we talked about two imaginary playheads. What you are seeing is what happens when the streamhead and the playhead catch up to each other. To visualize what is going on, think of the playhead as the playhead indicator at the top of the screen and the streamhead is the green bar. How long will they be stuck there?

On the left side of the Bandwidth Profiler is a value for the Preload setting, which in our example is 233 fr (frames) or 19.4 s (seconds). When you starting emulating the movie, the Settings area became active (you may have to increase the size of the Bandwidth Profiler window by dragging the bottom edge down). The size area will show you the activity, and you can see that it will take just over 19 seconds to load in all of the content in the first frame. What you should have seen, when everything loaded, was the green bar suddenly



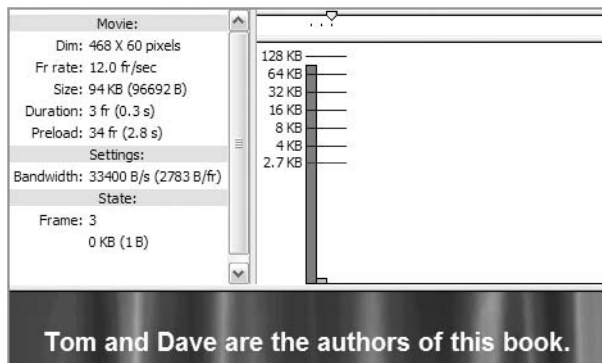
roaring off to the right, and the playhead moving across the frames as it follows the streamhead.

6. Change the download settings to DSL and select Simulate Download. You'll still experience a short delay, but there will also be a marked decrease in how long you have to wait. The Preload setting should show you 33 fr (2.8 s), which is a dramatic decrease from the almost 20 seconds you had to wait simulating a 56K modem.

As you can see, the Bandwidth Profiler is a rather powerful tool that you need to master. With it, you can tailor your movie to the bandwidth constraints of your user and ensure you meet that 15-second window of opportunity that will open to you. With the Download Settings option, not only do you get to see how bandwidth will affect your movie, you also actually get to experience it.

At this point, you may be thinking, "Shoot, I can cut back the preload by using ActionScript to play the sound because the sound is embedded into the library." Let's see if that works:

7. Open the BandwidthTest1.fla file. If you open the library, you will see the sound file is absent, and if you open the code in frame 3 of the Actions layer, you will see we have added the call to the sound.
8. Test the movie and select Simulate Download. The graph, as you see in Figure 13-6, has significantly changed, but the spike in frame 1 really hasn't. As well, the delay is still there. This tells you the issue really isn't the sound, but the FLV embedded into the movieclip. (The sound wasn't adding a lot up front because its Sync setting was set to Stream, which distributed its weight across many frames.) You have just discovered another use for the Bandwidth Profiler. Not only can it show you where the problem is, but it can also be used to isolate the content causing the delay.



**Figure 13-6.** Use the Bandwidth Profiler to identify the content causing the delay.

*How would we fix this? First off, there will always be that spike in frame 1 of any movie you will create. The goal is to get that spike as close to the red line as possible. To fix this, one approach would be to reduce the time of the curtains effect from its current 46 frames to 23 frames in the background movieclip. Do this, and the preload time drops to 1.4 seconds.*

## Optimizing and fine-tuning your Flash movies

As you saw in the previous example, a simple thing like reducing the number of frames in an FLV can have a dramatic impact on how the movie loads. What we plan to do in this section is to outline a few tips, tricks, and techniques you can use to make your Flash movies leaner, meaner, and faster. Surprisingly, the first mistake most people make is made even before a pixel is lit up. They forget to plan the movie.

### Structure

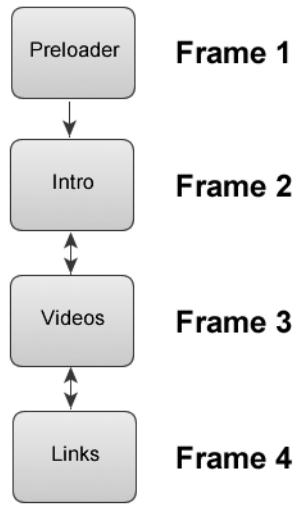
That old adage—*Plan your work and work your plan*—is especially true when working with Flash. You can't make it up as you go along. You have to take the time before you start to think about what the user sees, and in what order, before you start firing content into the library and then onto the stage. For example, a video site that lets the user choose from a number of videos would involve the following:

- Preloader
- Intro screen
- The main movie screen where the videos are chosen and viewed
- A set of links to other video sites you may have created

This means when the user arrives at the site, he or she would most likely do the following:

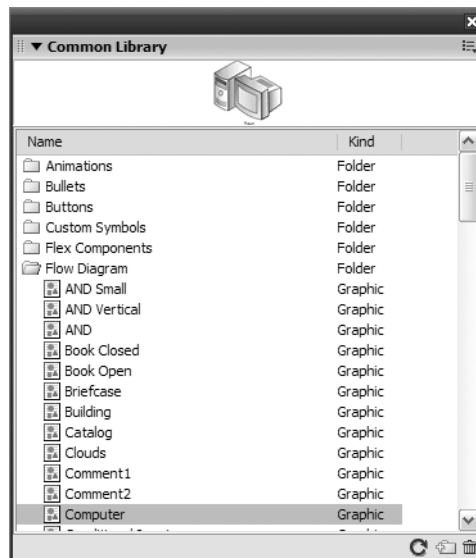
- The user would see the preloader for a few seconds and then be taken to the intro frame.
- From there, the user would choose to read the information and then move to the video picker screen by clicking a button.
- The video frame would load up, and the user can click a series of buttons to view the videos associated with the buttons.
- The viewer can then choose to return to the intro screen or go to a frame that contains a series of interactive links.

Now that you have an idea of what will happen, you might even want to pull together a small flowchart that shows the purpose of each frame in the movie. As you can see in Figure 13-7, having one of these handy allows you to visualize how the user will move around the movie and provides a broad view of the content of each frame.



**Figure 13-7.** Map out your plan.

In fact, if you have arrived at Flash CS3 through the Adobe Web Premium Bundle, you have an ideal tool for this process at your disposal. Fireworks CS3 has been repositioned as a rapid prototyping tool. If you open the application and select **Window** ► **Common Library**, you will see a bunch of folders that contain symbols for a variety of rapid prototyping tasks. The flow diagram symbols shown in Figure 13-8 are ideal for planning out a Flash or HTML project.



**Figure 13-8.** Use Fireworks CS3 as a planning aid.

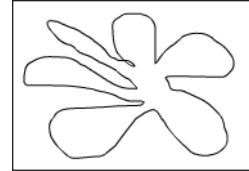
By writing out what each frame does or using Fireworks CS3 to create a flow diagram, what you are doing is ordering the content on the timeline. By “falling in love with the user” and streaming the content into the movie in that order, your site will meet the needs of your users. If you randomly place the content on the timeline, you have no way of ensuring it will load in any meaningful manner. The result is a site that has to download in its entirety before the user can interact with it. Though many sites do this, it is not considered to be a best practice within the Flash design community.

## Optimizing elements in the movie

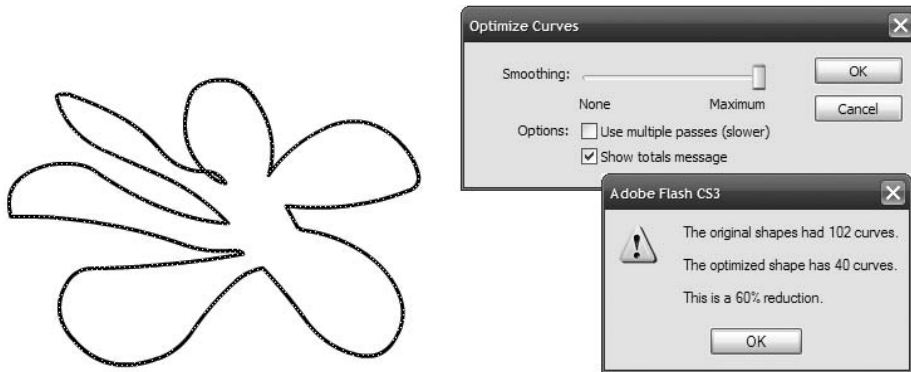
Every chapter in this book has directly or indirectly made it clear that Flash loves “small.” After your experiences with the Bandwidth Profiler, we think you now understand why we are so adamant on this point. A small file means a fast load. A fast load means short wait time. A short wait time puts you squarely in that 15-second window of opportunity. We have shown you several methods of keeping things small when it comes to images, fonts, sounds, video, and drawing. What about vectors?

We know Flash loves vectors. The thing is, vectors can be both small and large at the same time. Huh? The reason is every time Flash encounters a vector point, it has to load it into memory in order to draw the shape. If you create a vector with a large number of vector points, you may have a small file on your hands, but you have also increased the demand on memory to redraw the image. The result is the inevitable spike in the Bandwidth Profiler. Here’s one way of addressing this issue:

1. Create a new Flash document. When Flash opens, add three more keyframes to Layer 1 in the Layers panel. You now have four keyframes on the timeline.
2. Select the Pencil tool, and in frame 1, draw a “curvy” shape like we have done in Figure 13-9.
3. Copy your shape to the clipboard. Select each of the remaining three keyframes in Layer 1 and select Edit ► Paste in Place (Ctrl+Shift+V on a PC or Cmd+Shift+V on a Mac).
4. Select the shape in frame 2 and select Modify ► Shape ► Smooth. Not a lot seems to happen.
5. Select the shape in frame 3 and select Modify ► Shape ► Straighten. A couple of the lines straighten out.
6. Select the shape in frame 4 and select Modify ► Shape ► Optimize. This time you are presented with the Optimize Curves dialog box. Move the slider all the way to the right and click OK. The dialog box will close and be replaced by an Alert dialog box, shown in Figure 13-10, telling you how many curves were found, how many were optimized, and the size of the reduction as a result of the optimization.



**Figure 13-9.** Start by drawing a shape containing a lot of vector points.



**Figure 13-10.** Using shape optimization

*The image shown in Figure 13-10 is a composite image. We created it to show you the Alert dialog box resulting from clicking OK in the Optimize Curves dialog box.*

7. Test the movie. As you see in Figure 13-11, the graph shows you the file size of the content in each frame and the effect modifying the shape has in each frame. The results are quite dramatic.



**Figure 13-11.** Smoothing, straightening, and optimizing curves can have a profound effect upon download times.

You are most likely looking at the graph and thinking, “Wow, I am going to start optimizing all of my vector shapes.” Not so fast. Each of the three methods presented did a good thing and a bad thing. The good thing was they did indeed reduce the bandwidth load. The bad thing was they introduced distortions into the image. If you are happy with the distortions,

fine. If you aren't, you might want to consider doing the optimization, selecting the shape with the Subselection tool, and manually manipulating the shape and the points.

So why was there such a drop in the graph between the object in frame 1 and its counterpart in frame 4? Remember, vector nodes require bandwidth. You removed a ton of them, as shown in Figure 13-12, using the Shape Optimization dialog box, which accounts for the drop in required bandwidth.

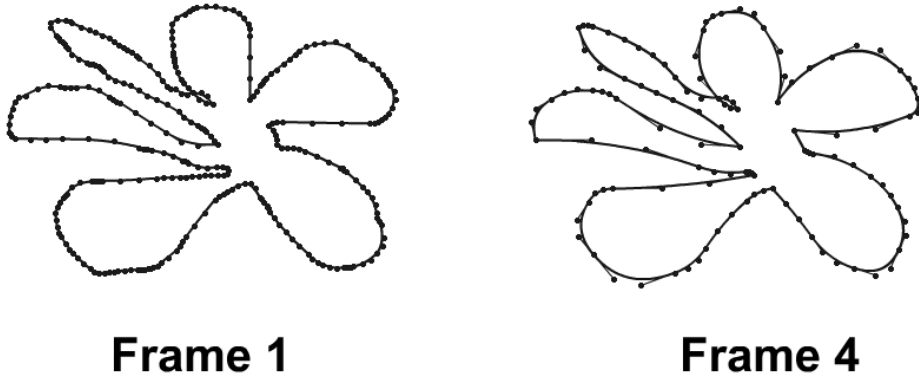


Figure 13-12. Each node on a shape requires a “piece” of bandwidth.

## Using the Loader class to display images and SWFs

As you saw in the Bandwidth Profiler exercise, using ActionScript to load content into a movie can result in a performance boost because the content is not contained in the library. Instead the data is streamed into the SWF from your HTTP server. This technique can also be applied to images and SWF files as well. Just keep in mind that content still needs to be streamed into the SWF that calls the content, meaning there will still be a slight delay, but the performance boost comes in the form of reduced wait times.

All of this is accomplished through the use of the Loader class in ActionScript 3.0. The Loader class is used to load SWF files or image (JPEG, PNG, or GIF) files into a SWF through the use of the load() method. The loaded display object is added as a child of the Loader object. Here's how all of this works:

1. Open the Loading.fla file in the Exercise folder. When the file opens, you will see there is a single movieclip on the stage, and if you open the library, you will see this movieclip is the only object in the library. If you select the clip in the stage, you will also see we have given it the instance name of clip.
2. Select the first frame of the scripts layer, open the Actions panel, and add the following code:

```
var loader:Loader = new Loader();
addChild(loader);
//clip.addChild(loader);
```

```

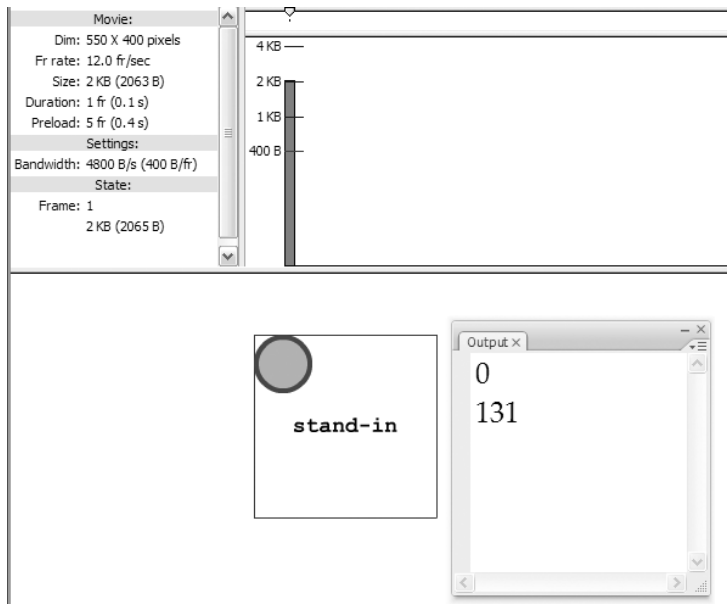
loader.contentLoaderInfo.addEventListener(
    Event.COMPLETE,function(evt:Event):void {
        //clip.x = 50;
        //clip.y = 50;
    }
);
loader.contentLoaderInfo.addEventListener(
    ProgressEvent.PROGRESS, function(evt:ProgressEvent):void {
        trace(evt.bytesLoaded);
    }
);
//loader.load(new URLRequest("toBeLoaded.swf"));
loader.load(new URLRequest("toBeLoaded.png"));

```

The first line of the code creates the Loader object. The next two lines either load the content onto the stage—`addChild(loader)`—or into the movieclip itself. The two `EventListeners` tell the loader where to place the content and to let you see the progress by using the `trace()` function to display how many bytes have been loaded in the Output panel.

The final two lines are used to tell the SWF what content is to be loaded.

3. Test the movie.
4. Return to the Actions panel and comment out lines 2 and 17 of the code (`//addChild(loader)`) and delete the comment in lines 3, 8, 9, and 18.
5. Test the movie. This time the SWF is called in, added to the stand in movieclip, and placed 50 pixels out and down on the stage, as shown in Figure 13-13.



