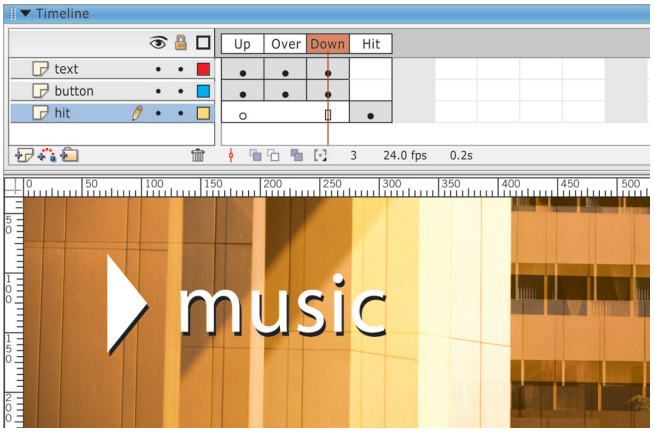
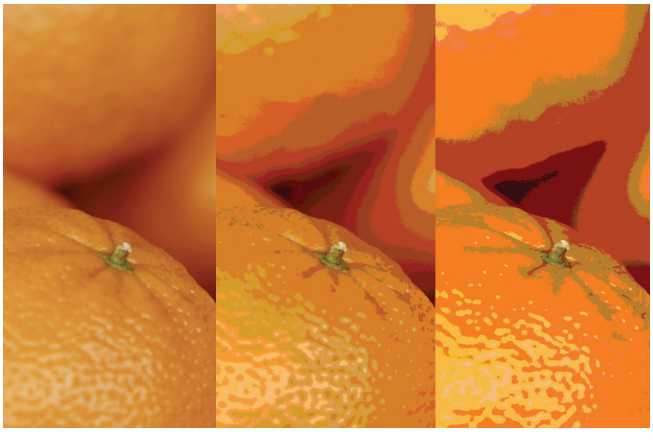
O'REILLY DIGITAL STUDIO

CD-ROM Included

Flash 8

Projects for Learning Animation and Interactivity





O'REILLY®

Rich Shupe & Robert Hoekman Jr.

Flash 8: Projects for Learning Animation and Interactivity

by Rich Shupe and Robert Hoekman, Jr.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Print History:

March 2006: First Edition.

Editor: John Neidhart

Production Editor: Genevieve d'Entremont

Copyeditor: Rachel Wheeler

Proofreader: Sada Preisch

Indexer: Johnna VanHoose Dinse

Cover Designer: Linda Palo

Interior Designer: David Futato

Illustrators: Robert Romano, Jessamyn Read, and Lesley Borash

This book uses RepKover™, a durable and flexible lay-flat binding.

0-596-10223-2
[C]

Working with Graphics

5

Flash's drawing tools are primarily vector-based, so they aren't ideal for editing bitmaps. Meaningful bitmap editing requires a program designed for that purpose, such as Adobe Photoshop or Macromedia Fireworks. Similarly, while Flash's drawing methods appeal to many designers, full-powered tools dedicated to this task, such as Adobe Illustrator and Macromedia FreeHand, are often used to create more complex vector-based illustrations. In this chapter, you'll look at how Flash can work with other applications to help you meet your project needs. This chapter includes several mini-projects to get you familiar with working with external graphic assets.

In this chapter

Importing Pixels

Although importing graphic assets is a basic task, there are a few subtleties when dealing with specific file formats and specific applications. To help you take advantage of the full range of features Flash has to offer, and to give you a complete design palette to work with, this section will briefly explain some of the ins and outs of importing pixel-based graphics.

NOTE

The graphics discussed herein are generically referred to as pixel-based or raster graphics, both as a means of categorizing topics and because that is the most common format for these types of graphics. However, you will soon see (especially when discussing Fireworks files) that some formats can contain both pixels and vectors.

A Few Words About File Formats

Many applications can create and edit pixel-based graphics. Adobe Photoshop will occasionally be referenced in this book, as it's the most common example. However, many (if not all) of the points discussed in this segment will apply to most pixel-editing programs.

NOTE

The term bitmap is often used to refer to a variety of similar, but not identical, items. The generic use of the term refers to pixel-based graphics, and a more specific usage can refer to the Windows file format BMP. Henceforth, the latter will be specified by the acronym, and "bitmap" will refer to a pixel-based graphic.

While Flash supports several bitmap file formats, including platform-specific formats such as Windows *BMP* and Macintosh *PICT*, here you will learn how to use the three most commonly used formats: *JPEG*, *GIF*, and *PNG*. It is outside the scope of this book to go into detail about these specific formats, but here is a very brief summary of the high points for each:

- JPEG is most often used for images that have continuous tones, such as photographs or gradients. JPEGs have a 24-bit color depth and can be compressed using a varying degree of quality settings. New to Flash 8, files using the *progressive* JPEG compression format can be imported, in addition to the more commonly used *baseline* JPEG format.
- GIF is usually used when an image has large areas of solid colors, or when a crisper look (possibly even at the expense of some anti-aliasing) is desired. GIFs can contain areas of 100% transparency, but not varying levels of transparency. GIFs can support up to 8 bits of color and can be compressed to varying degrees by limiting the number of colors available to an image. Figure 5-1 shows a continuous gradient, compressed in both JPEG and GIF. Note the banding in the GIF.

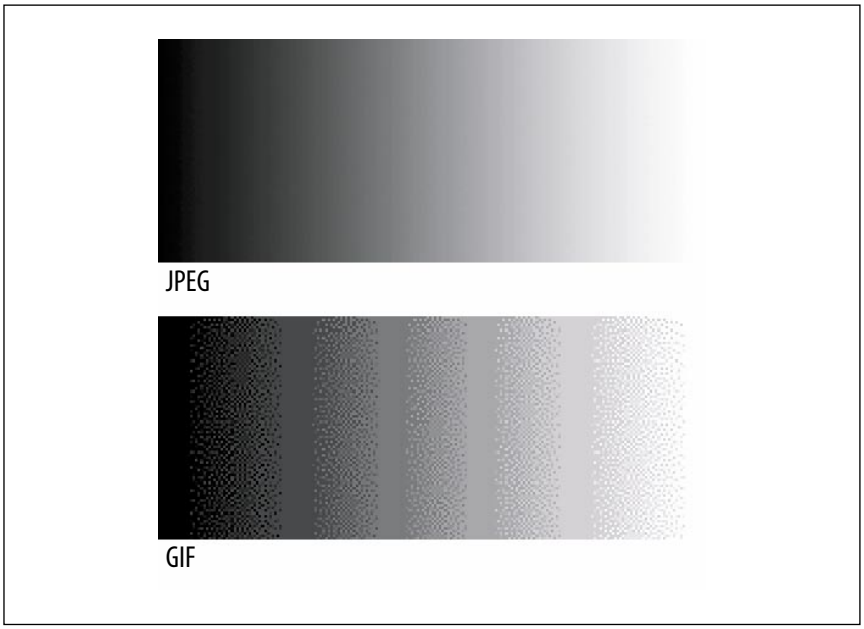


Figure 5-1. The JPEG file format is best suited to continuous-tone images. Note the banding of the gradient in the GIF at right.

