1

Getting Started with Macromedia Flash

Flash is a term used to refer to both the vector-based animation format (Flash movies) as well as the authoring tool that is commonly used to create the aforementioned content. Flash Player is the program that is capable of playing back Flash content. ActionScript is the scripting language that is encoded into Flash content and interpreted by Flash Player when that content is played back. You can use the Flash authoring tool to add ActionScript code to Flash content. As such, it's particularly useful to learn how to use the Flash authoring tool for that purpose as a first step when learning about ActionScript. In this chapter you learn about how to work with the Flash authoring tool for the purposes of writing ActionScript.

Introducing the Development Environment

Flash utilizes a panel system to organize specific functionalities available within the program. The panels can be docked and undocked as well as moved outside of the application area. This enables you to efficiently use and customize your working environment. You can find panels and layout options in the Window menu in the main navigation. Many of the descriptions in this chapter assume that you are working with the default panel layout. For example, when a panel is described as appearing in the upper left of the workspace, it means that the panel appears in that location in the default layout. You can open the default layout by selecting Window➪Workspace Layout➪Default.

By default, Flash 8 opens to a start page (see Figure 1-1) that enables you to launch projects quickly. You can specify whether Flash should display the start page. If you click the Don't Show Again checkbox at the bottom of the start page, it will not display by default. You can always edit the start page preferences from the Preferences dialog (Edit➪Preferences) by selecting the appropriate option from the On launch menu.

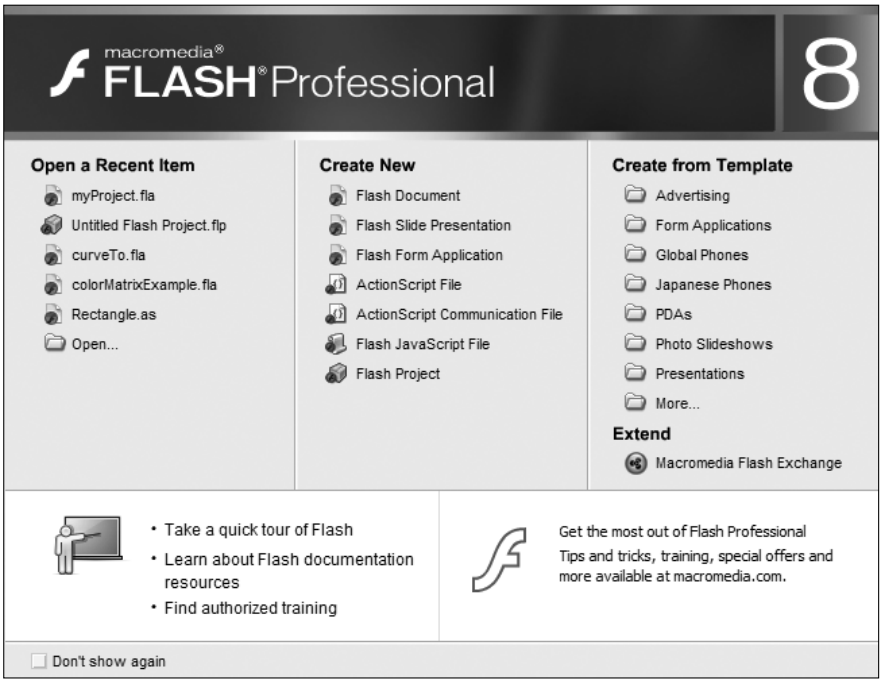

Figure 1-1

The following sections briefly explore other features of the Flash authoring program, which is frequently called the Flash IDE (Integrated Development Environment).

Tools Panel

On the left of the IDE is the Tools panel (see Figure 1-2), which contains the drawing tools that enable you to add vector content and edit existing vector content. This is where you select drawing tools, set tool attributes, and specify drawing tool options. The drawing tool attributes are expanded in the Properties panel, which is often called the Property inspector. This panel can be found at the bottom of the IDE. If you cannot see it, select Window➪Properties➪Properties.