**Figure 13-13.** A SWF is loaded into the movieclip on the stage, and the movieclip is moved to a new location. Note as well the Output panel shows you the `bytesLoaded` value.

*Final copies of these two movies—LoadingJPG.fla and LoadingSWF.fla—can be found in this chapter's Completed folder.*

*You may have noticed throughout this book that the layer where scripts are placed is either called Actions or scripts. What you are seeing is a clash of naming conventions. Ultimately, the name of the layer doesn't matter, as it has no practical effect on the functionality of the movie. This is not to involve you in a debate, but to make you aware that layer naming conventions need to be defined right at the start of a project. We aren't trying to be difficult by throwing two choices at you; we're following current industry best practice of giving layers a meaningful label—it's just that we didn't consult beforehand. Using one or the other is an acceptable best practice.*

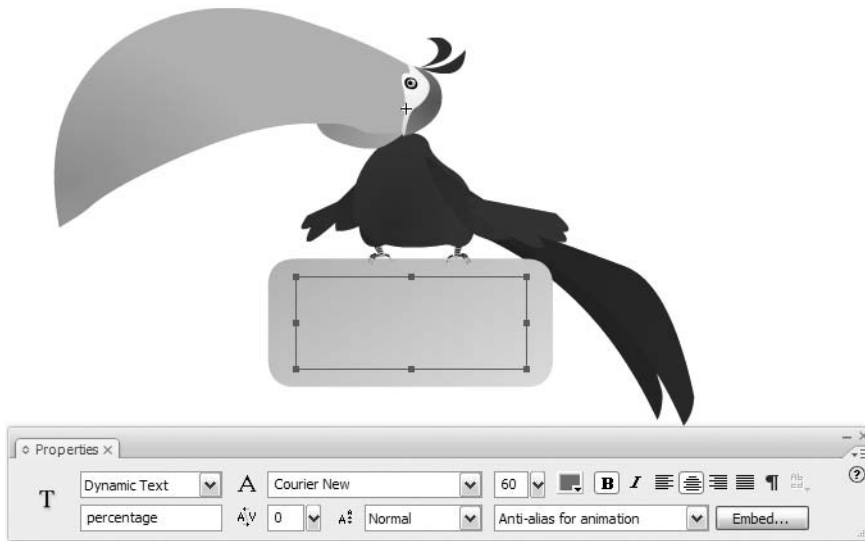
## Your turn: Creating a preloader

We have been talking about preloaders, and the time has arrived to create one of your own. In this exercise, you will not only create a preloader, but also use the bytesLoaded values to give the user some feedback regarding the loading of the file. Preloaders can range from the simple, such as the one you'll create here, to the incredibly complex, involving clever animation and even sound effects. Regardless of the approach taken, the purpose of a preloader is to get something happening within that 15-second window of opportunity and have the user become engaged in the site almost immediately. To do this, you will be using assets from the YawningParrot.fla file as the preloader for the file used in the Bandwidth Profiler exercise. Here's how:

1. Open the PreloadEX.fla file in your Exercise folder. Scrub through the timeline, and you will see the familiar yawning parrot and then be taken right into a clone of the heavy BandwidthTest1.fla file used earlier in this chapter.
2. Open the library and double-click the parrot movieclip to open it in the Symbol Editor. You will notice that we have added a dynamic text box with the instance name of percentage under the parrot. This text box, shown in Figure 13-14, will be used to display the percentage value of how much of the file has loaded.
3. Click the Scene 1 link to return to the main timeline. Add a keyframe to frames 1 and 5 of the scripts layer. Select the keyframe in frame 1 of the scripts layer and open the Actions panel.
4. Click in the Script pane and add the following code:

```
root.loaderInfo.addEventListener(
    Event.COMPLETE,
    function(evt:Event):void {
        if (currentFrame == 5) {
            play();
        } else {
            gotoAndPlay(10);
        }
    }
);
```





**Figure 13-14.** A yawning parrot and a dynamic text box are the key elements used in this preloader.

Let's take a moment and figure out what is going on. First off, this movie actually preloads itself. There is no external content being used. The “trick” therefore is to determine where to target the events used by ActionScript.

The first line of the code targets the `loaderInfo` property of the main timeline—`root`. Where does `loaderInfo` come from? This property hails from the `DisplayObject` class, which is in the inheritance chain for `Sprite` and ultimately the `MovieClip` class.

In frame 1, you “wire up” a `COMPLETE` event. A `COMPLETE` event is fired when the movie data has loaded successfully. In this case, you are saying, “If the data hasn't loaded, keep moving forward.” If the current frame is 5, that means all of the content hasn't loaded, and the parrot is yawning away. Instead of looping back to frame 1, you tell the playhead to keep going to frame 5 and `play()` from there. If not, because the user has already visited the page that contains this movie, it means that the content has loaded after all. In that case, you use the `gotoAndPlay(10)` method to bypass the parrot.

5. Click the keyframe in frame 5 of the scripts layer and add the following code:

```
stop();

root.loaderInfo.addEventListener(
    ProgressEvent.PROGRESS,
    function(evt:ProgressEvent):void {
        var percent:Number = Math.floor(evt.bytesLoaded /
evt.bytesTotal * 100);
        parrot.percentage.text = percent + "%";
    }
);
```

You know what the `stop()` method does, so let's concentrate on the listener and the function.

The first line tells the main timeline to listen for something that just happens to be the `loaderInfo` property of the main timeline. The next line uses the `ProgressEvent` class to keep an eye on when the load operation has started. A `ProgressEvent` is usually triggered when SWF files, images, or data are loaded into Flash Player. In the case of this movie, you are going to keep an eye on the data part of the equation because all of the stuff in the movie is in the library. One of the properties of the `ProgressEvent` class is `bytesLoaded`. This is great because Flash knows how many bytes there are in the movie, and as you saw in the previous exercise, it keeps track of how many bytes have been loaded at any point in time. Knowing this, you can then do the math to compare how many bytes have been loaded as a percentage of the total bytes in the movie. These calculations inevitably wind up with copious numbers after the decimal, so you strip them off using the `Math.floor()` method. One of the easiest ways to get a handle on this sort of thing is to actually do the math. Let's assume there are 275 bytes in the movie (`bytesTotal`), and you have loaded in 71 of them (`bytesLoaded`). Let's follow the numbers:

```
var percent:Number = Math.floor(71 / 265 * 100);
```

The result of  $71 / 265 * 100$  is 26.792, which you round down (`Math.floor()`) to a value of 26 for your `var percent:Number`.

Now that you have a number, you can use it. In this case, you take the number and use it in the dynamic text box with the instance name of `parrot` under the yawning parrot. As more data loads, this number will not only change, but also appear as a percentage to the user because you told ActionScript to tack a % sign after the number.

6. Close the Actions panel. Save the movie and test it.
7. When the SWF opens, change the download setting to 56K and select Simulate Download. The parrot fades in and yawns as it waits for the rest of the movie to load. While the parrot is yawning, you also see the percentage of the content that has loaded, as shown in Figure 13-15.

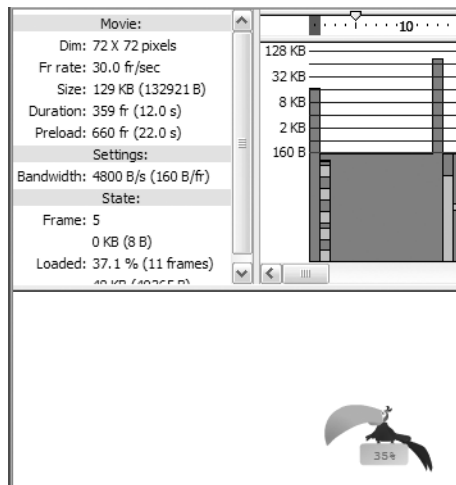


Figure 13-15. You are at the 35% percent mark.

What about those cases where content is loading from an external source? This could be the loading of a SWF or a JPEG image that is not embedded into a SWF. Here is how you do that:

1. Open the `YawningParrotPreloader.fla` file in your Chapter 13 Exercise folder. You will notice our bored friend is back, but this time in the library there is a blank movieclip named `photo`. In this exercise, you are going use the preloader to monitor the progress of a JPEG on another website loading into the `photo` movieclip. Not only that, but the photo loading from the other website will fade in.
2. Add keyframes to frames 15 and 30 of the `scripts` layer.
3. Select the keyframe in frame 15 of the `scripts` layer, open the `Actions` panel, and enter the following code:

```
stop();

var loader:Loader = new Loader();

loader.contentLoaderInfo.addEventListener(
    Event.COMPLETE,
    function(evt:Event):void {
        play();
    }
);

loader.contentLoaderInfo.addEventListener(
    ProgressEvent.PROGRESS,
    function(evt:ProgressEvent):void {
        var percent:Number = Math.floor(evt.bytesLoaded /
evt.bytesTotal * 100);
        parrot.percentage.text = percent + "%";
    }
);

loader.load(new
URLRequest("http://www.FoundationFlashCS3.com/rocket.jpg"));
```

There are three major differences between this code and the code entered into frame 5 of the previous example. The first is the playback head is told to stay put and not go anywhere until the image has loaded.

The next change is the use of the `contentLoaderInfo()` property. This property kicks out a `LoaderInfo` object, which is used to supply the loading progress information that is used in the dynamic text box under the parrot. The neat thing about that `LoaderInfo` object is that it comes into play before the content is fully loaded so the `addEventListener` is being used to monitor the progress of the download.

The final difference is the last line of the code. So far, you have used the `load()` method to bring in content that is in the same folder as the SWF. In this case, you are going to an external URL to grab the image and load it into the `photo` movieclip. If you have a lot of

images or SWFs to be loaded, this is a great way of having them all stay in one folder but still be accessible to your SWF.

4. Even though the photo of the rocket has been loaded, Flash doesn't have a clue where the photo is supposed to go. Let's deal with that. Add a keyframe in frame 30 of the scripts layer, open the Actions panel, and add this line of code:

```
photo.addChild(loader);
```

You used the `addChild()` method to tell Flash to add the content just loaded to the display list of the movieclip on the stage whose instance name is `photo`. The tween between frames 30 and 35 simply fades the image in from 0% alpha to 100%.

5. Save and test the movie. The parrot will tell you how much of the rocket has loaded, and then the rocket engine, as shown in Figure 13-16, fades in.

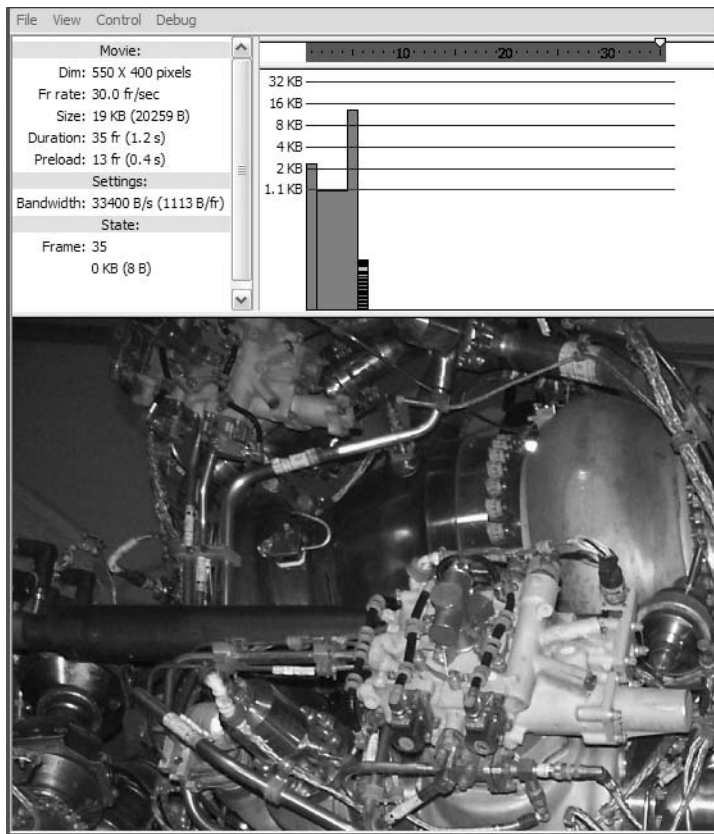


Figure 13-16. Loading external content into a movieclip

## Optimizing Flash content for use in video

As you may have surmised when reading the heading for this exercise, you are not going to be thinking “outside the box” but thinking outside of the room where the box is located.

This may come as a bit of a surprise, but Flash is often used to create animated cartoons that are shown on television and animations used in commercials. The process has historically been, to say the least, convoluted, but it could be done. The problem was, even though Flash could output to QuickTime, only the main timeline could be exported. Library content, ActionScript-driven animation, and even nested movieclips were not exported. When you consider the fact that the maximum length for the Flash timeline is just over 16,000 frames, the achievement of a short 10-minute cartoon was not possible unless the movie was broken into pieces and stitched together in a video editing program.

Flash CS3 contains a really cool feature that allows you to not only export the content on the timeline, but also create animations that are solely driven by ActionScript as QuickTime movies and then use them as motion graphics in such applications as Adobe After Effects CS3. In fact, in this exercise, the stage is blank. All of the letters that appear in this animation will be randomly generated, colored, and put in motion using ActionScript. Follow these steps to create a QuickTime movie in Flash:

1. Open a new Flash document and set the stage color to black.
2. Rename Layer 1 as scripts.
3. Select the keyframe in the scripts layer and open the ActionScript Editor.
4. Enter the following code:

```
var t:Timer = new Timer(50, 0);
t.addEventListener(TimerEvent.TIMER, createLetter);
t.start();

function createLetter(evt:Event):void {
    var f:TextFormat = new TextFormat();
    f.size = randomBetween(80, 120);
    f.color = Math.floor(Math.random() * 16777216);
    var mc:MovieClip = new MovieClip();
    var t:TextField = new TextField();
    t.autoSize = "center";
    t.text = String.fromCharCode(randomBetween(97, 122));
    t.setTextFormat(f);
    mc.addChild(t);
    mc.x = (Math.random() * stage.stageWidth);
    mc.y = stage.stageHeight;
    mc.ang = 0;
    mc.range = randomBetween(4, 20);
    addChild(mc);
    mc.addEventListener(Event.ENTER_FRAME, shimmy);
}
```

The first line of code creates a timer that tells Flash to create a new `Timer` object—new `Timer(50, 0)`—which will wait 50 milliseconds before adding another letter, and that the sequence repeats forever. The next line creates the listener that will “listen” for this virtual “beep” every 50 milliseconds, and when it hears the beep, it will create a random letter. The third line—`t.start()`;—is the method of the `Timer` class that starts the clock running.

The `createLetter()` function is how the letters arrive in the movie. The first variable creates a `TextFormat` object, which is used to format text fields.

Now that you have the object, you need to give it the formatting information. The size of the text is set by using a custom function, `randomBetween()`, that determines the maximum and minimum size of the letters. The color is set by picking a random value between 0 and 1 and then multiplying that result by 16,777,216 and rounding the answer down to the nearest integer. Where did that number come from? That’s how many colors you can find in the 24-bit color space.

You then create a movieclip and a text field. The text field is centered, the letters are added to it, and formatting is applied.

The line `t.text = String.fromCharCode(randomBetween(97, 122))`, which determines which letter appears in the text field, looks rather complex, but it isn’t really. You needed random values, so you used the `fromCharCode()` method of the `String` class to return a string derived from integers that represent ASCII values for the desired letters (the lowercase alphabet, as it happens). To ensure they are all different, you use that custom `randomBetween()` function again and tell Flash, in plain English, “You go pick an ASCII value between 97 and 122. When you have it, add the letter of the alphabet that value represents to the text field.” The next two lines tell Flash the size and color of the text formatting (`f`) and add the text field to the display list of the movieclip (`mc.addChild(t)`).