- PNG is a lossless format for high-quality images that can support up to 48-bit color and 16-bit grayscale. It is primarily used when support for alpha data is required, as varying levels of transparency are possible. Figure 5-2 shows what the same image in both GIF and PNG formats looks like on a background. The Stage color is medium gray, and the GIF and PNG files have an identical appearance: a black circle on a

transparent background. When the graphics are placed on the Stage, the PNG edge between art and transparency is smooth. However, note the halo in the GIF. This is caused by the semi-transparent pixels of the anti-aliased edge of the circle. Since only 100% transparency is supported, partially transparent pixels are converted to solid pixels with partial color.

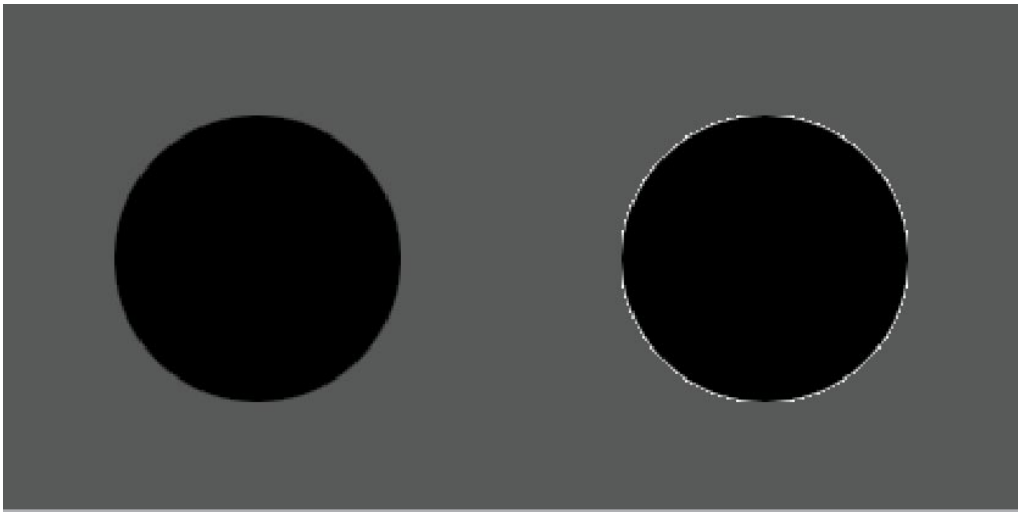


Figure 5-2. The PNG file format is optimal for varying levels of transparency. Note the halo in the GIF at right, caused by semi-transparent pixels.

Importing Standard Formats

When importing the standard versions of these file formats, as you are likely to do most often, Flash will automatically handle the file for you and import a single bitmap image. At the outset of the process, you can decide to import directly to the Stage, or to the Library. The latter is handy when you want to import many images at the same time.

Using the File→Import→Import to Stage menu command, for example, will prompt you with a standard operating system dialog that will allow you to find the file you want to import. Alternately, you can paste a bitmap into Flash from the clipboard, or drag an image in from your desktop.

NOTE

When you import or drag a bitmap into Flash, the filename, path, and certain properties of the external file are maintained, allowing you to update the file from within the Flash Library. This is convenient when you need to make changes to the external file. However, when you paste in a bitmap from the clipboard, Flash has no such information to maintain. A generic name of "Bitmap," followed by a sequential integer to avoid overwriting, will be applied.

When you import a bitmap, whether to the Stage or the Library, a Library item is automatically created. This is because bitmaps are typically large, and Flash wants to make working with them as efficient as possible. The item it creates acts like a symbol, in that you can drag multiple instances of the bitmap to the Stage, and any permanent edits made to that image will be reflected in the instances.

It will not be a symbol in every sense of the term, however, in that you cannot apply color effects and you cannot take advantage of some of the features built into graphic, button, and movie clip symbols. To gain these benefits, simply convert the bitmap to a symbol, the way you would any other relevant asset type.

WARNING

It's important to remember that if you create a symbol from a bitmap, the bitmap must remain in the Library. Do not delete the bitmap, thinking that it has been transformed into the symbol. Instead, think of a bitmap inside a symbol the way you'd think of a movie clip inside another movie clip. In both cases, if you delete the nested item, it will disappear from its related container.

Take a few moments to import the *cheesecake.jpg*, *black_circle_1.gif*, and *black_circle_2.png* sample files provided in the *05* folder of your working directory. (You'll find these files in a subdirectory called *Importing Pixels*.) Experiment with how they do and do not behave like symbols. Compare the transparency levels of the PNG and GIF files on a dark Stage color or background shape. In general, get a feel for working with bitmaps, and then save your file as *bitmaps.fla* in your *05* folder. You won't be using this file again for future exercises, but it will give you something to look at when learning about compression settings later.

Importing from Fireworks

NOTE

If you don't have it, you can install a trial version of Fireworks from this book's CD-ROM. If you don't want to install Fireworks, assets created in the program have been provided for you.

While Photoshop has earned the lion's share of the pixel-based graphics editing market, Macromedia has done a lot to ensure that Flash integrates tightly with its own products, such as Fireworks. This will be particularly pleasing to those who have purchased Studio 8, as Fireworks is part of the Studio suite.

If you use Fireworks, or are interested in trying it, you will likely find that its mixture of pixel-based and vector-based objects can be very useful. It likely won't serve as a substitute for Photoshop, but you may be willing to part company with ImageReady and use Fireworks in its place. Take a moment to open the *navigation.png* and *walk_cycle.png* files in the */05/Importing Pixels* folder. Poke around and see how pixels and vectors are used in the files.

Once you're ready to move on, import a Fireworks file into Flash to see how some of its unique features can be beneficial to Flash users:

1. Create a new, blank document and save it as *fw_navigation.fla* in the */05/Importing Pixels* folder.
2. Import the *navigation.png* file found in the same folder. You will see the dialog pictured in Figure 5-3.

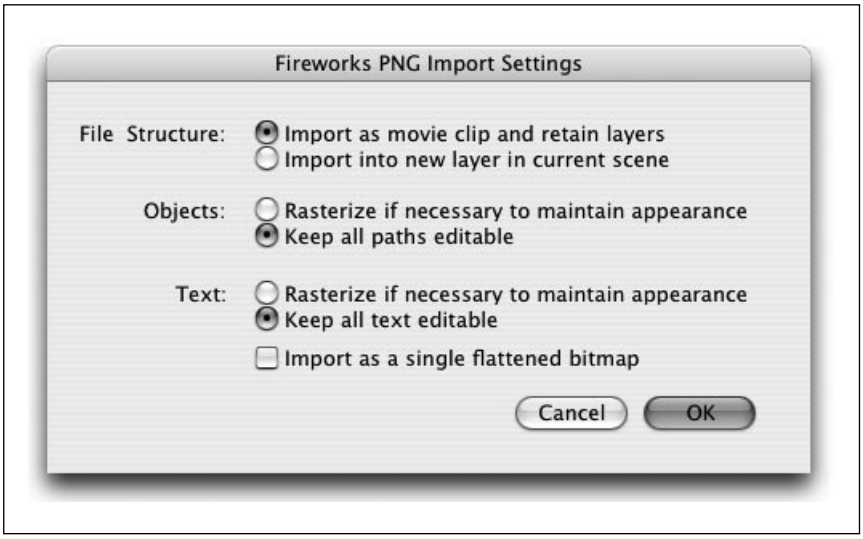


Figure 5-3. The Fireworks PNG Import Settings dialog allows you to choose to what degree the PNG will be preserved upon import