Properties Panel

The Properties panel (also shown in Figure 1-2) presents the options for the item in focus on the stage or the selected tool. It is clearly contextual in that it changes depending on what object is selected (or if no objects are selected) or what tool is selected. For example, if you have nothing selected on the stage and choose the Selection tool, the Properties panel enables you to adjust document-wide settings such as stage dimensions and frame rate. However, if you select the Line tool, you'll see that the Properties panel enables you to adjust the settings for the Line tool such as line thickness and style. If you use the Selection tool to select a line on the stage, you are presented with the same set of options as when using the Line tool. But when you change the properties for a selected line, the line immediately reflects those changes.

Timeline

The Flash Timeline (see Figure 1-2) appears below the main navigation. The timeline enables you to add content that changes over time in a visual manner. The timeline is a clear indicator of Flash's origins as an animation tool.

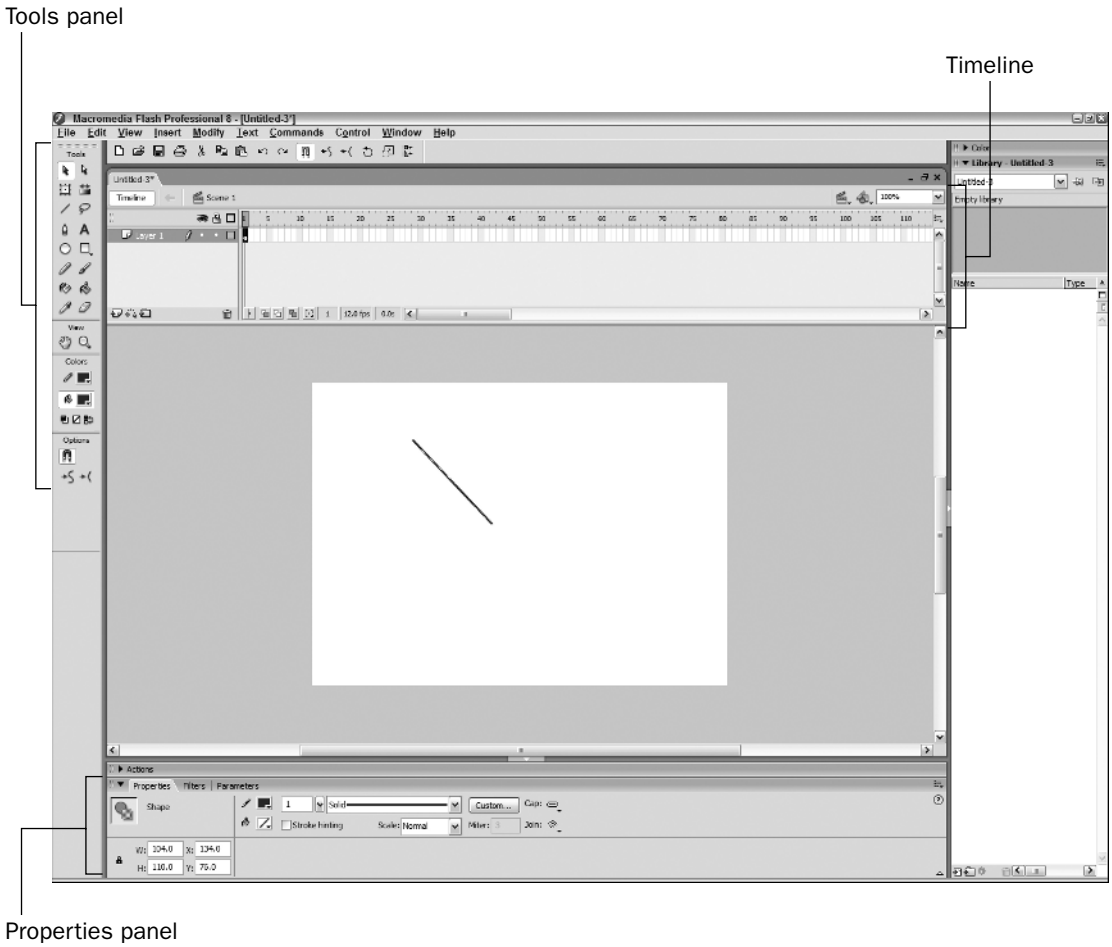

Figure 1-2

Keyframes and Animation

Timelines enable you to add frames to animate content over time. By default a timeline has just one frame at frame 1. You can add frames by selecting Insert➪Timeline➪Frame or by pressing F5. Where Flash adds the new frame or frames depends on what frame is currently selected on the timeline. If you select an existing frame in the timeline, Flash inserts the new frame immediately following the selected frame, and it moves subsequent frames to the right by one. If you select a frame placeholder after any

existing frames, Flash adds the new frame at that location. If necessary, Flash also adds frames for all the frames between the new frame and previously existing frames. For example, if the timeline has one frame and you select the frame placeholder at frame 5 and press F5, Flash adds four new frames — one at frame 5 and three in the frames between the existing frame and the new frame.

Flash also has keyframes, which are special frames to which you can add content. Regular frames simply add time between keyframes. Keyframes are the only frames to which you can add artwork, text, and so on. The default frame in a timeline is always a keyframe. You can add new keyframes by selecting Insert➪Timeline➪Keyframe or by pressing F6. When you select an existing keyframe and add a new keyframe, the new one is added immediately following the selected frame. If the following frame is an existing regular frame, it is converted to a keyframe. If the following frame is a keyframe, no change is made. If there is no frame following the selected frame, a new keyframe is appended. If you add a keyframe by selecting a placeholder where no frame yet exists, Flash adds the keyframe and regular frames to all frames between the last existing frame and the new keyframe. If you select an existing frame and add a keyframe, Flash converts the existing frame to a keyframe.