The remainder of the code tells Flash where to put the movieclip, adds the movieclip to the display list of the stage, and puts the movieclip in motion using a custom `shimmy()` function. Now you’ll write that function.

5. Press Enter (PC) or Return (Mac) twice and enter the following code that will put the letters in motion:

```
function shimmy(evt:Event):void {
    var mc:MovieClip = MovieClip(evt.target);
    mc.y -= randomBetween(6, 10);
    mc.x += (mc.range * Math.cos(mc.ang += 0.4));
    mc.scaleY -= 0.02;
    if (mc.scaleY <= 0) {
        mc.removeEventListener(Event.ENTER_FRAME, shimmy);
        removeChild(mc);
    }
}
```

Before we start explaining this code, remember, all motion in Flash is either across the stage on the x axis or up and down the stage on the y axis. Objects moving from the top to the bottom of the stage have increasing y values, and all objects moving from left to right have increasing x values . . . and vice versa.

This code block determines the stage movement of the movieclip containing the letter from the previous code block. Each one is randomly placed on the y axis, and the cosine method—`Math.cos()`—of the `Math` class is used to add a left-to-right shimmy, which really is the movement along a cosine wave, as the letters move upward.

As the letters move up the stage along the y axis, they are scaled down by 2%—`mc.scaleY -= 0.02;`—every time the timer “beeps.” The last three lines essentially tell Flash, “Look, this thing is going to eventually scale down to a value of 0. When you hear that, get rid of the movieclip.”

There’s that `randomBetween()` function again. Next, you’ll write that to finish off this code.

6. Press Enter (PC) or Return (Mac) twice and enter the following:

```
function randomBetween (min:Number, max:Number):Number {
    return (Math.random() * (max - min)) + min;
}
```

This function picks a random number based on the parameters provided within the parentheses. The return statement does a bit of math and spits back a number. Let’s see how this works in regard to setting the size of the letters. The line that does that is

```
f.size = randomBetween(80, 120);
```

This means the calculation would be

```
return (Math.random() * (120 - 80)) + 80;
```

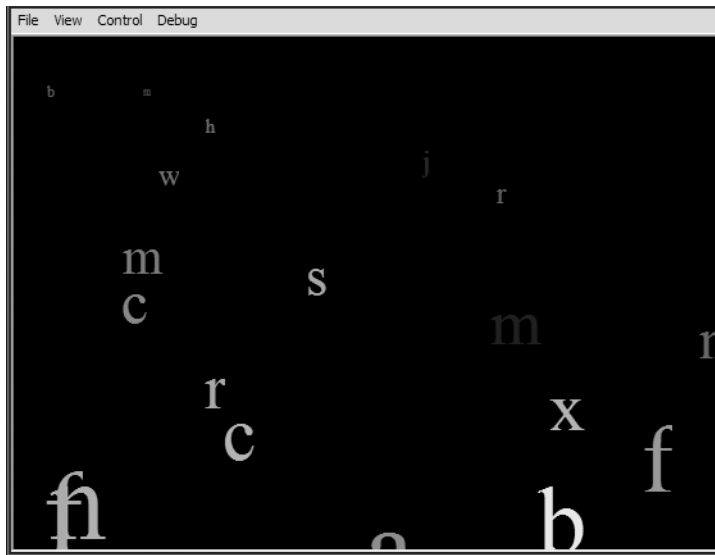
Let’s assume the `Math.random()` value is `.35`. The calculation would be

```
return (.35 * (120 - 80)) + 80;
```

Here, the result of  $(.35 * (120 - 80)) + 80$  is 93.6, so the size of this letter would be 93.6 pixels.

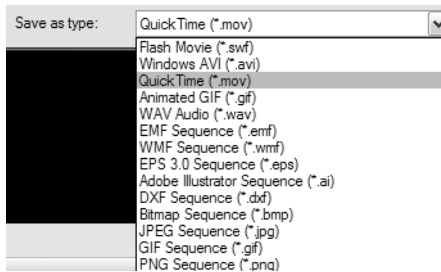
*Aren’t pixels integers? You betcha. Then how can Flash display something at a height of 93.6 pixels? Ultimately, the end result will be 94, but Flash stores an internal representation of coordinates down to the 1/20th of a pixel, a unit known as a **twip**.*

7. Save and test the movie. You should see letters, as shown in Figure 13-17, moving up the stage in a wavy motion. They get smaller as they move upward and eventually disappear.



**Figure 13-17.** Letters randomly generated, formatted, and put into motion using ActionScript

8. Select File ► Export ► Export Movie. When the Export Movie dialog box opens, navigate to the folder where you want to save the file and select QuickTime (\*.mov) from the Save as type drop-down menu shown in Figure 13-18. Click OK to open the QuickTime Export Settings dialog box.



**Figure 13-18.** Exporting a Flash file as a QuickTime movie

9. Select Ignore stage color (generate alpha channel) as shown in Figure 13-19 and change the After time elapsed value to 10. Click the Export button.





**Figure 13-19.** The QuickTime Export Settings dialog box

Let's quickly review this dialog box. The Render width and Render height values match the current stage size and will be the physical dimensions of the video. You can change these values. The Maintain aspect ratio selection, when checked, ensures distortion is not added to the video if you resize the video. The Ignore stage color (generate alpha channel) selection essentially turns the stage color invisible, which makes this ideal if you want this animation to play over content in After Effects CS3 or some other video editor.

The Stop exporting area gives Flash an idea of when to stop the process. The first choice, When last frame is reached, should be used if you have content on the timeline. The second one, After time elapsed, is ideal for situations such as this exercise where content is generated by ActionScript. In this case, you are producing a clip with a duration of 10 seconds.

The Store temp data options come into play during the render process. You can choose to either store the temp data in memory for short movies or create a temporary file on the desktop or some other location. The QuickTime Settings button opens the QuickTime Settings dialog box where you can change the codec and audio settings for the final movie.

When you click the Export button, a progress bar will appear as well as an alert telling you where the export log can be found.

- 10.** Click OK and quit Flash. Open your new QuickTime movie and check it out—pretty neat!

As we said earlier, you can export these things as animations for use in other applications. For example, one of the authors dropped the file into an After Effects CS3 project, as shown in Figure 13-20, and then created the QuickTime movie named *BubblingLettersFinal.mov* that is found in your Chapter 13 Exercise folder.

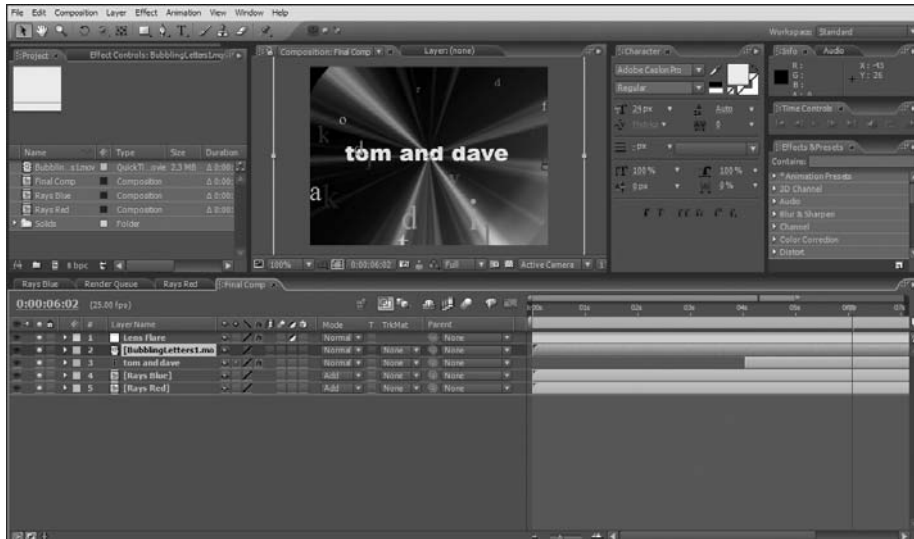


Figure 13-20. The video is used in After Effects CS3.

## What you've learned

- How Flash movies are streamed to a web page
- A couple of ways of turning the Bandwidth Profiler into your new best friend
- Tips and tricks for optimizing content for fast download
- One method of creating preloader in Flash
- How to convert a Flash movie into a QuickTime video for use in a video editing application

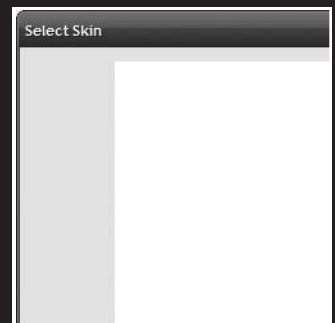
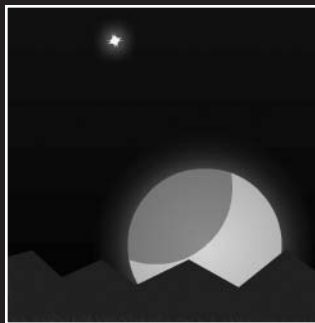
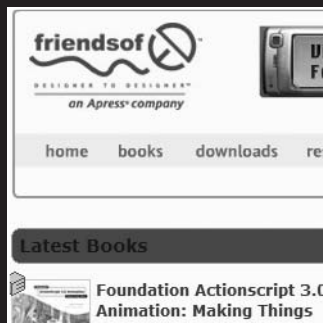
There wasn't a lot of "geeky" or cool stuff in this chapter. Instead the focus of this chapter was on how to optimize your Flash movies for web playback. We examined how the data in your Flash movie gets from "here to there" and in what order. We reviewed several ways of using the Bandwidth Profiler from identifying content bottlenecks to actually emulating the download of a bloated Flash movie into a dial-up modem. It wasn't pleasant, but we then showed you a number of ways to fine-tune your Flash movies in order to let you maximize that 15-second window of opportunity you get when a user hits your site. The chapter wrapped up by showing you a couple of ways to create preloaders for your site, how to load remote content into a movieclip, and ended by moving into the "uber-cool" zone as we showed you how to convert a Flash movie into a QuickTime video.

Now that you know how to prepare files for streaming, let's look at the end game . . . preparing the SWF file. Turn the page, and we'll see you in the next chapter.





# 14 PUBLISHING FLASH MOVIES



If there is one fundamental fact regarding publishing your Flash movie to the Web, it is this: *The SWF isn't a web document*. Nothing drives us crazier than somebody telling us, “Dudes, check out my Flash site,” only to have that individual double-click a SWF on his or her computer’s desktop. Flash SWFs should only appear on the Web if they are embedded into an HTML page. Thus a “Flash site,” to be precise, is composed of an HTML page that points to the SWF and any media—audio, video, images, text—that the SWF may need from external sources.

What we’ll cover in this chapter:

- The web formats used by Flash
- Publishing a SWF for web playback
- Dealing with remote content

Files used in this chapter:

- YawningParrot.fla (Chapter14/ExerciseFiles\_CH14/Exercise/YawningParrot.fla)
- ParrotFW.gif (Chapter14/ExerciseFiles\_CH14/Exercise/ParrotFW.gif)
- MoonOverLakeNanagook.fla (Chapter14/ExerciseFiles\_CH14/Exercise/MoonOverLakeNanagook.fla)
- MoonOverLakeNanagook.fla (Chapter14/ExerciseFiles\_CH14/Exercise/Nanagook/MoonOverLakeNanagook.fla)
- Player.fla (Chapter14/ExerciseFiles\_CH14/Exercise/Player.fla)

## Web formats

Creating the SWF is a bit more complicated than selecting File ► Publish Preview and merely clicking away in the Publish panel. As we pointed out in the previous chapter, you need a grounding in what’s under the hood before you create the car.

If there is one theme we have been stressing since page 1 of this book, it is *Keep it small!* This is the reason for Flash’s broad acceptance on the Web and where an understanding of the publishing process is invaluable. Up to this point, we have essentially created a bunch of FLA files and asked you to test them. The time has arrived to get off of the test track and put the vehicle on the street. When you publish your movie, Flash compresses the file, removes the redundant information in the FLA, and what you are left with—especially if you took the last chapter to heart—is one sleek, mean web presentation. The default output file format—yes, there is more than one—is the SWF (pronounced *swiff*). The SWF is wrapped in HTML through the use of <object> and/or <embed> tags, plus extra information on how the browser should play the SWF.

*Yes, you can link directly to a SWF without that bothersome HTML. Just be aware that the SWF will expand to the full size of the browser window, meaning all of the content on the stage will also enlarge. In many respects, linking directly to the SWF is “Rookie Error #1.”*

Before we move into actually publishing a movie, let's look at some of the more common file types used on the Web, listed here:

- Flash (.swf)
- HTML (.htm or .html)
- GIFs (.gif)
- QuickTime (.mov)

## Flash

Before there was Flash, there was Director. Though used primarily for interactive CDs, DVDs, and kiosks, it was at one time the main method employed to get animations to play on the Web. The technology developed by Macromedia to accomplish this was named **Shockwave**, and the file extension used was `.dcr`. Flash also made use of this technology, and in order to differentiate between them, it became known as **Shockwave for Flash**. Flash Player is the technology that allows the SWF to play through a user's browser. Through a series of clever moves, Flash Player has become ubiquitous on the Web. In fact, Adobe can rightfully claim that Flash Player, regardless of version, can be found on 98% of all Internet-enabled computers on the planet. This means, in theory, you can assume your movies are readily available to anyone who wishes to watch them. But the reality gets a bit more complicated.

*For you trivia buffs, the first couple of iterations of Shockwave for Director used a small application named Afterburner to create the .dcr files. When a Director developer prepared a presentation for the Web, he or she didn't create the .dcr. The movie was "shocked." One of the authors happened to be around the night Macromedia quietly released Shockwave and Afterburner to the Director community, and he still remembers the excitement generated by members of the group as they posted circles that moved across the page, and the "oohs" and "aahs" that followed as the circles moved up and down.*

Each new Flash Player brings with it new functionality. Flash Player 8 introduced filter effects and blend modes, which can't be played in Flash Player 7. FLV video can't be played in Flash Player 5. Any movie you prepare using ActionScript 3.0 can only be played in Flash Player 9 or higher. Though you may initially regard this as a nonissue, you would be making a gross miscalculation. Corporations, through their IT departments, have strict policies regarding the addition or installation of software to corporate-owned computers. We personally know of one organization that isn't budging, and its Flash Player policy is Flash Player 6 or lower. The upshot is, it is the shrewd Flash designer who actually asks a potential client to let him or her know what versions of Flash Player are to be targeted for the project. The last thing you need is to find yourself rewriting every line of code and reworking the project when you assumed the target was Flash Player 9 but corporate policy dictates Flash Player 7 or lower.

*Flash Player 9 follows a tradition that each successive version of Flash Player will play content faster than its predecessors. Adobe is claiming that there is a 75% speed increase of Flash Player 9 over Flash Player 8. This sort of increase is usually enough for most users to install the new version. Even so, in many instances, actually downloading and installing the plug-in is becoming a thing of the past. Flash Player has the ability to download and install in the background, but, as one of the authors is quick to point out: “It takes a programmer to make it work.”*

## HTML

HTML is short for **HyperText Markup Language**. Where HTML and ActionScript part company is that HTML is a formatting language, whereas ActionScript is a scripting language. This means HTML is composed of a set of specific instructions that tell the browser where content is placed on a web page and what it looks like. ActionScript has nothing to do with the browser. It tells Flash how the movie is to work.

The HTML instructions, or tags, are both its strength and its weakness. HTML was originally developed to allow the presentation of text and simple graphics. As the Web matured, HTML found itself hard pressed to stay current with a community that was becoming bored with static content on pages.