3. Match your settings to those pictured. The first setting will create a self-contained movie clip that contains everything imported from the *.png*. The second will keep vector shapes editable rather than converting them to pixels, and the third will do the same with text elements. As an alternative, you can always import a Fireworks file as a flattened bitmap, but you will lose the ability to edit the mentioned features.
4. OK the dialog and look at the result. In one fell swoop, you've got a navigation system fully contained in its own movie clip, complete with functioning buttons. Test your movie and roll over the navigation column. Once you see the buttons working, close the *.swf* and return to your *.fla* file.
5. Double-click on the movie clip and look within it. You will find several instances of a button, as well as three movie clip symbols. In the next chapter, you'll learn more about movie clips and buttons nested within other movie clips, and how to control them with ActionScript. For now, however, concentrate on how your file has changed upon importing this *.png* file.

6. Open the Library and see what you find. You will notice that it contains two bitmaps and a folder called *Fireworks Objects*. The bitmaps are the two graphics found at the top and middle of the navigation column—as you'll recall, bitmaps are automatically added to a file's Library for improved efficiency. Opening the folder reveals the three movie clips and the button you saw on the Stage earlier.
7. When you are finished experimenting, save your work and close the file. You won't need it again.

When you create a file in Fireworks, whenever possible, symbols you create there will be converted into Flash symbols upon import. Depending on how carefully you maintain your Fireworks files, these and other assets will be added to the Library, and the appropriately named folder you just saw will be created. This is a step toward organizing your Library, which you'll read more about later.

You will have a similar experience when importing Fireworks animations. Open your next Fireworks file to see the result:

1. Create a new, blank document and save it as *fw_walk_cycle.fla* in the */05/Importing Pixels* folder.
2. Import the *walk_cycle.png* file found in the same folder. You will again see the dialog pictured in Figure 5-3. Apply the same settings and click the OK button.
3. This time, you have a walk cycle. Test your movie and watch him walk.
4. You'll learn more about walk cycles in the next chapter, but for now, move on to working with the files you just imported. Save your work and close the file. You won't need it anymore.

Properties

An ActionScript *property* is a way of describing an object, much like an adjective. These descriptions can be obvious, like a movie clip's *_width*, *_height*, and *_rotation* properties, or they can be a bit subtler, like a movie clip's *menu* property, which helps you add a custom menu item to the contextual menu that pops up in the Flash Player when you right/Ctrl-click on the movie clip.

You may notice that some property names begin with an underscore, and others do not. The underscore is meant to indicate that the keyword is a property at first glance, and it was more commonly used in earlier versions of Flash. Knowing when to use the underscore and when not to use it can take a little practice, but it will become second nature after a short time.

At runtime, you can always get the value of a property, such as when checking to see how wide a movie clip is; you can usually set the value of a property, too, such as when setting the rotation of a movie clip to 90 degrees. Some properties, however, are read-only, such as the *_totalframes* property, which tells you how many frames a movie clip has. Since you cannot add frames to a movie clip at runtime, this property cannot be set.

Working with Pixels

While major pixel pushing is reserved for other applications, such as Photoshop or Fireworks, Flash still offers a few ways to edit bitmaps to creative effect.

Breaking Apart a Bitmap

In one way, breaking apart a Flash asset is equivalent to sending it one rung down the Flash evolutionary ladder. Depending on how you created your assets, a symbol can be broken into a group, which can be broken into a shape. The same is true of text: a word can be broken into letters, which can then be broken into shapes.

NOTE

If you haven't already read about the Modify→Break Apart command in Chapter 1, you may want to do so now.

In the case of bitmaps, breaking them apart lets you work with them in non-traditional ways. For example, you can select a portion with the Selection or Lasso tool and remove it, or you can use the Bush tool and paint a new color into it. These are behaviors commonly associated with shapes, but a bitmap that has been broken apart doesn't exactly behave like a shape. For instance, if you click a color hoping to select it, you will find that the entire bitmap is selected.

This is because the bitmap is still behaving as if it were intact, as though you simply modified the container in which it resides. This can be demonstrated using the *bitmap_break_apart.fla* file in the */05/Working with Pixels* folder, as shown in Figure 5-4. After breaking apart the bitmap, step 1 shows the upper-right corner of the bitmap selected by the Selection tool. Step 2 shows that selection deleted, seemingly removing that portion of the bitmap. Step 3 shows the lower-right corner being moved like a traditional shape. Finally, step 4 reveals that the bitmap is still seen in the previously deleted area. What's more, the bitmap continues in areas previously never occupied by its container. This demonstrates what is commonly called *bitmap tiling*.

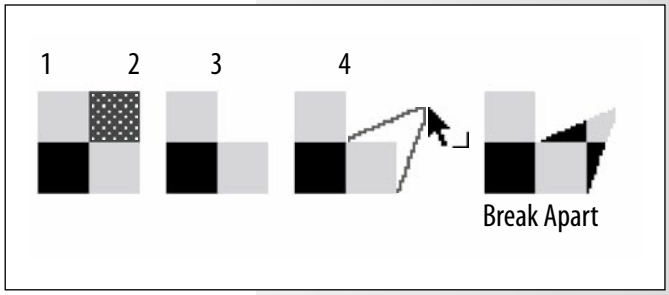


Figure 5-4. As seen in step 4 of this figure, when a bitmap is merely broken apart, a single newly created shape is filled with the tiling bitmap. When a bitmap is traced, it is replaced with multiple newly created shapes that contain normal vector fills.

Using Bitmap Fills

Once you break apart a bitmap, it is possible to eyedrop its image into a fill pattern. That pattern can then be used as a fill for a larger shape, creating a seamless tile, as seen in Figure 5-5.