Keyframes are denoted by circles on the timeline. When a keyframe has no content, the circle is an outline. When content is added to the keyframe the circle appears as a solid black dot. Regular frames following a keyframe appear white when the keyframe has no content. When the keyframe has content, the subsequent regular frames are gray.

At the document level, there is always at least one timeline. This default, top-level timeline is often referred to as the main timeline or the root timeline. In ActionScript, the main timeline has two names: `_root` and `_level0`. Many types of library symbols also have timelines. For example, movie clip symbols have timelines. A common error for novice and expert Flash developers alike is to edit the wrong timeline by mistake. You can determine what timeline you are editing by looking to the right of the Timeline toggle button. Flash displays the name of the timeline you are currently editing. By default, the name Flash displays for the main timeline is Scene 1. If you edit a symbol, the name of the symbol appears to the right of Scene 1. You are always editing the timeline that appears the farthest to the right of Scene 1, and you can always return to the main timeline by clicking Scene 1.

About the Library and Symbols

Flash content is composed of elements such as text, vector artwork, bitmaps, sounds, and video. To most effectively manage content, it is frequently placed within symbols. Flash has many symbol types, including Button, Graphic, Movie Clip, Bitmap, Font, Sound, and Video. Of the types, three (Button, Graphic, and Movie Clip) allow you to group together content from the stage, and place it into a reusable symbol. The symbols are then stored in the Library. The Library appears on the right side of the IDE in the default layout. You can toggle the visibility of the panel by selecting Window➪Library.

You can make a new Button, Graphic, or Movie Clip symbol in one of several ways. You can draw content on the stage, select it, and convert it to a symbol by selecting Modify➪Convert to Symbol or by pressing F8. Optionally, you can select Insert➪New Symbol, press Ctrl+F8 on Windows or Cmd+F8 on a Mac, or select New Symbol from the Library panel menu to add a new symbol that doesn't initially contain any content. In either case — whether converting to a symbol or adding a new empty symbol — you're presented with a dialog that prompts you for a symbol name and type.