The real problems with HTML start when you try to drop multimedia or interactive media into a web page. HTML simply wasn't designed for this sort of “heavy lifting,” which explains why JavaScript (a language that shares roots with ActionScript) is now so widely used.

For a Flash designer, knowledge of how HTML works is critical, because it is the technology that enables your movies to be played on the Web. Of course, this isn't as difficult as it once was. Today, through the use of Dreamweaver CS3 and even Flash, it involves nothing more than a couple of mouse clicks to create the HTML that makes this possible. You will still need to play with the HTML—you saw this in Chapter 7 when you had to dig into the JavaScript code to enable full-screen playback of a Flash video—because your HTML document can do things that Flash can't. This would include such features as ALT attributes for screen readers and key words used to attract search engines.

*Dreamweaver CS3 is what is called a **WYSIWYG editor**. The acronym is short for “what you see is what you get.” In many cases this is true, but more often than not, these pages, when viewed in a browser, don't look the same way they did in Dreamweaver CS3. Maybe it should be called a **WYSINNWYG editor**—what you see is not necessarily what you get.*

The other thing to stick in the back of your mind is that Flash web pages aren't as common as they once were. Web pages consisting solely of one SWF are still around, but Flash is also becoming a medium of choice for the delivery of banner ads, videos, and other interactive content that are elements of an HTML web page. To see an example of this, you



need look no further than our beloved publisher. If you hit the friends of ED home page at [www.friendsofEd.com](http://www.friendsofEd.com) (see Figure 14-1), you will see a Flash banner at the top of the home page, while the rest of the page is composed of HTML.



Figure 14-1. A typical Flash/HTML hybrid page

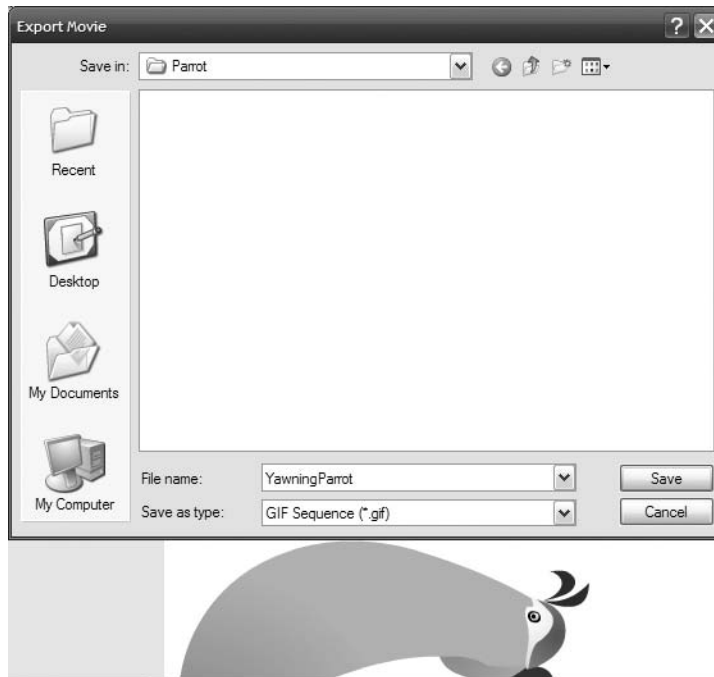
## Animated GIFs

Before there was Shockwave, there was the infamous animated GIF file. These files were the original web animations, and you can export your Flash movie as an animated GIF. Why would you want to do this if Flash Player is so ubiquitous? The reason is users don't need to install the Flash plug-in to view them. In fact, it is a two-way street: you can import a GIF animation into a Flash movie, and you can export a Flash movie as an animated GIF. Here's how:

1. Open the `YawningParrot.fla` file. This is the file to be exported out as an animated GIF. How this works is Flash will convert each frame of the movie to a GIF image. There are 355 frames in this animation, meaning you really should prepare yourself to create 355 separate GIF images.

*OK web heads, settle down. Creating an animated GIF consisting of 355 frames is, as our editor, Chris Mills, would say, "Simply not done, old chap." We know that, but if you understand what happens . . . in a big way . . . you'll be more cautious in your efforts. Anyway, the parrot is pretty cool and makes for a rather interesting workout for Fireworks CS3.*

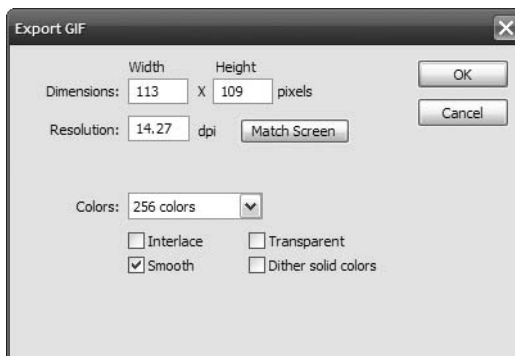
2. Select **File** ► **Export** ► **Export Movie** (press `Ctrl+Alt+Shift+S` on the PC or `Cmd+Option+Shift+S` on the Mac) to open the Export dialog box. Navigate to the Parrot folder in the Chapter 14 Exercise folder and select **GIF Sequence (\*.gif)** in the Save as type drop-down menu (see Figure 14-2).



**Figure 14-2.** Select GIF Sequence (\*.gif) as the image type.

3. Click Save to open the Export GIF dialog box shown in Figure 14-3. Specify these settings:

- Dimensions: 113 X 109
- Colors: 256
- Smooth: Selected

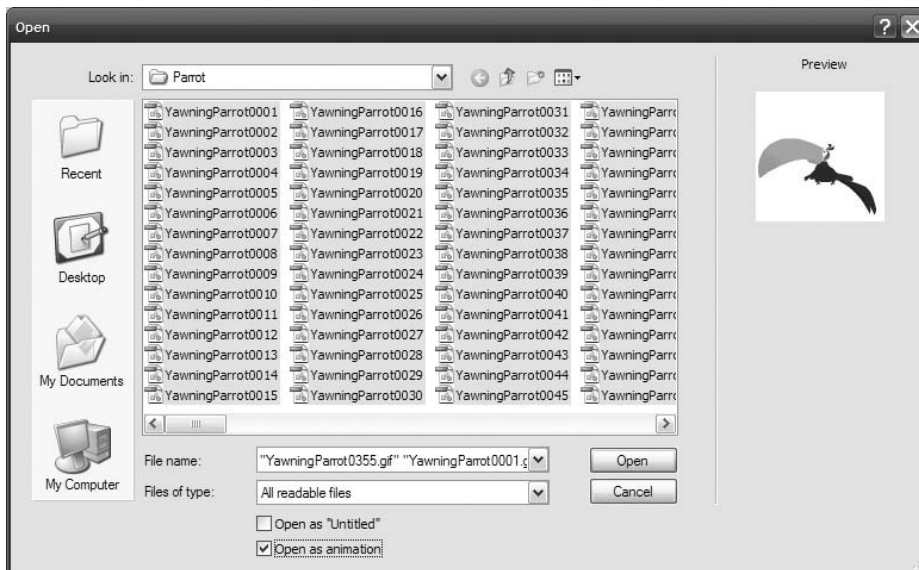


**Figure 14-3.** Preparing to export the Flash timeline as a GIF animation

You should have noticed that when you changed the Dimensions settings, there was a corresponding reduction in the Resolution value. If you click the Match Screen button, you will be returned to the original settings for this image. The physical reduction of each frame and its corresponding reduction in resolution have the net effect of creating a rather small GIF image.

4. Click the OK button, and a progress bar will appear showing you the progress of the export. This is a fairly quick process and should take less than 10 or 15 seconds. When it finishes, the progress bar will disappear, and you will be returned to the Flash stage. At this point, you are now the proud owner of the 355 GIF images that will be used to create the animation.

We aren't going to get into the nitty-gritty of creating the GIF animation in Fireworks CS3. The process is fairly simple. Launch Fireworks CS3, click the Open button on the Start screen, and navigate to the folder containing your GIF images. Select all of them in the Open dialog box, as shown in Figure 14-4, and select Open as animation. When you click the Open button, Fireworks will create the animated GIF by putting each image in a frame. You can then do what you need to do and export the file out of Fireworks CS3 as an animated GIF.

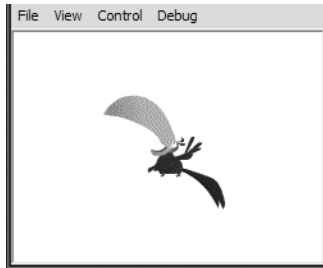


**Figure 14-4.** Importing the GIF files into Fireworks. The key is to select Open as animation.

*Only the main timeline is considered when Flash content is converted to an animated GIF. Nested movieclip timelines do not make it through the translation process. The simple rule of thumb is, if you can see it move while you manually scrub the timeline, the GIF can too. If you can't, it won't show.*

Now that you know how to create a GIF animation in Flash, let's look at the reverse process: importing a GIF animation into Flash.

1. Open a new Flash CS3 document and select File ► Import ► Import to Library.
2. Navigate to the ParrotFW.gif file in your Chapter 14 Exercise folder and click Open.
3. When the process finishes, you will see that each image in the animation, along with a movieclip, has been added to the library.
4. Drag the movieclip to the stage and test the movie. You have a low-res version of the yawning parrot, as shown in Figure 14-5.



**Figure 14-5.** A yawning parrot in the GIF format

*Yes, the earlier export exercise was partly mischievous. If you select File ► Export ► Export Movie, you can bypass the need to restitch the GIF sequence in Fireworks by choosing Animated GIF from the Export Movie dialog box. Still, it's good to know your options!*

## QuickTime

QuickTime is Apple's Internet steaming video technology. As we have pointed out throughout this book, QuickTime is losing its grip as the premiere web video technology. What you should have learned from the BubblingLetters.fla exercise in the "Optimizing Flash content for use in video" section of the previous chapter is this: *The reports of its death are premature.*

Flash is gaining ground as a broadcast animation technology, and no matter how you slice it, QuickTime is the way to go with digital video. Up until this release of Flash, QuickTime and Flash have had a rather uneasy relationship. It was extremely difficult to get Flash animations into QuickTime for editing in a video editing application. This impediment has been removed, and publishing a Flash document as a QuickTime movie is easier than it ever has been.

Which begs the question: How do you publish a Flash movie?

## It's showtime!

Everything works as it should. You have sweated buckets to optimize the movie, and the client has finally signed off on the project. It is showtime. The Flash movie is ready to hit the Web and dazzle the audience. Though you may think publishing a Flash movie involves nothing more than selecting Publish in the File menu, you would be seriously mistaken. The process is as follows:

- Open the Publish Settings window to determine how the movie will be published.
- Publish the movie and preview the SWF.
- Upload the SWF and any support files to your web server.

Let's publish a movie. Here's how:

1. Open MoonOverLakeNanagook.fla. Seeing as how this is the last chapter, let's finish the book by working with the file you created when you started the book.
2. Select File ► Publish Settings (Ctrl+Shift+F12 on a PC or Option+Shift+F12 on a Mac) to open the Publish Settings dialog box shown in Figure 14-6.

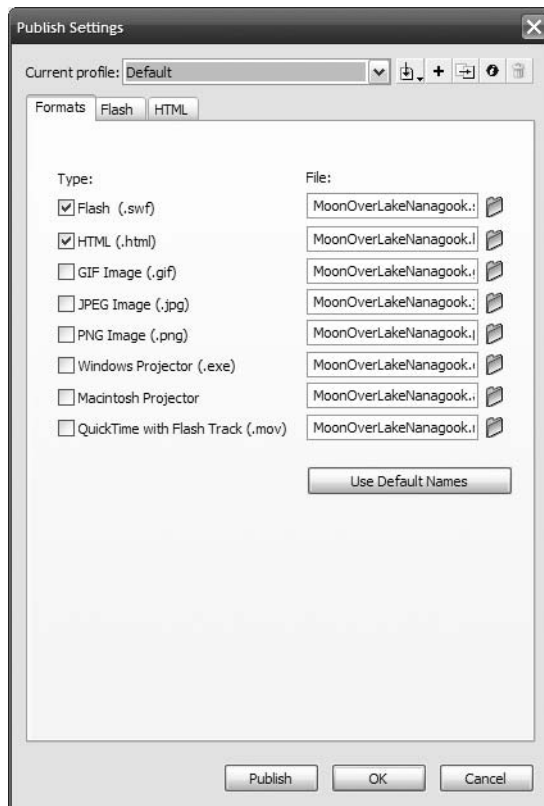


Figure 14-6. The Publish Settings dialog box

*You can also click the Settings button on the Property inspector to launch the Publish Settings dialog box. The one thing you don't want to do, unless you have a lot of Flash experience under your belt, is to select File ► Publish. Selecting this will publish the movie using whatever default settings are in place.*

As you can see, this dialog box is divided into three distinct sections: Format, Flash, and HTML. In fact, that last tab (or tabs) will change depending on the format chosen. We'll get to that in a minute. The five buttons along the top are the Profile buttons. These allow you to “tweak” your settings and then save them for future use.

The file types are as follows:

- Flash (.swf): Select this, and you will create a SWF that uses the name in the File area unless you specify otherwise.
- HTML (.html): The default publishing setting is that the Flash and HTML settings are both selected. This does not mean your SWF will be converted to an HTML document. What it means is Flash will generate the HTML wrapper for the SWF.

*If you are a Dreamweaver CS3 user, you don't need to select the HTML (.html) option. Dreamweaver will write the necessary code for the SWF when it is imported into the Dreamweaver CS3 page.*

- GIF Image (.gif): Select this, and the Flash animation will be output as an animated GIF, or the first frame of the movie will be output as a static GIF image.
- JPEG Image (.jpg): The first frame of the Flash movie will be output as a JPEG image.
- PNG Image (.png): The first frame of the movie will be output as a PNG image. Be careful with this one because not all browsers can handle a PNG image.
- Windows Projector (.exe): Think of this as being a desktop SWF that is best suited to playback from a Windows desktop or CD, not from the browser.
- Macintosh Projector: The same thing as the Windows projector. Just be aware that a Mac projector won't play on a PC, and vice versa.
- QuickTime with Flash Track (.mov): Select this, and the Flash movie is output as a QuickTime movie, providing you have chosen Flash Player 5 as your target player. Why Flash Player 5? That is the only version of Flash Player the current version of QuickTime supports.

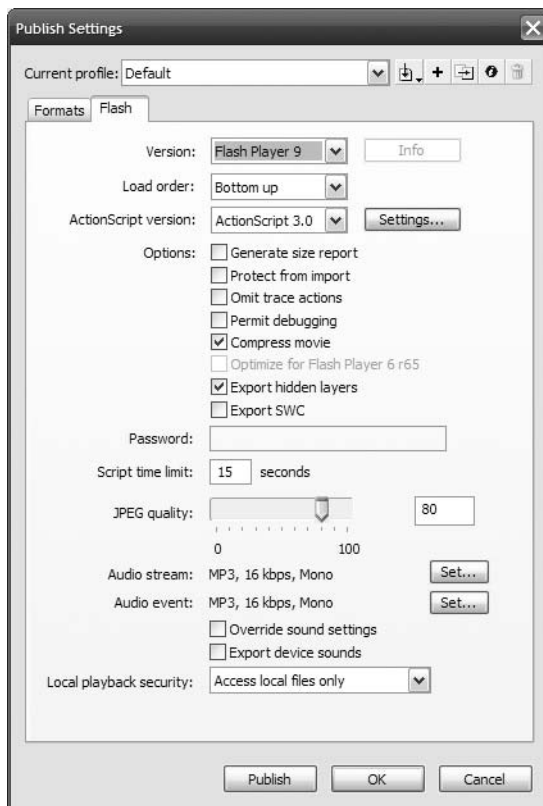
*We suspect the addition of the ability to publish to QuickTime from the File menu will make this choice even more infrequently used than it currently is.*

The Navigate buttons (they look like folders and are located beside each file type) allow you to navigate to the folder where the SWF will be saved (see Figure 14-7). If you see a path, click the Use Default Names button to strip out the path from the file name.