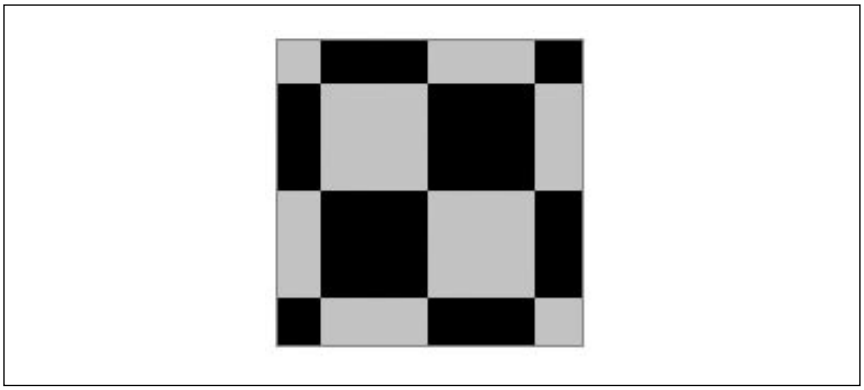


Figure 5-5. When a larger shape is filled with a bitmap, the bitmap tiles seamlessly

Once you have a bitmap fill, you can do some creative things with it. For example, you can cover large backgrounds with a sacrifice in file size that pales in comparison to a full bitmap of the same dimensions. In the next project, you will try something a little less intuitive. You can apply tweens to the fill itself:

1. Create a new, blank document and save it as *bitmap_tile_tween.fla* in the */05/Working with Pixels* folder.
2. Import the *bitmap_tile.png* file found in the same folder.
3. Break apart the bitmap using Modify→Break Apart (Ctrl/Cmd-B).
4. Select the Eyedropper tool and click on the bitmap. Your fill chip will switch to the new pattern.
5. Delete the bitmap, select the Rectangle tool, and fill the Stage with a new shape. You will see that the fill is a repeating pattern created by the bitmap you broke apart.

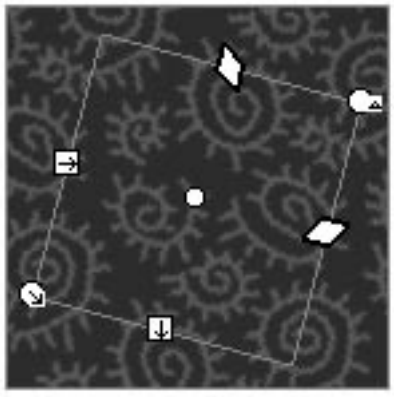


Figure 5-6. Using the Gradient Transform tool, you can scale, rotate, and skew tiled fills for interesting effects

6. In the timeline, select frame 24 and add a keyframe. This will create a 2-second animation using Flash's default 24 fps.
7. Still in frame 24, select the Gradient Transform tool and click on the shape. You will see the same tools you used to manipulate the color background in your Jimi Hendrix poster in Chapter 1. Grab the upper-right corner of this tool grid with your mouse, and drag down and to the right a bit. This will rotate the fill.
8. Grab what was the lower-left corner of the square, and scale the fill. Hold down the Shift key to scale the fill proportionally, if you prefer. Your fill should look somewhat similar to Figure 5-6.
9. Next, click in the keyframe in frame 1, and select Shape from the Tween menu in the Properties panel. (If your Properties panel is not visible, show it from the Window menu.)

10. Test your file. Your bitmap fill will scale and rotate to the extent that you changed it. If you want to check your work, open the *bitmap_fill_animation.fla* file in the */05/Working with Pixels* folder.

Tracing Bitmaps

In the previous project, you saw that a bitmap behaves as a fill when broken apart. However, you can also convert bitmaps into shapes that respond the way you've come to expect shapes to respond. This is accomplished by tracing the bitmap into a collection of vectors.

Using this technique, Flash will analyze a picture and attempt to outline discrete areas of color, based on tolerance settings you provide. The result can be clarified by comparing the process to breaking apart a bitmap, described earlier. Once traced, the first difference is that you can now select an area of vector with one click. After removing that area, pulling away the lower-right corner will effectively increase the size of the self-contained shape manipulated (Figure 5-7). This is just how the shapes you are familiar with would behave if a bitmap was never involved.

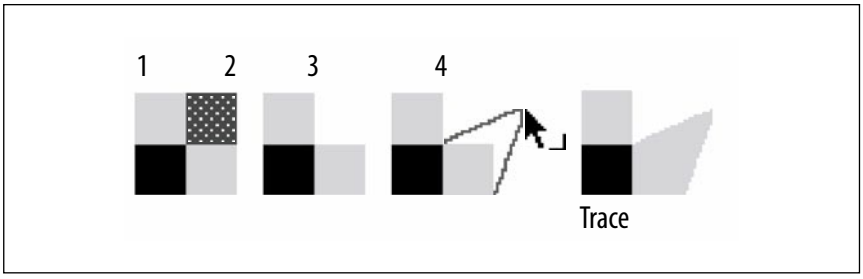


Figure 5-7. A traced bitmap behaves just like a collection of simple Flash shapes

For your next project, trace a bitmap to gain some experience with the dialog settings:

1. Create a new, blank document and save it as *bitmap_trace.fla* in the */05/Importing Pixels* folder.
2. Import the *cheesecake.jpg* file found in the */05/Importing Pixels* folder.
3. Select the bitmap, but do not break it apart. Instead, choose the Modify→Bitmap→Trace Bitmap menu command. The dialog seen in Figure 5-8 will appear. The Color Threshold setting allows you to adjust how clearly Flash separates similar colors. This is similar to Photoshop's Tolerance setting. A higher number means that more hues will be included in any given selection. The Minimum Area setting dictates the minimum number of adjacent pixels that must be included in any single shape. This allows you to set the granularity of the vectors, preventing too many from being created. (This is somewhat akin to using fewer polygons to make a 3D model more efficient.) The Curve Fit

and Corner Threshold settings both contribute to how curvy, or blocky, each vector shape is.

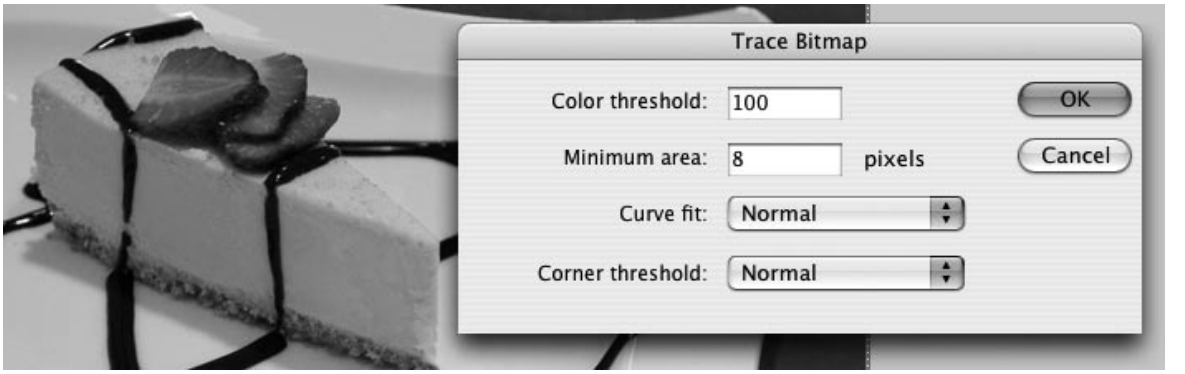


Figure 5-8. Adjust the settings in the Trace Bitmap dialog box

4. Start with the default values and see what you get. Next, try higher and lower color thresholds, and experiment with the other settings. Figure 5-9 shows three such settings groups and their results. When you're done experimenting, save your work.