The name of a symbol is arbitrary — there are no naming rules you must follow. The name you assign a symbol is the name that appears next to the symbol in the Library. It's advisable to select a name that accurately reflects the content of the symbol. For example, if you create a symbol containing a drawing of a cloud, it would be appropriate to name the symbol `cloud`.

The type you select for a symbol is important. The three options — Movie Clip, Button, and Graphic — behave very differently, and each ought to be used appropriately. Graphic symbols are useful for grouping artwork and/or animation sequences. However, you cannot control Graphic symbols or instances of those symbols with ActionScript, so they are not discussed in this book. Button symbols are symbols with very specialized timelines. Button symbol timelines have four keyframes labeled Up, Over, Down, and Hit. When you place an instance of a Button symbol on the stage and export the movie, Flash automatically jumps to the keyframes that correspond to the mouse state. For example, when the user moves the mouse over a Button instance, Flash jumps to the Over frame. You can read more about Button symbols in the following section, "Working with Buttons." Button symbols are useful for making clickable objects. Movie Clip symbols, however, are the powerhouse of symbols in the context of ActionScript. Using ActionScript, you can add instances of Movie Clip symbols and you can control those instances. You can read more about Movie Clip symbols in the "Working with Movie Clips" section a little later in this chapter.

Once you've added a Graphic, Button, or Movie Clip symbol to the Library, you have a repository of reusable templates. You can think of symbols as blueprints. You can then add one or more instances of a symbol to the stage. Generally, you add instances by dragging the symbol from the library to the stage. Each instance is based on the same template — the symbol. However, each instance is unique and independent of any other instances that may happen to be based on the same symbol. Deleting a symbol from the Library also deletes all the instances.

Working with Buttons

Button symbols have four states available: Up, Over, Down, and Hit. The Button symbol timeline is instantly recognizable because it has four frames labeled to correspond to the states. You can add Movie Clip instances, Graphic instances, ActionScript code, sounds, and so on to the frames just as you would any other timeline.

By default Button timelines have just one frame defined — the Up state. If you add content to the Up state, add an instance of the Button symbol to the stage, and test the movie, you'll notice that the mouse cursor changes when you move the mouse over the instance. If you define a keyframe on the Over frame, add content to the frame that differentiates it from the Up state, test the movie, and move the mouse over the instance, you'll notice that this time the content of the Button instance automatically changes to the Over state. As you move the mouse off of the instance, the content changes back to the Up state. Likewise, if you add a keyframe and unique content to the Down state, Flash automatically changes to that frame when the user clicks the button. The Hit state is never visible when testing the movie; it defines the region that activates the Over and Down states. For example, if the content for the Up state consists of text, the user will have to move the mouse directly over the font outline to activate the Over or Down states. If the mouse is not directly over the font outline, it will not activate. However, if you define a Hit state, moving the mouse anywhere over the Hit state will activate the Button.

Try It Out Working with Button Symbols

In this exercise you create a simple interactive button. Here's what to do:

1. Create a new Macromedia Flash document by selecting File➪New and choosing Flash Document from the New Document panel. Save it as `symbolExample.fla`.
2. If the Library is not open, select Window➪Library from the main navigation.
3. Select Insert➪New Symbol.
4. The New Symbol dialog opens. Name the symbol `myButton` and be sure the Button option is selected. Don't worry about the other options. Click OK.
5. The button's stage and timeline appear in the IDE's main view area. Select the first frame, which is named Up.
6. In the Tools panel on the left side of the IDE, select the Rectangle tool.
7. The Tools panel contains two color swatches. The upper color is for lines, and the lower color is for fills. Click the upper swatch and select a dark gray or black for the line color. Click the Fill swatch and select a dark red.
8. Draw a rectangle on the stage by clicking the stage and dragging until the desired shape of a rectangle is achieved. Don't worry about centering the rectangle perfectly within the symbol. The rectangle should look something like the one shown in Figure 1-3.