**Figure 14-7.** Strip out any paths in the file name to avoid problems.

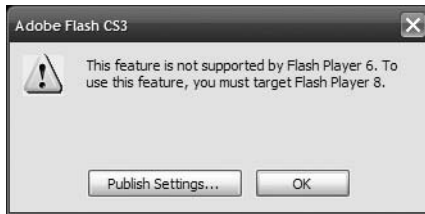
3. Select all of the types except for QuickTime. Notice how each file type kicks out its own tab. Deselect everything but the Flash (.swf) option, and click the Flash tab to open the Flash settings shown in Figure 14-8.



**Figure 14-8.** The Flash settings in the Publish Settings dialog box

There is a lot here, so let's review each of the areas in this panel:

- **Version:** This drop-down menu allows you to choose any version of Flash Player from Versions 1 to 9 (the current version) and any version of Flash Lite Player from versions 1 to 2.1. If you have the Property inspector open, you will see the version chosen also appears in the Property inspector. It is extremely important you understand that if you change your Flash Player version and are using features in the Flash movie that aren't supported by the Flash Player version you have chosen, you will be greeted by the Alert dialog box shown in Figure 14-9—but this only happens when you return to the Flash stage and try to add or manipulate something that isn't supported. In the case of Figure 14-9, we tried to add a drop shadow to a movieclip, and that feature is not supported in our target player.



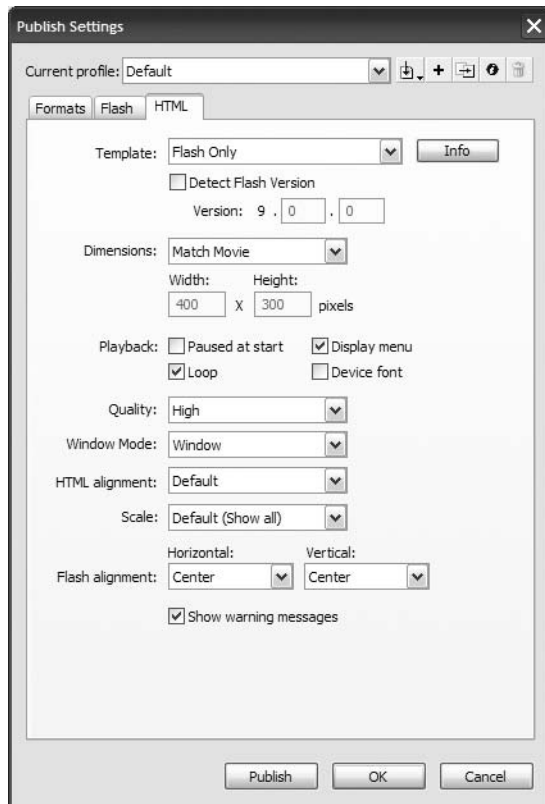
**Figure 14-9.** Flash will let you know you can't do that when you try to do something that isn't supported by the version of Flash Player you have targeted.

- **Load order:** Your choices are Bottom up or Top down. This is the order in which Flash will load timeline layers into Flash Player. ActionScript is not affected by this setting: no matter what, ActionScript is performed in order of the highest layer to the lowest.
- **ActionScript version:** There are three versions of the language. If you are publishing to Flash Player 9, you are safe with ActionScript 3.0, ActionScript 2.0, or ActionScript 1.0 (we recommend ActionScript 3.0). If you are publishing to Flash Player 8 to 6 or Flash Lite 2 or 2.1, ActionScript 2.0 is your choice, though ActionScript 1.0 will work. Everything else uses ActionScript 1.0.
- **Options:** You have a number of options regarding the treatment of the SWF available to you. They are as follows:
  - **Generate size report:** Select this, and Flash will generate a .txt document that shows you where potential bandwidth issues may be located. When this option is selected, the .txt file is generated when you publish the SWF.
  - **Protect from import:** When this option is selected, the user will be prevented from opening your SWF in Flash.
  - **Omit trace actions:** Flash will ignore any trace() actions you may have added to your ActionScript (they will actually be removed from the SWF). You've used these to track the value of a variable and display that value in the Output window. Tracing is great for debugging, but enough such statements can affect performance.
  - **Permit debugging:** Select this, and you have access to the Debugger panel in Flash even if the file is being viewed in a web browser. You really should turn this off before you post the movie to the Web.



- **Compress movie:** Even though Flash compresses the FLA when it creates the SWF, selecting this allows Flash to compress the SWF itself—usually text-heavy or ActionScript-heavy—to an even greater extent during the publish process. If you are publishing to Flash Player 5 or lower, you can't use this option.
- **Optimize for Flash Player 6 r65:** Though this option is usually grayed out, you may select it when targeting Flash Player 6 to even further optimize the SWF. What's the r65? That particular release of Flash Player 6 introduced Flash Player enhancements that made it a “sneak peek” at Flash Player 7.
- **Export hidden layers:** All this means is that any layer with the visibility icon turned off will be compiled into the SWF. Developers often like to keep reference layers handy during authoring, but in previous versions of Flash, such layers would show in the SWF, even if they were hidden in the FLA. An old trick to “really” hide them was to convert such layers to guide layers—but that can get tedious. If you really want those layers gone, just delete them. If you're a little lazy, use this feature instead.
- **Export SWC:** Select this if you are an absolute code jockey and create your own custom components for Flash. That sort of thing is way out of the scope of this book.
- **Password:** This option works in conjunction with the Debugger panel. If you add a password to this text entry box, whoever opens the Debugger panel will be prompted to enter the password if debugging the SWF in a browser. If the plan is to test and debug your Flash app remotely, this is a “must do” option.
- **Script time limit:** Sometimes your scripts will get into a loop, sort of like a dog chasing its tail, and these things can go on for quite a long time before Flash sighs and gives up. Enter a value here, and you are telling Flash exactly when to give up.
- **JPEG quality:** This slider and text field combo specifies the amount of JPEG compression applied to bitmapped artwork in your movie. The value you set here will be applied to all settings in the Bitmap properties area of the library unless you override it for individual bitmaps on a per-image basis.
- **Audio stream:** Unless there is a compelling reason to do otherwise, leave this one alone. The value shown is the one applied to the Stream option for audio in the Property Inspector.
- **Audio event:** Same warning as the previous choice but for event sounds.
- **Override sound settings:** Click this and any settings—Stream and Event—you set in the Sound properties area of the library are, for all intents and purposes, gone.
- **Export device sounds:** Use this only if you are using Flash Lite and publishing to a mobile device.
- **Local playback security:** The two options in this drop-down menu—Access local files only and Access network only—permit you to control the SWF's network access. The important one is the network choice. Access networks only protects information on the user's computer from being accidentally uploaded to the network.

- Click the Formats tab and select the HTML (.html) file type. When you do that, the Publish Settings dialog box sprouts an HTML tab. Click the HTML tab to open the HTML settings shown in Figure 14-10.



**Figure 14-10.** The HTML tab in the Publish Settings dialog box in Flash CS3

The important thing about this dialog box is to be aware that it does not convert your SWF to HTML. The best way to consider this option is like buying a hamburger at a large international chain. When the hamburger is finally ready, it will be wrapped in paper or placed in a colored box that identifies the contents. For example, you have ordered the MegaBurger, and the burger is wrapped in blue paper that has the words “MegaBurger” printed on it. The HTML option performs the same job: it provides the wrapper that tells the browser what’s inside.

*If you are a Dreamweaver CS3 user or prefer to “roll your own” HTML code, it still won’t hurt to review this section, but Dreamweaver CS3 does this job for you.*

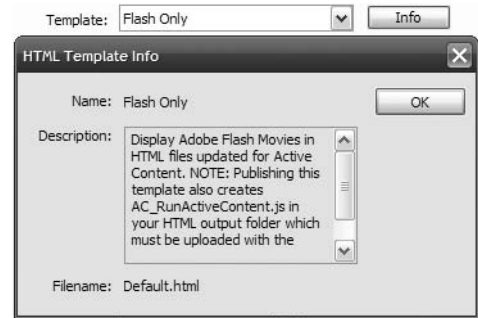
*If the Flash movie is to appear in a CSS-based layout, a lot of the options in this dialog box will not be used by the coder. Still, the HTML page to be created is a good starting point for a code jockey.*

Let's review the main features of this panel:

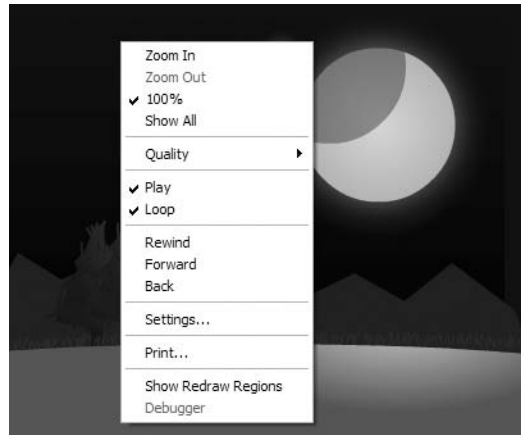
- **Template:** This drop-down menu contains 11 options, but they all specify the type of HTML file you want the SWF to be embedded into. The Info button will give you a brief description of the selected template (see Figure 14-11). These templates can be found in C:\Program Files\Adobe\Adobe Flash CS3\en\First Run\HTML on your PC or HD:/Applications/Adobe Flash CS3/First Run/HTML on your Mac. If you are a hardcore coder and know exactly what you are doing, feel free to change them only after you have made a backup of the files. Though there are a number of templates, the Flash Only template will most likely become the one you use most often.
- **Detect Flash Version:** This option determines whether the JavaScript code for this purpose is added to the HTML. What this does is to check to see whether the user's Flash plug-in will work with the version of Flash Player you have targeted. If the user has the version, life is a wonderful thing and the movie will play. If not, the user will see an error message along with a link to the location where the latest plug-in can be found.

*If you are a JavaScript wizard, feel free to customize the detection JavaScript to react differently if the wrong plug-in version is detected. For instance, if the IT boys have decreed "Thou shalt not add software to our machines," you could rewrite the code to load and play an alternate version of the SWF instead of suggesting the user do something that is forbidden.*

- **Dimensions:** You get three choices—Match Movie, Pixels, or Percent—in this drop-down menu. Select one of the last two options, and you can change the physical size of your movie. If you choose Percent, you will discover the one circumstance that allows content positioned outside the stage to possibly show.
- **Playback:** These four choices determine what happens when the movie starts playing:
  - The first option, Paused at start, means the user gets things going. This is very common with banner ads, and you'd have to provide a button to tell the play-head to start moving, or the user would have to be smart enough to right-click and use the plug-in's context menu to select Play.
  - The Display menu option is actually quite important. It has nothing to do with menus in the movie and everything to do with Flash Player. If you test MoonOverLakeNanagook.fla and right-click (PC) or Ctrl-click (Mac) the SWF, the menu shown in Figure 14-12 appears. This menu allows the user to modify how Flash Player displays the movie. Many Flash designers and developers turn this off because they don't want people switching to low-quality graphics or zooming in on the stage. Still, there is a very important use for this menu. If your site requires visitors to use a web camera or a microphone, clicking the Settings option will allow them to choose the devices to be used. (The Settings option is always available, even if you hide the rest of the context menu.)

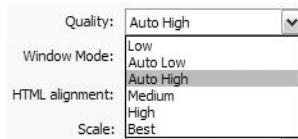


**Figure 14-11.** The Flash Only template description



**Figure 14-12.** The Flash menu that is displayed at runtime

- The Loop option plays the movie continuously if it is selected or only once if deselected. The key point here is any stop() actions you may have in your ActionScript will override this selection.
- The Device font selection replaces any static text in your movie with a system font—\_sans, \_serif, and \_typewriter—which can result in a significant file-size reduction. The downside to this choice is you have absolutely no control over which font is used because if the user doesn't have the three fonts installed, the machine will use one that is closest to the font, meaning the text may wrap or even change the “look” of your movie. Is this one of those things that falls into the category of things you should never do? Not really. It is your movie, and if you decide this is the way to go, you at least are aware of the potential hazards in the choice.
- Quality: This drop-down menu contains the six choices shown in Figure 14-13. These specify the render quality that your movie will play at, and the choice you make determines the speed at which your movie runs on the user's machine or device. We suggest you start with Auto High, which permits Flash to automatically drop the quality to maintain the frame rate and synchronization if needed. In many respects, this area is not one that should concern you because if Display menu is selected, the user can change this setting at runtime.



**Figure 14-13.** Try starting with the Auto High quality setting.

- **Window Mode:** The selection you make here will appear in the `wmode` settings in the `<object>` and `<embed>` tags used in the HTML. If you are unsure as to what the choices do, just leave the choice at the default: `Window`.
- **HTML alignment:** This selection allows you to specify the position of your movie window inside the browser window. The default will place the SWF in the center of the browser window.
- **Scale:** If you have changed the dimensions of the movie using the `Dimensions` option, the choices in the drop-down menu determine how the movie is scaled to fit into the browser window.
- **Flash alignment:** These two options permit you to set the `Vertical` and `Horizontal` alignment of your movie in its window and how it will be cropped, if necessary.
- **Show warning messages:** If this box is checked, any errors discovered when the HTML file is loaded—missing images is a common one—are displayed as browser warnings when the user arrives on the page.

Now that we have reviewed the major points, let's publish `Lake Nanagook` and look at it in a browser. Before you start, click the `OK` or `Cancel` button to close the `Publish Settings` dialog box and return to the `Flash` stage. Save the `MoonOverLakeNanagook.fla` to the `Nanagook` folder in your `Chapter 14 Exercise` folder. We'll explain why in a moment. Now open the `Publish Settings` dialog box and let's get busy:

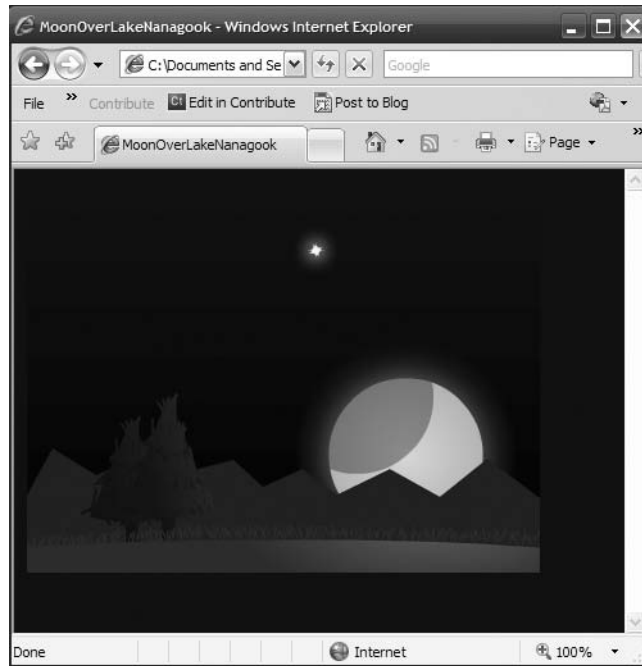
1. Click the `Formats` tab and select the `Flash` and `HTML` formats.
2. Click the `Flash` tab and specify these settings:
  - `Version:` `Flash Player 9`
  - `Load order:` `Bottom up`
  - `ActionScript version:` `ActionScript 3.0`
  - `Compress movie:` `Selected`
  - `Export hidden layers:` `Deselected`
3. Click the `HTML` tab and specify these settings:
  - `Template:` `Flash Only`
  - `Dimension:` `Match Movie`
  - `Quality:` `Auto High`
  - `Flash alignment:` `Center for both Horizontal and Vertical`
4. Click the `Formats` tab and, when the panel opens, click the `Use Default Names` button to strip off any paths that might be associated with this movie.
5. Click the `Publish Button`. You will see a progress bar that follows the publishing process. Click `OK` to close the `Publish Settings` dialog box and return to your movie.