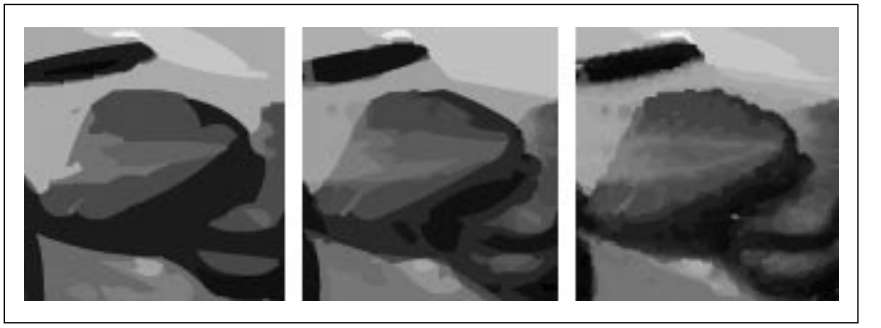


Figure 5-9. These three tracings show the successive result of decreasing Color Threshold, Minimum Area, and Curve Fit/Corner Threshold settings. For example, the Color Threshold settings were 100, 50, and 25, from left to right.

NOTE

If your results are not similar, test your movie. If you then see noticeably different results, you may have your Preview Mode set to a speedier, but rougher, setting. Try changing your View→Preview Mode setting to a higher value. If your Flash authoring doesn't slow down significantly, this will enable you to preview your files more accurately as you work.

Comparing your file size before and after the tracing usually results in a file size drop, although that's not always true. Sometimes trace tolerances can

be set too strictly, and a highly compressed image can be converted into many small vectors, which actually increases file size. One way of getting the most file size economy following a bitmap trace is to optimize your curves:

1. With the bitmap tracing still visible, select all the vectors in the tracing using Edit→Select All.
2. Choose the Modify→Shape→ Optimize menu command. This will bring up the Optimize Curves dialog shown in Figure 5-10.
3. Experiment with the Smoothing setting, being sure to enable the "Use multiple passes" and "Show totals message" options. The former, although slower, will be more accurate and will usually result in a slightly better looking optimization. The latter will show you the amount of reduction in total curves accomplished by the process.

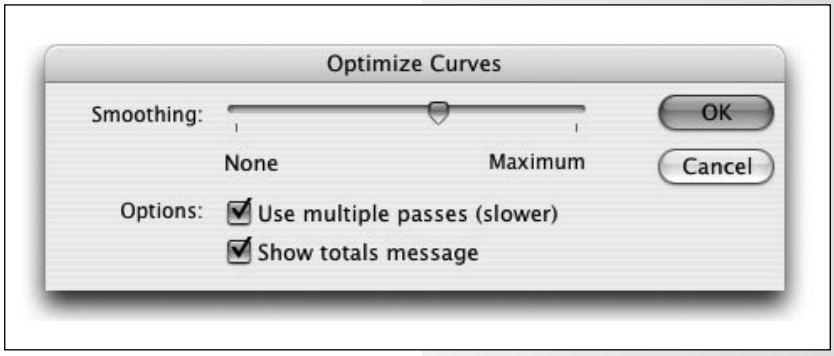


Figure 5-10. Bitmap tracings, as well as any other vectors, can be made more efficient by removing unneeded curves

As a comparative example, *bitmap_trace_02.fla*, found in the */05/Working with Pixels* folder, is an interactive look at different trace settings, complete with possible shape optimization data. Open and test the movie to see the difference a setting can make.

Methods

When reading a text that includes discussions about ActionScript, you will frequently run across the term *method*. A method is an action that is taken by an object, akin to the way a verb and a noun work together. For example, just as a DVD can play, so can a movie clip.

You have already seen several uses of methods. For example, in Chapter 3, you used the *stop()* method in the main timeline of your animation project, and in Chapter 4, you used methods such as *gotoAndPlay()* and *getURL()* for part 2 of the same project.

There are two common structures for methods. The first is when a method affects any timeline (or other object) in which the method call itself resides. In this usage, the method is sometimes included without additional syntax. For example, in Chapter 4, your replay button used the *gotoAndPlay()* method to control the main timeline, which is where the *replay_btn* was placed:

```
replay_btn.onRelease = function() {
    gotoAndPlay(1);
};
```

Another example usage is when you want a method to affect a specifically referenced object, such as a different movie clip. For example, the following line instructs a movie clip called *legs_mc* to stop:

```
stop_btn.onRelease = function() {
    legs_mc.stop();
};
```

Here you see that the method is attached to the specific object in question with the dot syntax you've used in previous projects. You will use this approach in the next chapter, as you expand the interactivity used with movie clips. In both cases, it is helpful to recall the aforementioned mnemonic device: the relationship between object and method is very similar to the relationship between noun and verb.

Importing Vectors

As is true with raster graphics, Flash can import vector assets from a variety of external applications, and in a variety of formats. Here, we'll look at how to import Adobe Illustrator, Macromedia FreeHand, and Adobe PDF assets, and even other SWF files.

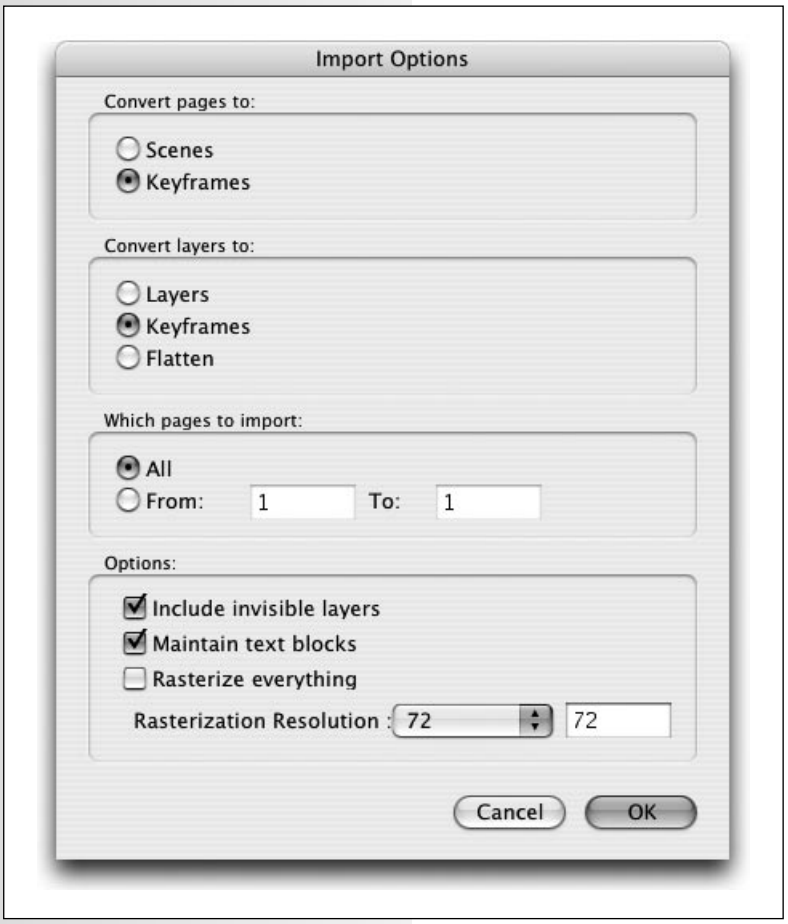


Figure 5-11. Standard vector asset import settings

Importing from Illustrator or FreeHand

Both Illustrator and FreeHand support enhanced importing options, as seen in Figure 5-11. Both will allow you to convert drawing layers to Flash layers or keyframes, as well as flatten the drawing into one layer and keyframe of vectors. They also both allow you to optionally import hidden layers and maintain text blocks. Finally, Illustrator allows you to rasterize the entire drawing into a bitmap of almost any resolution. (You will learn more about pages in the section "Importing from PDF.")

If you have Illustrator, you can open the *tabbed.ai* file (found in the */05/Importing Vectors* folder) in Illustrator and the companion *ai_keyframes_01.fla* file in Flash. You will see the beginnings of an interface, as layers are converted to keyframes upon import. In the next step of the process, shown in *ai_keyframes_02.fla*, simple scripts have been added directly to the buttons to mock up the tabs in action. Although a very simple example, this is an excellent way to get an interface working quickly in Flash if you are comfortable with a particular drawing program.

Special features in drawing programs can also be used to great advantage. For example, FreeHand's Xtra, Animate→Release to Layers, can be used to turn shape tweens (called *blends* in the FreeHand vernacular) and groups into multi-layered files. Using the "Convert Layers to Keyframes" option in the vector import dialog discussed previously, you can then create an instant animation when importing the file into Flash.

If you have a recent copy of FreeHand, begin by opening *flower_01.fh11*, found in the */05/Importing Vectors* folder. Using the aforementioned Xtra, this single-layer file is turned into *flower_02.fh11*. Each element is repeated on a new layer, adding a new element each time. The result is an animation build that can be immediately translated into Flash.

If you don't have FreeHand, or you want to move on, import the result that has already been prepared for you:

1. Create a new, blank document and save it as *fh_flower.fla* in the */05/Importing Vectors* folder.
2. Import the *flower_02.fh11* file found in the same folder.
3. Match your settings to those found in Figure 5-11.
4. Once the import is finished, save your work and test your movie. The flower animation will build over time.

When you are finished viewing the animation, close the *.swf* and return to your *.fla*. Open the Library and notice that six new symbols have been added to the file. These were originally FreeHand symbols, but they have been conveniently converted to Flash symbols for you. With a little advanced planning, FreeHand economies can make your Flash files more efficient, too.

Functions

As discussed briefly in Chapter 4, a *function* is a set of actions that Flash can perform only when called upon to do so. This is useful for preventing the actions from executing automatically, as would be the case in a simple frame script.

If you've been following along with the projects in this book, you've already been exposed to functions a few times. Although not too much attention was focused on functions themselves, you've seen them used with button and movie clip event handlers, like so:

```
start_btn.onRelease = function() {
    myClip1.play();
};
```

This type of function is called an *anonymous function* because it has no name. Instead, it is directly associated with an event handler, and the function is called by the event in question. In the above case, the function is called when the mouse is clicked on the button named *start_btn* and then released.

Custom functions can also be named and called any time, not just when associated with a specific event handler. For example, the following function will set the location of two movie clips and play them both:

```
function playAllClips() {
    myClip1._x = 20;
    myClip1._y = 20;
    myClip1_mc.play();
    myClip1._x = 60;
    myClip1._y = 20;
    myClip3_mc.play();
}
```

Once defined, you can then call this function by name from another script:

```
playallClips();
```

A few simple rules must be followed when working with functions, but they're easy to understand and remember. You'll learn more about their use when you work with functions in several other chapters, and during the discussion of scope in Chapter 7. You won't need to grapple with all of this now—for now, it's helpful merely to understand the purpose and structure of a function so you'll know what it is when you see it.

Importing a SWF

In most cases, you can even import a *.swf* into your *.fla* file. (In a later chapter, you will learn how to protect your *.swf* files from importing, but for now, you will work with the default setting where protection is not enabled.)

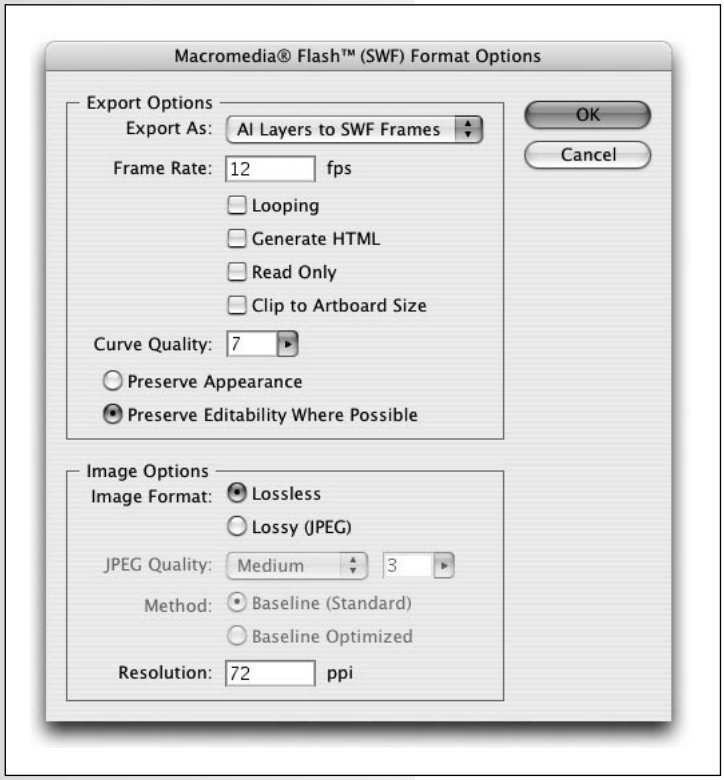


Figure 5-12. The export features in FreeHand and Illustrator (shown here) allow you to create SWF files directly from within these programs

When importing SWF files, you will not be able to import much more than graphics. However, this is still useful if you want to bring a SWF created by another program into a Flash file that you're working on.

Many third-party programs, including FreeHand and Illustrator, can create *.swf* files. These export options can even be used to create simple animations without ever opening Flash. Illustrator, in particular, can often provide better results in color and shape translations (especially from a CMYK original) by exporting a *.swf* that you then import into Flash, rather than importing the Illustrator or *.swf* document directly.

Exporting a *.swf* from a third-party tool is usually straightforward. Figure 5-12 shows the Illustrator options, as an example. Note that you can convert layers to keyframes (or files, or create one composite file). You can also apply a small selection of settings for direct export to animation, preserve appearance or editability, and set bitmap compression settings for converting embedded bitmaps.

Importing from PDF

Flash 8 can import PDF documents, using the same vector import dialog discussed previously. Depending on how complex the PDF is, Flash usually does a pretty good job of reproducing the original content.

In basic terms, importing a PDF is no different from importing an Illustrator or FreeHand file. However, the following project will give you a chance to experiment with Flash's Convert Pages to Scenes option. Although pages are supported in other applications, the idea of pages in a PDF is very intuitive and common, so you'll run with that idea now:

1. Create a new, blank document and save it as *pdf_scenes.fla* in the */05/Importing Vectors* folder.
2. Change the dimensions to 640 × 480, using the Modify→Document menu command.