- Minimize the Flash stage and open the Nanagook folder in the Chapter 14 Exercise folder. You will see that Flash has created four files, as shown in Figure 14-14. There are the FLA, SWF, an HTML file, and a JavaScript file named `AC_RunActiveContent.js`. This last file is used to prevent Internet Explorer from kicking out the “Click to activate and use this control” warning that has plagued developers since Microsoft changed how its browser displays active content such as Flash and even its own Windows Media Player. The only file that doesn’t need to get uploaded to the server is the FLA.



**Figure 14-14.** The results of publishing the Flash movie

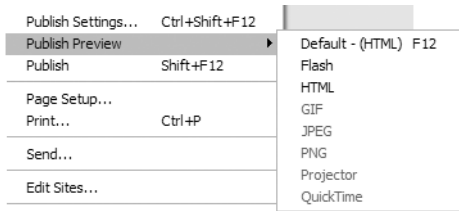
- Open the `MoonOverLakeNanagook.html` file in a browser. The movie, as shown in Figure 14-15, starts playing. Congratulations!



**Figure 14-15.** Playing the movie in a browser

*Hang on. How did the background color of the browser page turn blue? There was nothing in the HTML settings for that one. If you publish a Flash movie and use the HTML option, the background color of the HTML document will change to the stage color of the Flash movie.*

Before we move on, there is one last option you may have noticed in the File menu that we'd like to talk about. The Publish Preview menu—File ► Publish Preview—contains the formats from the Publish Settings dialog box (see Figure 14-16). Selecting this will publish the movie and then launch the results in a browser if you select Default - (HTML). This menu reflects the choices made in the Publishing Settings window, which explains why a lot of the options are grayed out. If you are a Dreamweaver CS3 or Fireworks CS3 user, this menu item is the same as being able to do a browser preview in both of those apps. In fact, they all use the same key, F12, to launch the preview. The browser that opens will be the default browser used by your computer's operating system.



**Figure 14-16.** You can preview the movie in a browser without leaving the Flash interface.

## Publishing Flash movies containing linked files

In Chapter 5, you created an MP3 player. Though you tested it locally, nothing beats testing on a remote server. The other aspect of that exercise, which we didn't review until now, is that of playing content located in another folder on the server. In the case of the MP3 files, this actually makes sense. Let's assume you are going to use the same MP3 soundtrack in five Flash movies over the coming year. If that MP3 is 5 MB in size, you will have used up 25 MB of server space if the file is slipped into the folder for each project that uses it. Doesn't it make more sense to upload it once and have the movies call it into the SWF from a single location?

In this example, we are going to assume the three audio files are located in a folder named Tunes in the mythical domain of mySite.com.

1. Open the Player.fla file located in your Exercise folder. When the file opens, open the Actions Panel and scroll down to the loadSong function in line 41 of the Script pane.
2. The critical line in this function is line 43, which uses the load() method to get the song. Change this line to the following:

```
song.load(new URLRequest("http://www.mySite.com/Tunes/" + thisSong));
```

Everything is straightforward if you use absolute paths, like you've seen so far. Absolute paths contain the full domain name, which means they're accessible from anywhere on the Internet. That's both a plus and a minus. If you hardcode all your file references as absolute paths, you know they'll work—until you decide to change your domain name, or until you repurpose your content for another project in another folder structure somewhere else. In cases like that, a relative path may suit your needs. Relative paths do not reference a domain

name, and because of that, they depend entirely on a very particular “point of view,” namely, the physical location of the file making the reference.

You would think that a SWF looking for MP3s (or any external file) would consider itself as the beginning of the path—“Where is that file in relation to *me*?”—but that’s not how it works. When a SWF references external files with relative paths, its point of view is actually that of the HTML document that contains it. If the SWF and the HTML file are in the same folder, this is a moot point, but keep it in mind if you decide to put all your SWFs in one folder and your HTML files in another.

To make matters even more interesting, there’s an exception, and FLV video files are it. If you are using the `FLVPlayback` component, the path to the video, if it is a relative path, takes its cue from the location of the SWF itself. Same goes for a video object using the `NetStream` class. That said, the `FLVPlayback` component optionally uses skins, and skins are SWF files. If your movie uses relative paths to reference an `FLVPlayback` skin, set your point of view to the HTML document that contains this movie, but when referencing the FLV, set your point of view to the movie itself.

This “gotcha” often raises its ugly head if you have a custom controller or video skin or are using a server that dynamically loads the content. Either understand the “gotcha” fully, or enter the paths, as shown in Figure 14-17, as absolute paths.



**Figure 14-17.** You can save FLV skins to remote sites as well.



## What you've learned

- How to prepare a SWF for web playback
- How to export a Flash movie as a GIF animation and how to import a GIF animation into Flash
- How to deal with remote content needed by the SWF

This chapter dealt with the “end game” in Flash. We think you are now aware that preparing your Flash files for web output involves a lot more than simply selecting **Publish** in the **File** menu. There is a lot to consider, and those considerations range from what format will be used to output the file to a number of very important options that need to be addressed. We also dealt with remote content and how the SWF can grab it from elsewhere on your site and on the Web.

Speaking of the “end game,” we are at the end of this journey that started and ended at Lake Nanagook. We hope you had fun and that you are inspired to explore Flash CS3 even further. As you do, you will discover a fundamental truth about this application: the amount of fun you can have with it should be illegal. We'll see you in jail.



**INDEX**

## Numbers and Symbols

9-slice scaling, 118–124  
 + (addition) operator, using with text, 184  
 @ (at symbol), in E4X, 457  
 [] (array access operator), 458  
 = (assignment operator), 179, 184–185  
 \ (backslash) character, 258–260  
 {} (bracket characters), 432  
 <br/> (break) tag, 434  
 . (dot) notation syntax  
   in E4X, 457  
   in `setStyle()` method, 436–437  
 && (double ampersand), 186  
 || (double bar), 186  
 // (double forward slash), 175–176, 435  
 == (equality operator), 185  
 > (greater than) operator, 185  
 >= (greater than or equal to) operator, 185  
 ! (logical NOT) operator, 232  
 < (less than) operator, 185  
 <= (less than or equal to) operator, 185  
 () parentheses, grouping expressions with, 184  
 ; (semicolons), in ActionScript, 175  
 /\* and \*/ characters, commenting code blocks with, 176

## A

<a> (anchor) tag, 264, 434  
 actions, code blocks as, 17  
 Actions panel, ActionScript, 160–163  
   Apply line comment button, 170  
   context menu, 163  
   new features since Flash 8, 161  
   zones in, 161–162  
 Actions toolbox, Actions panel, 162  
 ActionScript. *See also* ActionScript 3.0  
   accessing properties with, 166–169  
   adding items to `TileList` instance with, 420  
   basics of, 158  
   vs. behaviors, 164  
   class files, 190  
   code coloring in, 170  
   creating scroll buttons with, 279  
   creating text fields with, 261–263  
   displayed in color, 175  
   dot (.) notation in, 177–178  
   escape sequences supported by, 258–260  
   grammar rules, 174–175  
   looping the timeline, 202–203  
   objects in, 164–173  
   pausing main timeline, 201–202  
   playing video with, 361, 364  
   power of, 159–160  
   putting letters in motion with, 529–530  
   setting properties via, 168  
   syntax for, 174–175  
   triggering to make frog disappear, 275–276  
   using, 200–203  
   writing and testing code, 168  
   writing for mobile devices, 495–502  
 ActionScript 2.0  
   syntax errors reported in, 196–200  
 ActionScript 3.0  
   case sensitivity of, 174  
   controlling audio with, 219–224  
   creating Object instance in, 432  
   event handling in, 171–173  
   migrating to, 191  
   reading language and components reference, 192–195, 260  
   setting version, 546  
   syntax errors reported in, 196–200  
   tutorials and articles, 160  
   using tooltips, 222  
 ActionScript version, 546  
 :active pseudo-class, 441  
 addChild(loader), 521  
 addition operator (+), using with text, 184  
 Adobe, web site, 322  
 Adobe Developer Center, web site, 160  
 Adobe Flash Video Encoder, opening, 344  
 Adobe Mobile & Devices Developer Center, web site, 474  
 Advanced Effects dialog box, 254  
 Afterburner, creating .dcr files with, 537  
 AI File Importer preferences, 100  
 AIFF (Audio Interchange File Format), 207, 237  
 Alert dialog box, 518–519  
 Align panel  
   aligning buttons in, 226  
   and stacking order, 140–143  
 alignment of text. *See* text  
 alpha channel video  
   encoding, 346  
   playing with, 383–384  
   preparing and using, 369–371  
 alpha transition, tweening, 323–325  
 Alpha value, 32  
 anchor points, 68–69  
   Bezier curves, 67  
   shape tweens use of, 291–292  
   using sparingly, 296  
 anchor tag (<a>), 264, 434  
 Angular blend, shape tween settings, 290  
 animated buttons, 328–333  
 animation  
   creating (exercise), 148  
   in Flash CS3, 284–336  
   shared symbols used by, 124  
   start and end points, 15–16

“As We May Think” article, 508  
 aspect ratio, 348  
 assignment operator (=), 179, 184–185  
 audio
 

- adding to Flash, 210–213
- in Flash CS3, 206–237
- pausing a track, 233
- previewing files, 211
- working with in Flash, 214–216

 Audio event option, 547  
 audio formats, Flash and, 207  
 Audio Interchange File Format (AIFF). *See* AIFF (Audio Interchange File Format)  
 Audio stream option, 547  
 authoring environment interface. *See* Flash authoring environment interface  
 authors under glass (exercise), 147–148  
 auto kern option, 248  
 auto-widen mode, text fields, 246  
 Available Devices bar, Search Devices icon on, 478  
 AVI (Audio Video Interleave) format, 342

## B

backslash (\) character, 258–260  
 Balderson, Joseph, 124  
 bandwidth, 532
 

- effect on download speed, 509–510
- limit, 513

 Bandwidth Profiler
 

- dealing with image distortions, 519–520
- for Flash movies, 512–515
- spike issue from vectors, 518–519

 banner advertising
 

- creating in Flash, 107–109
- creating in Photoshop, 101–103
- GIF image use for, 91

 BaseButton.selected property, 395–396  
 behaviors
 

- ActionScript vs., 164
- panel in Flash, 164

 Bezier curves, 67–68, 307–309  
 Bezier, Pierre, 67  
 bit depth, 207–209  
 bitmap images, 51–52
 

- in Flash, 82–84
- tracing in Flash, 85–87

 bitmap layers, importing as movieclips, 103  
 Bitmap Properties dialog box, 88–89  
 blend, shape tween settings, 290

blend modes, 155
 

- applying, 134–136
- filters and, 129–136
- manipulating static text with, 253–255

 block elements vs. inline elements, 436  
 bold styles, in text fields, 246  
 bold tag (<b>), 264  
 Book Category list box, filtering searches in, 194–195  
 break statements, 189  
 break (<br>) tag, 264  
 Bringham, Robert, 240  
 Brush Mode, option modifiers, 65  
 Brush tool, 53
 

- options, 65–66
- vs. Pencil tool, 65

 bunny bits, library of, 127–128  
 Bunnies Theatre cartoons, 127–128  
 Bush, Vannevar, 508  
 button
 

- adding sound to, 218
- aligning, 226
- bar, creating, 143
- giving instance name, 226
- library, 225
- symbols, 115–116

 Button component, 391–396
 

- changing appearance of, 396–398
- events available to, 396

 bytesLoaded values, 522

## C

Capellari, Claudio, 206  
 captions
 

- syntax for, 378–381
- Timed Text, XML for, 366–369
- XML for video, 378–383

 <cartoon> elements, 453–454  
 Cascading Style Sheets (CSS). *See* CSS (Cascading Style Sheets)  
 <cast> elements, 454–455  
 center point control (Gradient Transform tool), 58  
 changeFunction() declaration, 470  
 character design, 322  
 Character Embedding dialog box, 270–272  
 character position, 247  
 check boxes, interacting with (exercise), 402  
 CheckBox component, 401–402  
 Check syntax button
 

- in ActionScript 2.0 vs. 3.0, 196–200

 class files, ActionScript, 190  
 class identifier, for library items, 125–126  
 class selectors vs. element selectors, 434–437

- classes, objects defined by, 165–166
  - CLICK event handlers, coding, 230
  - CMX Suite, 130
  - code, commenting, 175–176
  - code blocks, in frames, 17
  - code coloring, ActionScript, 170
  - codec (enCODer/DECoder), 209, 211
  - Color panel, 56
  - Color Picker
    - component, 402–404
    - in Property inspector, 19–20
  - colors
    - basic Color palette, 74
    - changing for layer outline, 28
    - changing static text, 253–255
    - changing value on Mac, 75–76
    - color models in Flash, 72–109
    - gradient tricks, 78–79
    - panel, 60–61
    - persistent custom, 76–78
    - selection and options tools, 53
    - window, 74–75
    - ComboBox component, 404–407
  - commenting code, 175–176
  - Compiler Errors panel
    - syntax errors reported in, 199
  - Compiler Errors tab, Property inspector, 181–182
  - Component Inspector
    - changing component parameters in, 393
    - setting FLVPlayback parameters in, 358–359
  - components
    - finding packages for, 400
    - styling, 398–401
  - Components panel, 392
  - compression. *See* file compression
  - compression settings, MP3 format, 212
  - conditional statements, ActionScript, 185–189
  - Constan, Randy (Peter Pan), 120
  - constants, ActionScript 3.0, 175
  - content types, 482–483
  - context menu, 17
  - Convert to Symbol dialog box
    - advanced options in, 113–114
    - opening, 63
  - copy motion as ActionScript 3.0, 331–333
  - crashing text, 247
  - crayons, 61–62
  - Create Shape Tween choice, 289
  - createLetter function, 528
  - CSS (Cascading Style Sheets)
    - applying formatting to, 431–433
    - Flash and, 426–449
    - loading external, 445–448
    - power of, 427–433
    - style properties in, 427–428
    - styling applied to series of <li> tags, 431
    - using styles in (exercise), 428–430
  - cue points, adding to FLV files, 378
  - Curtis, Hillman, 108, 509
  - curves, optimizing, 518–519
  - Custom Ease In/Ease Out dialog box
    - areas of, 304–305
    - custom easing in, 304–310
    - how grid works in, 305–306
  - Custom Ease In/Ease Out editor, 304–310
- ## D
- data types, ActionScript 3.0, 180–182
  - DataGrid component, 407–408
  - Davis, Joshua, 117
  - de Visser, Martin, 341
  - deinterlace area, Encode Video panel, 346
  - Detect Flash Version option, 549
  - Device Central CS3, 474
    - accessing through Adobe Bridge, 481
    - choosing specific device in, 477
    - creating Flash documents with, 480–483
    - Emulator tab in, 484–486
    - Group By button in, 480
    - new to CS3 Studio, 476–480
    - publishing mobile movies in, 487–489
    - returning to Flash from, 487–489
    - specific searches in, 479
  - device fonts, 243–244. *See also* fonts; fonts and typefaces
  - Device Profiles panel, 478
  - Device Sets panel, removing set from, 480
  - dictionaries, 274
  - Difference mode, 136
  - Dimensions option, 549
  - Director, 537
  - display lists, 262
  - Distort option, Free Transform tool, 287
  - Distribute to Layers, rules for using, 141–142
  - Distributive blend, shape tween settings, 290
  - documents, managing properties in, 11
  - Document class, 190
  - Document Properties dialog box
    - changing movie frame rate in, 311
    - managing document properties in, 10–11
    - shrinking stage to video in, 356
  - documents
    - managing properties in, 10
    - setting preferences and properties, 9–14
  - dot (.) notation, 177–178
    - in E4X, 457
    - in `setStyle()` method, 436–437
  - drawers, in panel collapse process, 8