3. Double-click the *editorial.pdf* file to open it, and click through its two pages. It is a simple two-page mock magazine article about a car, with an image and a pull-quote.
4. Import the *.pdf* into Flash. When the import dialog appears, match your settings to those seen in Figure 5-11, *except* for the first option. For this project, change the "Convert Pages to" option to Scenes instead of Keyframes.
5. Save your file and look at the result of the import.

You will probably notice right away that only one page of the PDF is visible. This is because the two pages have been separated into two discrete segments, called *scenes*.

Using Scenes

Scenes provide a way to organize large files or animations into multiple timelines. This makes it easier to view smaller sections at a time, without having to scroll endlessly. For example, you may have a large linear animation that is organized into chapters. Such a file may be easier to work with if you separate each chapter into its own scene. Similarly, you may have a story-driven game with levels that are separated by animations that further the story. In this case, each level and animation might be placed into its own scene.

To add Scenes, use the Insert→Scene menu command. Scenes can be given descriptive names to make it easier for you to distinguish one from another. To switch between scenes in authoring mode, use the Edit Scene menu in the upper-right corner of the main document window, as seen in Figure 5-13.

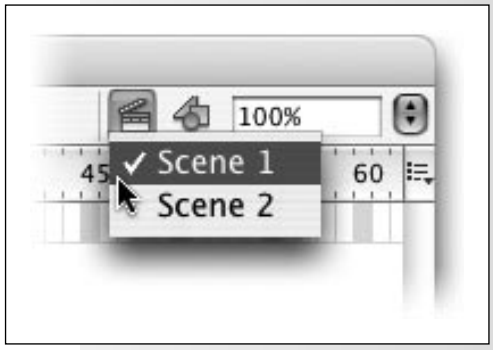


Figure 5-13. Use the Edit Scene menu to switch scenes and timelines.

Alternately, you may use the Scene panel (Window→Other Panels→Scene) to accomplish all of these tasks. It lists all current scenes, allowing you to click once to switch scenes, or double-click to rename a scene. In addition to adding scenes in this window, you can also delete and even duplicate scenes—handy when the bulk of a timeline remains intact and only subtle changes are required in the new scene.

Scripting Scene Changes

At runtime, a movie with multiple scenes behaves just like a movie with one long timeline. For example, if you test the movie you just created, you will see the two pages from the PDF flashing back and forth, just as if they were side by side in keyframes.

Adding scene navigation is as simple as adding frame navigation:

1. Continue with the file you created earlier, when importing the *editorial.pdf* file.

2. Create a button in the lower-right corner of each scene that you can use to switch between them. If you prefer, open the provided *pdf_scenes_01.fla* in the */05/Importing Vectors* folder, which already has buttons included.
3. In Scene 1, give the button an instance name of **`next_btn`** using the Properties panel. In Scene 2, name the button **`prev_btn`**.
4. Remember that the playhead will automatically move between scenes by default, so you need a script to prevent this from happening. In the first and only frame of each scene, add a *stop()* action.
5. Now you must script the button. In the first frame of Scene 1, add an *onRelease* event handler to the `next_btn` instance with a *nextScene* action. Your frame script should now look like this:

```
stop();
next_btn.onRelease = function() {
    nextScene();
};
```

6. Follow a similar course in the first frame of Scene 2, this time adding the *prevScene* action to the `prev_btn` instance:

```
stop();
prev_btn.onRelease = function() {
    prevScene();
};
```

7. Save your work and test your movie. You will see that the movie does not automatically switch scenes, and now does so only when you click on the buttons you created. If you want to check your work, *pdf_scenes_02.fla*, found in the same folder, is the complete file.

Working with the Library

Importing many graphics, and perhaps creating symbols along the way, can leave your Library a bit of a mess. To begin with, since bitmaps can be large, Flash automatically places a copy of each one in your Library. You will soon find out that the same is true for sounds and videos. Much like with symbols, you can drag these elements to the Stage, making multiple instances without noticeably increasing file size.

Furthermore, importing some types of assets will result in your Library filling up with additional objects. As you read earlier, FreeHand and Fireworks both translate their symbols into Flash symbols whenever possible. Sometimes, as you saw when working with the Fireworks file, additional folders are created in the process. Keeping the Library tidy is important if you want to keep your files manageable.

Organizing Your Library

There are many ways to organize a Library. Some developers like to use folders for each type of asset, while others organize assets according to their function. Still others prefer to organize Libraries based on the order of use. For example, in a Flash file for a cartoon, the Library might be organized so that each character's assets are in one place, and are further divided into subfolders to keep animated and inanimate symbols separate. A movie clip for a walking sequence could be in a folder with blinking eyes, for instance, while the hair and nose might be in another folder.

To create a folder in a Library, use the New Folder button in the lower-left corner of the Library panel, which looks like a folder with a plus sign (+). You can also create a new symbol using the New Symbol button to the left, and delete an element using the Delete button, which looks like a trash can.

You can add as many folders to the Library as you want, and inside each folder, you can nest other folders. How you organize the Library does not affect the movie at runtime at all. The important thing is just to stay organized, so assets are always easy to find. Large projects can contain hundreds of assets, and keeping them organized is key to maintaining a good workflow—including working with multiple files.

NOTE

Be aware that deleting a folder also deletes all of its contents. If you want to keep certain assets from a folder you plan to delete or want to reorganize your Library, simply drag assets from one folder to another. Although deletions can be undone, it is a good idea to make a backup copy of your .fla before doing anything major.

Sharing Libraries During Authoring

Your organization can have far-reaching effects, because Flash 8's new Library enhancements make it easier than ever to share assets between multiple files. As discussed briefly in Chapter 1, you can now show the Libraries for all open documents in one Library panel, using a menu to choose the current Library (see Figure 5-14). This simplifies dragging an asset you want to reuse from one document to another, or from a common Library to the document of your choice. If you prefer, you can spawn a new Library panel to drag symbols between them.

Even with these features, working with multiple documents can still get confusing. If desired, you can "pin" a Library to a specific document. This way, you can still select any open Library, but each time you switch between documents, any Library pinned to the active document will be displayed.

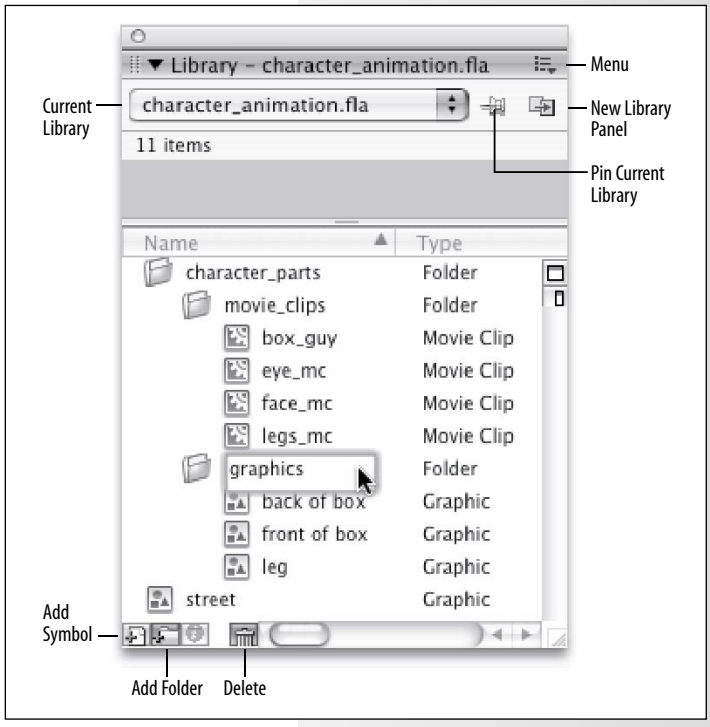


Figure 5-14. An organized Library, with key features highlighted

The Library is also a central location for modifying many properties for symbols and imported assets. A context-sensitive Properties button will show the properties of most selected Library items. It looks like an *i* and can be found between the New Folder and Delete buttons in the Library panel's lower-left corner. (The menu in the upper-right corner of the Library panel is also handy for exposing these properties.) You'll look at several properties in upcoming chapters, but for now you'll concentrate on two that are specific to images.

Compression Settings

Flash allows you to compress your graphics in a few different ways. With a JPEG, for example, you can retain the compression setting applied when the JPEG was created, or you can override its compression setting and apply a new one on a case-by-case basis. Figure 5-15 shows the Bitmap Properties about a JPEG, accessible from the Library's Properties button or menu after selecting the JPEG you want to examine.

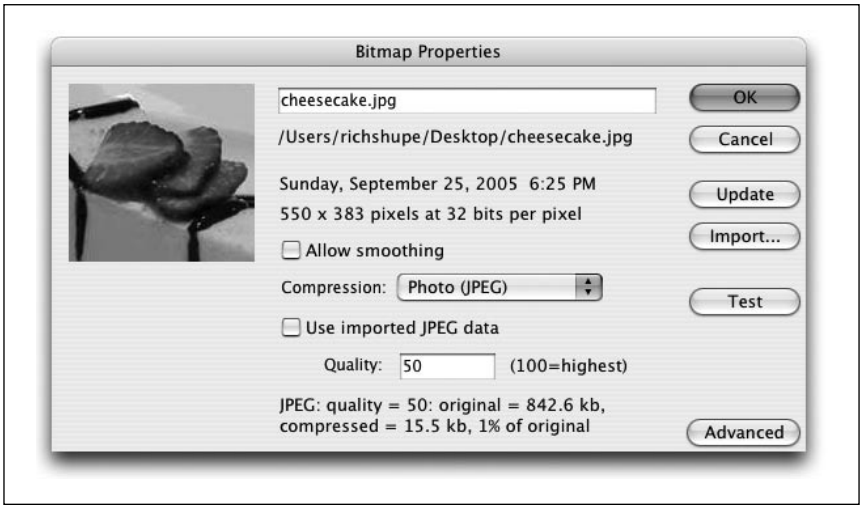


Figure 5-15. A test of the cheesecake.jpg compression settings reveals that, at a quality setting of 50, the compressed image is expected to occupy 15.5 KB of file size

In addition to information such as the import time and the location of the source file, there are a few settings here that you can adjust to optimize the bitmap. You can smooth its edges with anti-aliasing, choose between compression methods of Photo (best for continuous tones) and Lossless (best for larger areas of solids), and set the degree of compression used. In this case, the compression used when the graphic was created is being changed to 50% compression. (Higher values yield better looking, but larger, graphics.) Clicking the Test button will show you the expected size of the graphic at runtime.

Roundtrip Editing

Right/Ctrl-clicking on a bitmap in the Library is another way to quickly access additional features, including *roundtrip editing*. This convenient feature enables the editing of a selected asset in an appropriate external editor, and then automatically updates the embedded asset with the changes. For example, Figure 5-16 shows the launching of Fireworks to edit a *.png* document. This can dramatically speed workflow, because it means that you don't have to reimport assets that require editing.

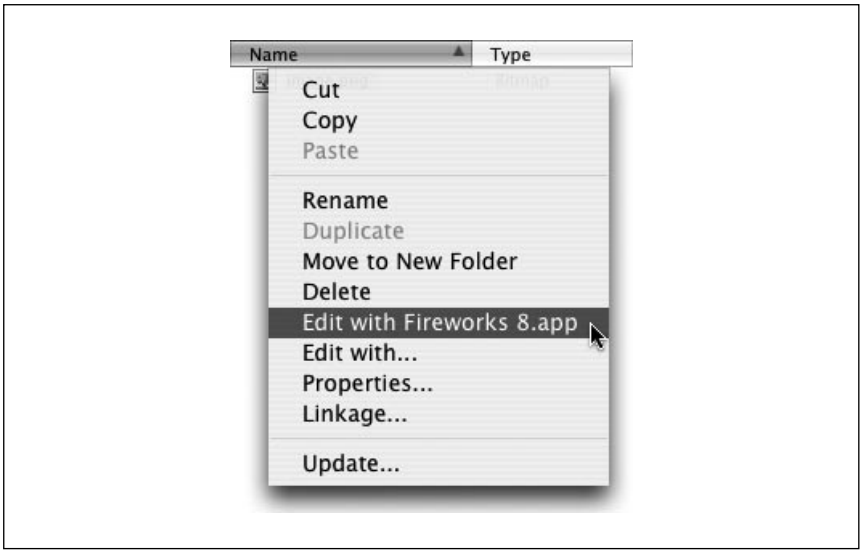


Figure 5-16. Roundtrip editing allows you to edit a Flash asset in a specified application, make changes, and then automatically update the Flash file upon closing the document in the external editor

These Library features, and your growing familiarity with the Libraries, will become very important when you start filling them with assets. In Chapters 8 and 9, you will begin using sound and video, and in the next chapter, you will work with additional symbols, including movie clips.

What's Next?

As you can see, Flash is capable of importing several different kinds of graphic file formats and manipulating those assets in many ways. Now that you know what you can work with, the creative part is up to you. In Chapter 12, you'll learn how to load external graphics files on the fly, but for now try working with the skills you've already acquired.

Toy with bitmaps in clever ways to try to create interesting effects. For example, break one apart and recolor portions of the image for a primitive silk-screened look. Create different kinds of bitmap tile tweens for large but

file-size-efficient background effects. Trace different kinds of images with different Color Threshold and Curve Fit settings to achieve a fun, posterized effect. If you end up with something you like, put it aside until Chapter 7, when you can begin morphing the image into new, unimagined artworks.

In the next chapter, you'll work more closely with movie clips and add more interactivity to your files. You'll learn:

- How to create movie clip animations
- How to create, and improve, character walk cycles
- More about symbol instance names
- How to use ActionScript to control one movie clip independently of another
- How to target a symbol using absolute or relative pathnames
- How to make your movie clips respond to user mouse clicks