- drawing
    - objects, 52
    - tools, 63
  - Dreamweaver CS3, 373, 476
  - Drop Shadow filter, 130–133
  - DV format, 342
  - dynamic data (XML), Flash and, 452–471
  - dynamic text box, in status layer, 227
  - dynamic text fields
    - controlling events, 276
    - formatting (exercise), 256–263
    - properties, 256
    - selectively formatting, 261
    - vs. static text fields, 255–256
- E**
- E4X (ECMAScript for XML)
    - accessing data in XML, 457–462
    - filtering with comparison operators, 461–462
    - returning film titles, 462–463
    - viewing attributes in, 458–460
  - easing
    - custom, 304–310
    - shape tweens, 289
    - with standard easing controls, 301–303
  - Ecma International, ECMA-262 specification by, 160
  - Edit Format Options button, 246
  - Edit Grid, 139
  - Edit Multiple Frames button
    - adjusting keyframes with, 316–317
    - timeline, 314
  - Elapsed Time indicator, 311
  - element selectors
    - vs. class selectors, 434, 436–437
    - exercise using, 434–435
  - The Elements of Typographic Style*, 240
  - embedFonts property, 442
  - Emulator tab, 485
  - Encode Video panel
    - deinterlace area, 346
    - setting video data rate in, 344–345
  - Encoding Settings dialog box
    - Crop and Resize tab, 348
    - opening, 344
    - preset encoding profile in, 345–346
    - setting values in, 347–348
  - encoding values, 347
  - ENTER\_FRAME event, 232–233
  - Envelope option, Free Transform tool, 287
  - equality operator (==), 185
  - Eraser tool, 66–67
  - escape sequences, 258–260
  - Event class instance, required for event handling, 171–173
  - event handlers
    - in ActionScript, 171–173
    - coding, 228–236
    - EventCOMPLETE, 470
  - Event sound, 214
  - Event.ID3 event, alert raised by, 230–231
  - EventListeners, 521
  - events
    - controlling with dynamic text fields, 176
    - handling in ActionScript, 171–173
  - exercises
    - creating animation (authors under glass), 147–148
    - creating soft masks, 151–155
    - element selectors, using, 434–435
    - formatting dynamic text fields, 256–263
    - hyperlinks, 275–276
    - interacting with check boxes, 402
    - making a frog disappear, 275–276
    - Peter Pan, 120–121
    - static text, 248–255
    - styles in CSS, using, 428–430
    - text scrolling, 277–280
  - Export device sounds option, 547
  - Export dialog box, 539
  - Export GIF dialog box, 540
  - Export Movie dialog box, 530
  - external sound. *See* sound
  - ExternalInterface class, 452
  - eyeball icon, 27
  - Eyedropper tool, 78
- F**
- file compression, 387
    - applying settings, 91
    - changing, 89
    - checking type used, 342
    - handling in Flash, 88–91
  - file size
    - balancing, 108
    - vector vs. bitmap images, 51
  - file types, on Web, 537–542
  - <film> elements
    - adding <title> elements to, 454
    - adding to <cartoon> root element, 453–454
  - filters
    - applied through ActionScript, 129
    - applying in Flash, 130
    - applying to static text and tweening, 248–250
    - blend modes and, 129–136
    - in Flash, 129
  - Finish Video Import screen, 355

- Fireworks CS3
  - bitmap images, 51
  - importing documents into Flash, 93–95
  - importing GIF files into, 541
  - as planning aid, 517–518
  - round tripping feature in, 84
  - vector images, 51
- Fireworks PNG Import Settings dialog box, 94–95
- Flash. *See also* Flash CS3
  - alignment options, 551
  - audio in, 210–216
  - audio formats and, 207
  - common file formats in, 209
  - CSS and, 426–449
  - and devices, 474–475
  - evolution of, 537–538
  - going mobile in, 474–503
  - guidelines, 510
  - importing GIF animation into, 542
  - importing video into, 351–356
  - mass editing in, 314–316
  - MP3 and, 209–210
  - optimizing content for use in video, 527–531
  - publishing mobile movie, 487–489
  - sound types in, 214
  - video in, 340–387
- Flash 8, Object Drawing mode, 58–59
- Flash 9 vs. Flash mobile, 475
- Flash authoring environment interface, 6–7, 46
  - exploring panels in, 15–23
  - opening, 5
- Flash Color Picker, 74–76
- Flash CS3, 237, 471
  - animation in, 284–336
  - audio formats and, 207
  - banner advertising, 107–109
  - bitmap images in, 82–84
  - changes in, 53
  - Create Shape Tween choice in, 289
  - drawing in, 63–72
  - dynamic data (XML) and, 452–471
  - font formats supported by, 243
  - GIF files in, 91–93
  - going mobile in, 474–503
  - graphic objects, 52
  - graphics, 50
  - optimizing JPG images, 88–91
  - playing an FLV in, 351–386
  - spell-checking feature, 273–274
  - Start page in, 4–7
  - text in, 240–281
  - tracing bitmaps in, 85–87
  - using GIF files in, 91–93
- Flash CS3 UI components. *See also individual component names*
  - building interfaces with, 390–423
  - skinning, 396–398
  - styling, 398–401
  - weight added to movies by, 396
- Flash documents
  - accessing MovieClip class members, 166
  - creating, 4–7
- Flash Exchange link, to Adobe web site, 5
- Flash Lite 2.0, Flash mobile use of, 475
- Flash movies. *See also movies*
  - creating for mobile devices, 480–483
  - general guidelines for, 510
  - optimizing and fine-tuning, 516–521
  - publishing, 536–555
  - testing mobile, 484–486
  - Web formats for, 536–542
- Flash Only template, 549
- Flash Platform, evolution of, 159
- Flash Player
  - functionality in new versions, 537–538
  - playing audio and video with, 338
  - preparing and using alpha channel video, 367–369
- Flash Player 9, strict (or strong) data typing in, 182
- Flash stage, distributing buttons to, 226
- Flash timeline, exporting as GIF animation, 540–541
- Flash Video (FLV) Encoder, Flash 8 Professional, 340
- Flash video players, 341
- Flex Builder 2, SWFs produced by, 159
- Flick, Chris, 130
- Fluffy Flash video player, web site, 341
- FLV (Flash video)
  - and cue points, 378–383
  - connecting TV to, 361–364
  - converting existing video to, 342–350
  - encoding, 342–350
  - filename extension, 341
  - playing in Flash CS3, 351–386
- FLVPlayback component
  - control components, 364–365
  - in Flash 8 Professional, 340
  - manually connecting FLV to, 357
  - setting content path to, 360
  - setting parameters for, 358–359
- FLVPlaybackCaptioning component, 366
- focus rectangle, 496
- folders, 28–29
- Font Rendering method, 247
- fonts, 281. *See also device fonts*
  - effect on SWF file size, 242
  - embedded, 442–444
  - <font> tag, 264, 431



- outlines, embedding, 269–272
  - setting size, 246
  - symbol, embedding fonts with, 271–272
  - and typefaces, 241–243
- for each..in statement, in E4X filtering, 462
- Foundation ActionScript 3.0 with Flash CS3 and Flex 2*, 158, 190, 331
- Foundation ActionScript Animation*, 331
- Foundation Flash Applications for Mobile Devices*, 475
- frame rates
  - default in Flash movies, 16
  - Frame Rate indicator, 311
  - setting in Encode Video panel, 346
- frames, 16–18. *See also* keyframes
  - editing multiple, 314–317
  - options for, 17
  - in timeline, 7
- Free Transform tool, 55–56
  - adjusting picture frames with, 121
  - mastering white dot in, 56
  - Rotate and Skew option, 300
  - scaling and rotating objects with, 114
  - stretching shapes with, 286–287
- friends of ED, web site, 539
- fromCharCode() method, String class, 528
- From After Effects to Flash: Poetry in Motion Graphics*, 375
- Frutiger, Adrian, 245
- fscommand2() functions, 496
- FSCommands, 476
- Full Screen button, making functional, 373

## G

- GalaxyGoo, web site, 191
- gallery of images, adding to mobile applications, 494
- Gallery page, setting Listener to return to, 501–502
- Georgenes, Chris, 319–321
- GIF animation
  - creating in Fireworks CS3, 541
  - exporting Flash movie as, 539–541
  - filename extension, 537
  - importing into Flash, 539–542
  - importing into movieclips, 92–93
  - used before Shockwave, 539
- GIF files, using in Flash CS3, 91–93
- GIF images, 91–92
- glint, adding to Over frame, 329
- gotoAndPlay(10) method, 523
- gradient
  - changing colors, 61–62
  - filling a span with, 80
  - overflow options, 79

- Gradient Transform tool, 56–58
  - altering gradients with, 296–297
  - controls, 58
  - filling shapes with, 57
- graphic symbols, 115
  - changing displayed frame of, 318
  - vs. movieclips, 318
  - swapping across keyframes, 315–316
- graphics, Flash CS3, 50
- greater than (>) operator, 185
- greater than or equal to (>=) operator, 185
- Green, Tom, 334–335, 375
- Grid dialog box, editing Grid in, 139
- Group By button, Device Central CS3, 480
- guide layer, 25, 325
- guides, 139–140
- Guides dialog box, 140

## H

- Hanna, William, 341
- Head layer, formatting a mobile application, 492
- Help menu, user manuals in, 22–23
- Help panel
  - accessing and using, 192–193
  - component styles under class entry in, 398
  - keywords and reserved words in, 179
  - search tactics, 193–195
- Henry, Kristin, 191
- Hide Timeline button, 16
- :hover pseudo-class, 441
- HTML (HyperText Markup Language), 538–539
  - vs. ActionScript, 538
  - alignment selection, 551
  - filename extensions, 537
  - formatting, 264–266
  - page, 508
  - using for hyperlinks, 267–268
  - tab (Publish Settings dialog box), 549–551
  - styling tags, 432–433
  - vs. XML, 453
- Humber School of Technology, School of Media Studies, 341
- hyperlinks
  - absolute and relative, 267
  - exercise, 275–276
  - and Flash text, 266–269
  - option, 248
  - styling, 440–442
  - triggering ActionScript with, 268–269
  - using HTML for, 267

- I**
- icons, 27–28
    - Search Devices, 478
    - Symbol Editor, 32
  - ID3 tags, MP3 file metadata in, 227
  - if..else statements, 189, 194
  - if statements, 186–189
  - Illustrator CS3
    - Device Central available in, 476
    - importing documents, 96–100
    - tool of choice for type design, 243
    - vector images, 51
  - Illustrator File Importer, 96–100
  - image tag (<img>), 264–265
  - images
    - changing compression of, 89
    - converting imported to symbol, 322–325
    - filling objects, 80–81
    - types, 51
    - vector vs. bitmap, 51–52
  - ImpactNormal font, 443
  - Import Bitmap dialog box, 89
  - Import statement, 400
  - Import Video wizard, 351–356
  - inheritance, 439–440
  - Ink Bottle tool, 53
  - input text fields, properties, 256, 263
  - Instance Name field, 166
  - interlacing, 346
  - Internet, Flash “love-hate” relationship with, 506–510
  - Internet Protocol (IP), 508. *See also* Transmission Control Protocol/Internet Protocol (TCP/IP)
  - !isPlaying variable, 232
  - italic
    - applying in CCS, 430–431
    - applying to text fields, 246
    - tag (<i>), 265
- J**
- JPG (JPEG)
    - optimizing images, 88–91
    - quality option (Publish Settings dialog box), 547
- K**
- kerning, auto kern option, 248
  - key frame areas, Encode Video panel, 347
  - key label, changing, 502
  - keyframes. *See also* frames
    - adding to a timeline, 17–18, 172
    - inserting and removing, 286
    - swapping graphic symbols across, 315–316
  - keywords and reserved words, in Help panel, 179
  - knob, dragging in Seek knob layer, 226–227. *See also* Seek knob layer
  - Kricfalusi, John, 334–335
- L**
- Label component, 408
  - Layer content menu, opening, 144
  - layer modes, assigning to layers, 25
  - Layer Properties dialog box, 214–215
  - Layer Visibility icon, eyeball icon as, 27
  - layers, 25–29
    - adding content to, 26
    - adding to mobile application, 492
    - grouping using folders, 28–29
    - hiding, 28
    - icons used in, 27–28
    - of timelines, 24–29
  - leading, applying in CCS, 431
  - Leggett, Richard, 475
  - less than (<) operator, 185
  - less than or equal to (<=) operator, 185
  - letter spacing (tracking), 247
  - <li> tag, applying italic to, 430–431
  - library
    - in Property inspector, 21–22
    - playing sounds from, 219
    - sharing, 125–127, 155
  - Library Preview pane
    - GIF files in, 91
    - sampling colors from, 78
  - Lighten mode, applying, 136
  - line breaks, adding in ActionScript, 259–260
  - Line Numbers, in Action panel context menu, 163
  - Line Type option, 247
  - linear gradient, filling square with, 58
  - linkage identifier, assigning, 113
  - Linkage Properties dialog box
    - adding items to shared library in, 125–126
    - specifying class in, 220
  - linked files, publishing movies with, 553–554
  - lip-synching techniques, 322
  - List component, 409–410
  - list item tag (<li>), 265
  - Load Order option, 546
  - load() method, 520–521
  - Loader class, 169, 520–521
  - Loader object, 520
  - loadSong() function, 231–232
  - Local playback security option, 547
  - logical NOT operator, 232
  - Longo, Ryan, 206
  - Lorem ipsum, origination of, 243
  - lossy format, JPG format as, 88

**M**

- Macintosh Color Picker, 75–76
- Magnification menu, zooming the stage from, 13
- manuals. *See* user manuals
- masks, 155. *See also* soft masks
  - adding motion to movie, 149
  - applying filter to text, 150
  - applying locks to, 145
  - creating simple, 144–148
  - creating soft (exercise), 151–155
  - layer, 25
  - masking and, 143–150
  - using strong fonts as, 148–149
  - using text as, 148–150
  - without a mask layer, 153–155
- max data rate area (Encode Video panel), 346
- Maximum Characters property, 263
- Maynard, Jay (Tron Guy), 323
- McSharry, Sean, 158, 190
- members of a class, in ActionScript, 165–166
- methods, for ActionScript object types, 169–171
- mobile application
  - adding gallery to, 494–495
  - adding layers to, 492
  - constructing, 489–493
  - creating, 491–493
  - writing ActionScript for, 495–502
- mobile devices
  - bitmap and vector rules for, 495
  - soft keys on, 490
- mobile document, creating, 480
- Modify menu
  - Group option, 54
  - Ungroup option, 54
- Modify Onion Markers button, 312
- Modify Remove Transform, 56
- motion guide layer, changing normal layer into, 325–326
- motion guides, 25
  - tweening along curves with, 325–327
- motion tweening, 297–298
  - between two keyframes, 316
  - changing objects with, 18
  - creating animations with, 18
  - creating twinkling stars with, 36
  - custom easing, 304–310
  - easing, 301–303
  - effects, 322–325
  - entities required by, 297
  - lack of shapes support for, 288
  - properties, 299
  - rotation, 298–299
  - scaling, stretching, and deforming, 300
- MouseEvent.CLICK event handlers, coding, 228–230
- MOV format, 342, 537
- Movie clips
  - icon for Symbol Editor, 32
  - in New Symbol dialog box, 38
- movie mask, adding motion to, 149
- MovieClip
  - class, methods in, 169–171
  - instance, exploring, 169–171
  - symbols, 116–117
  - unsettable properties in ActionScript, 169
- MovieClip.currentFrame property, 174
- MovieClip.gotoAndPlay() method, 171
- movieclips
  - acting as draggable buttons, 229
  - giving instance name, 221
  - vs. graphic symbols, 318
  - importing bitmap layers as, 103
  - importing GIF images into, 92–93
  - manipulation objects in, 384–386
  - nesting, 116
  - playing sounds with, 220–222
  - setting properties via ActionScript, 168
  - testing, 91
- movies. *See also* Flash movies
  - adding audio to, 41–43
  - adding grass and lake, 35
  - adding moon over lake in, 37–41
  - adding mountains and color to, 32–33
  - building, 29–45
  - composition of, 16
  - creating illusion of depth in, 33–34
  - creating moon shadow in, 38
  - creating moonrise over lake, 44–45
  - creating twinkling star, 36
  - length range, 16
  - tasks to build, 29
  - testing, 43–44
- MP3 format, 207, 237
  - compression settings, 212
  - metadata in files, 227
  - perceptual encoding used by, 209–210
  - player, coding, 228–236
- MPG/MPEG (Motion Picture Experts Group) format, 342
- Multiply mode, applying to image, 135–136

**N**

- \n (newline escape sequence), 259–260
- NetConnection, between player and server, 363
- New from Template dialog box, 5
  - Flash Lite 2.0 templates in, 480
- New Symbol dialog box, 69
  - use of Movie clip type in, 38
- Next button, coding, 233–234
- nextSong() function, coding, 233
- NumericStepper component, 411

## O

- O'Meara, Robert, 341
- Object Drawing mode, 58–60
- Object instance, creating, 432
- objects
  - classes defined as, 165–166
  - constraining proportions of, 56
  - moving, 54
  - registration points, 31
  - scaling, skewing, and rotating, 55–56
  - turning on snapping, 294
  - working with ActionScript, 164–173
- onion skinning, in animation, 311–314
- Open link, on Start page, 5
- operator precedence, 183–185
- operators, ActionScript, 182–185
- Optimize Curves dialog box, 87, 518
- Options menu, 546–547
- orientation, changing static texts, 247
- Oval Primitive tool, 52
- Oval tool, using in Object Drawing mode, 59–60
- Over frame, adding glint to, 329
- overflowing process, 78
- Override sound settings option, 547

## P

- packages, finding for components, 400
- packets, 508
- Paint Bucket tool, 53
  - filling shapes, 57
  - Lock Fill feature, 80
- panel collapse process, 7–8
- panels, managing, 7–8
- paragraph tag (<p>), 265
- Password option, 547
- paste commands, for layers, 26–27
- path, linked series of objects as, 178
- pauseSong() function, 230
- Pen tool, 67–69
- Pencil icon, 28
- Pencil tool, 63–65
- Peters, Keith, 331
- Photos button, in mobile applications, 493
- Photoshop CS3
  - bitmap images, 51
  - Device Central available in, 476
  - importing documents into Flash, 100–106
- Photoshop File Importer, 104–106
- Pilotvibe, 216
- pipe. *See* bandwidth
- placeholder text, generating, 243
- Playback choices, 549–550

- playhead
  - in timelines, 7
  - navigating to timeline frames with, 18
- playSong() function, 230, 232
- PNG image, importing into Flash, 94–95
- Popeye cartoons, organizing collection of, 453
- PostScript drawing applications, 243
- Preferences dialog box, 102
- Preferences panel, 9–10
- preloader, 522–526
- Previous button, coding, 233–234
- prevSong() function, coding, 233
- primitives. *See* Oval Primitive tool; Rectangle Primitive tool
- Programming ActionScript 3.0 book (Help panel), 182
- ProgressBar component, 412–413
- properties
  - accessing with ActionScript, 166–169
  - vs. variables, 178–179
- Property inspector, 12
  - accessing Document class through, 190
  - collapsing panels in, 8
  - Compile Errors tab, 181–182
  - concept of, 18–20
  - fonts in, 244
  - Instance Name field, 166
  - setting FLVPlayback parameters in, 358–359
  - Smooth option, 66
  - and static text, 245–248
  - using stream or event sound in, 216
- PSD File Importer, 101
- pseudo-classes, 440–442
- Publish Preview menu, 553
- Publish Settings dialog box, 372
  - Export device sounds setting in, 547
  - file types in, 544
  - Flash Only template in, 549
  - Flash settings in, 545
  - HTML alignment selection in, 551
  - HTML tab in Flash CS3, 548
  - JPEG quality option in, 547
  - launching to publish movies, 543–544
  - Navigate buttons in, 545
  - options in, 547–551
  - Playback choices in, 549–550
  - Profile buttons in, 544
  - publishing Lake Nanagook movie, 551–552
  - Quality menu in, 550
  - Scale menu in, 551
  - setting ActionScript version in, 546
  - setting Window Mode in, 551
  - Show warning messages box in, 551
  - Version menu in, 546
- Publish Settings panel, audio options in, 213

**Q**

quality area (Encode Video panel), 346  
 Quality menu, 550  
 QuickTime, 542
 

- creating movies in Flash, 527–531
- Export Settings dialog box, 530–531
- formats, 207

**R**

radial gradient tool, 57  
 RadioButton component, 413–414  
 radius handle control (Gradient Transform tool), 58  
 randomBetween() function, 528  
 rapid prototyping tool, Fireworks CS3 as, 517–518  
 Rectangle Primitive tool, 52  
 registration points, of objects, 31  
 relational operators, 185  
 Render Text as HTML option, 247  
 reserved words, in Help panel, 179  
 resize handle control (Gradient Transform tool), 58  
 Riva FLV player, web site, 341  
 root element, XML documents, 453  
 rotate handle control (Gradient Transform tool), 58  
 Rotate property, 299  
 rotation, 298–299  
 rotoscoping video images, 329–331  
 round tripping, in Fireworks CS3, 84  
 routers, 508  
 rulers, aligning objects on stage with, 139

**S**

salmon recipe, styling, 426–449  
 sample rates, 207–209  
 Scale menu, 551  
 scaling, skewing, and rotating objects, 55–56  
 Script Assist feature
 

- Actions panel, 200

 script editors, 161  
 Script navigator, 162  
 Script pane
 

- Action panel, 162
- adding code to, 221, 522–523

 Script time limit option, 547  
 Script window, script editor lock out by, 161  
 scroll bars, 415  
 scroll buttons, symbol buttons as, 279–280  
 ScrollPane component, 415  
 scrubbing, across timelines, 18  
 Search Devices icon, Available Devices bar, 478  
 Seek bar, 226  
 Seek knob layer, 226  
 Seek slide layer, 226

Seek slider, coding, 234  
 Selectable option, static text, 247  
 Selection tool, 53–55  
 semicolons (;), in ActionScript, 175  
 shape hints, guiding anchor points, curves, or lines with, 292–296  
 shape tweening, 285–297
 

- altering gradients, 296–297
- altering shapes, 290–292
- anchor points in, 291–292
- easing in and out, 289, 301–303
- modifiers, 289–290
- scaling and stretching, 286–289
- shape hints, 292–296

 shapes, altering, 290–292  
 shared libraries. *See* library  
 Shiman, Jennifer, 127–128  
 shimmy() function, 528–529  
 Shockwave products, by Macromedia, 537  
 Show All Layers As Outlines icon, 28  
 Show Border Around Text option
 

- input text fields, 263
- not available for static text, 247

 Show Grid, aligning objects on stage with, 138–139  
 Show warning messages box, 551  
 Shroeder, Dave, 216–217  
 skin style, selecting, 354–355  
 skinning, UI components, 396–398  
 Skip Intro button, 506–507  
 Slider component, 416–417  
 slideshow
 

- bringing to life, 465–468
- building with XML, 463–471
- changeFunction() declaration, 470
- changing reference to button event handler, 470–471
- EventCOMPLETE event handler, 470
- geocaching photos for, 465–468
- handling events and populating combo box, 465–468
- walkthrough of “hard-wired” movie, 464

 Smooth option, in Property inspector, 66  
 Snap Align, aligning objects with, 138  
 snapping, turning on, 294  
 soft key events, creating listeners for, 497  
 soft masks, 155. *See also* masks
 

- creating (exercise), 151–155

 SoftKeyID, mobile application settings, 492  
 sound, 237. *See also* audio
 

- adding to buttons, 218
- loading external, 222–223
- playing, 219–223
- turning remote on and off, 223–224

 Sound Properties dialog box, 211–213  
 SoundChannel class, stop() method in, 224  
 soundTransform property, 232

## INDEX

- SOUND\_COMPLETE event handler, 232
  - span tag (<span>), 265
  - speed-dial buttons, in mobile applications, 493
  - spell checking, 273–274, 281
  - squares, moving, 54
  - stacking order, 140–143
  - stage
    - aligning objects on, 138–139
    - changing size or color of, 11
    - grouping content on, 137–138
    - in authoring environment interface, 7
    - managing content on, 136–143
    - side-by-side image comparisons, 14
    - zooming, 13–14
  - Start page, 4–7
  - static text
    - applying filters and tweening, 248–250
    - configurable items, 245–248
    - exercises, 248–255
    - Property inspector and, 245–248
  - static text fields
    - vs. dynamic text fields, 255–256
    - properties, 248
  - status layer, dynamic text box in, 227
  - Stiller, David, 489–490
  - stop() method, in SoundChannel class, 224
  - Stream sound, 214
  - streamhead, adding to timeline, 512
  - streaming buffer, creating, 512
  - streaming movie content, 510–512
  - strict (or strong) data typing, 180–182
  - Stroke Style dialog box, settings in, 71
  - style inheritance, 439–440, 449. *See also* inheritance
  - StyleSheet class, importing, 429
  - StyleSheet.setStyle() method, 430
  - styling components, 398–401
  - Subselection tool, 69–70
  - SWF file
    - advantages of small, 113
    - filename extension, 537
  - switch statement, 189
  - symbol buttons, 279–280
  - Symbol Editor
    - Movieclip icon for, 32
    - opening, 117
  - Symbol Properties dialog box, 119
  - symbols, 155. *See also* button symbols; graphic symbols;
    - MovieClip, symbols
    - 9-slice scaling and, 118–124
    - adding to shared libraries, 126–127
    - creating, 113–114
    - editing, 117–118
    - essentials of, 113–118
    - and libraries, 112–115
    - sharing, 124–127
    - swapping graphic across keyframes, 315–316
    - types of, 115–117
    - updating changed in shared library, 127
    - using from another movie, 124–125
  - syntax
    - checking, 196–200
  - syntax errors
    - checking for, 196–200
- ## T
- \t (tab escape sequence), 259–260
  - tabular data, displaying with HTML, 265–266
  - tags, creating custom, 437–439
  - TCP/IP. *See* Transmission Control Protocol/Internet Protocol (TCP/IP)
  - templates, creating documents with, 5
  - Test Set folder, creating custom groupings in, 480
  - testing panels, Device Central, 485
  - text
    - adding color in CCS, 431
    - adding fields with ActionScript, 261
    - aligning, 246, 281
    - auto-widen mode, 246
    - changing orientation of static, 247
    - character position, 247
    - field types, 245–263
    - format tag (<textformat>), 265
    - hyperlinks support, 266–267
    - import options, 102
    - scrolling exercises, 277–280
    - setting color, 246
    - type, 246
  - text masks, creating Places intro screen with, 148–150
  - TextArea component, 417–418
  - TextField class, 169
  - TextField.setTextFormat() method, 261
  - TextField.text property, input text fields, 263
  - TextFormat class, 528
    - declaring and setting variables, 400–401
    - setting properties, 260–263
  - TextInput component, 418–419
  - this keyword, ActionScript use of, 170–171
  - <title> elements, adding to <film> elements, 454
  - Thomas, Adam, 183
  - Thumbs movieclip, activating buttons, 498–500
  - TileList component, 419–420
  - Timed Text (TT) XML, for captions, 366–369
  - Timeline panel
    - checking settings, 288
    - dashboard strip, 310–311

- timelines, 15–16
    - in authoring environment interface, 7
    - combining, 318–322
    - layers feature of, 24–29
    - looping, 202–203
    - in movieclips, 169–171
    - navigating to frames in, 18
    - pausing the main, 201–202
    - scrubbing, 7, 18
    - for symbols, 32
    - tweening inside butterfly graphic symbol, 327
  - Timer object, creating in Flash, 527–528
  - Title box, in Document Properties dialog box, 11
  - tools, 53–56. *See also* Free Transform tool; Oval tool; Rectangle tool; Selection tool
  - Tools panel, 52–63
    - areas in, 20–21
    - selecting objects in, 19–20
    - turning object snapping on, 294
  - Trace Bitmap dialog box, 85–86
  - trace() function, 521
    - ActionScript, 167–168
    - seeing operators in action with, 182
  - traced images, optimizing, 87
  - tracing bitmap images, in Flash, 85–87
  - tracking. *See* letter spacing (tracking)
  - Transform panel, motion tweens use of, 300
  - transformation point, scaling shapes with, 287
  - transformations, removing, 56
  - Transmission Control Protocol/Internet Protocol (TCP/IP), 508, 532
  - transparency, adjusting for imported photos, 322–325
  - Tween property, 288
  - tweening, 46
    - along curves, 325–327
    - masks, 327–328
    - static text with filters applied, 248–250
    - updating multiple symbol properties, 309–310
  - typeface, 242. *See also* device fonts; fonts
  - typographic tools, CS3, 240
- U**
- UI components. *See* Flash CS3 UI components
  - UIComponent.setStyle() method, 401
  - UILoader component, 420–422
  - UIScrollBar component, 277–278
  - UIScroller component, 422–423
  - underline tag (<u>), 265
  - Uniform Resource Locator (URL), 508
  - Univers 55 font, 243
  - URLLoader class, 452
  - URLVariables class, 452
- Use device fonts, Font rendering method menu, 242, 245
  - user manuals, Flash, 22–23
- V**
- var keyword, 179
  - variables
    - naming, 228
    - vs. properties, 178–179
    - setting, 231–232
  - vector images, 51
  - vector morphing, shape tweens, 292
  - vectors (vector points), effect on Bandwidth Profiler, 518–519
  - Version menu, 546
  - video
    - applying captions to, 367–369
    - building custom controller, 364–365
    - codec (Encode Video panel), 346
    - converting to FLV format, 342–350
    - Flash content interacting with, 377
    - going full screen with, 371–374
    - hearing-impaired access to, 365–366
    - importing into movieclip, 375–377
    - live preview, 360
    - playing with alpha channel, 383–384
    - setting data rate, 346
    - streaming into video object, 363
    - trimming, 344
    - on the Web, 341
    - XML captions for, 376–383
  - video object, 361–362
  - Video Properties dialog box, 361–363
  - :void, using in functions, 182
  - volume control, 227
  - volume slider
    - coding, 235–236
    - layer, 227
- W**
- W3C (World Wide Web Consortium) specification, 426
  - WAV format, 207
  - waveforms, 207–208
  - Web. *See* World Wide Web
  - Web formats, for Flash movies, 536–542
  - web pages, 508
  - Webster, Steve, 158, 190, 331
  - white dot, mastering in Free Transform tool, 56
  - Window Mode, 551
  - workspaces, managing, 7–8
  - World Wide Web, evolution of, 508–509
  - WYSIWYG editor, Dreamweaver CS3 as, 538

## INDEX

### X

X-height, 245

XML (eXtensible Markup Language). *See also* dynamic data (XML)

building slideshows with, 463–471

captions for video, 387

file, loading, 456–457

vs. HTML, 453

loading a file, 456–457

markup languages and formats based on, 453

power of, 453–457

rules for using, 455–456

writing, 453

XMLList instance, 462–463

### Z

Zoom tool, 69