Figure 1-3

9. Click the Selection tool (small black arrow) in the Tools panel. Use it to select the rectangle on the stage by double-clicking the rectangle's fill area.
10. The Properties panel at the bottom of the IDE reflects the properties of the selected rectangle. In it, change the height to `30` and the width to `120` and set the x and y values to `0`.
11. Select the Over, Down, and Hit frames in the timeline. To do this, click the empty frame named Hit, and drag the mouse to the left until all three frames are selected.

12. With the three frames selected, right-click (option+click) the Over frame. Select Convert to Keyframes from the context menu. The new keyframes contain copies of the content from the first keyframe. In this case, each keyframe now has a copy of the rectangle.
13. Select the Over frame. Select the Rectangle by clicking it with the Selection tool. Change the fill color to pink.
14. Select the Down frame, and then select the Rectangle by clicking it with the Selection tool and change the Fill color to white.
15. Create a new timeline layer by clicking the Insert Layer icon at the bottom left of the Timeline panel. (The icon looks like a sheet of paper with a plus sign (+) on it.)
16. Select the first frame of layer 2, and choose the Text tool (capital A) in the Tools panel.
17. Click anywhere on the stage to produce a text field. Enter the word `Play` in the field.
18. Choose the Selection tool from the Tools panel and click the text field you just created. (If you lose the location of the text field because it is the same color as the stage, click the first frame in layer 2 again.) The Properties panel populates with the attributes of the text field. Click the color swatch and change the color to a light gray.
19. Align the text to the center of the rectangle.
20. In the Timeline panel select the words `Scene 1`. Your button disappears from view and is in the Library. Open the Library, select the button, and drag it to the stage to create a new instance.
21. Test your application by selecting Control➪Test Movie.
22. Test your button by mousing over it and clicking it.

How It Works

In this exercise, you used basic vector graphics tools, the Library, and the icons and options within the Timeline to quickly produce an interactive button that reacts to mouse events. You also resized and realigned your button.

Later in this book you see how to import image files into your button symbols for each button state. You can use your favorite image editing program to create `.jpeg`, `.gif`, and `.png` files to give the button states a slick look.

Save your Flash document; you'll use it in the next Try It Out.

Working with Movie Clips

Movie Clip symbols have their own timelines that can play back independently of any other timeline in the Flash movie, and you can control instance of Movie Clip symbols programmatically with a sophisticated assortment of functionality. These things make Movie Clip symbols the most common symbol type for use with ActionScript. Movie Clip symbols are of such importance to Flash movies that they require an entire chapter to discuss. You can read more about Movie Clip symbols in Chapter 7.

About Layers, Depth, and Levels

Timelines enable you to add layers. If you place content on unique layers, you can then sort the layer order to affect which content appears in front and which appears behind. You can change the order of layers by dragging them up or down within the timeline.

Layers are an authoring time convention. Flash Player doesn't know anything about layers. Instead, when a Flash movie is exported, it converts all content on layers to depths. Conceptually, depths are similar to layers, but they are accessible programmatically. Flash allows only one object per depth. If you add content to a depth that already contains content, the existing content is deleted.

ActionScript enables you to add content programmatically. Because programmatic content is added at runtime, you must use depths and not layers. Methods such as `attachMovie()` and `createTextField()` that add content programmatically require that you specify depths. Depths are integer values. The higher the number, the further in front an object appears. For example, an object with a depth of 2 appears in front of an object with a depth of 1. Content added at authoring time (on layers on the timeline) are placed at depths starting at –13683. That means that authoring time content appears behind content added programmatically, assuming you use positive depths when you add that content.

Every movie clip encapsulates its own set of depths. For example, if you have two Movie Clip instances, A and B, and each contains nested Movie Clip instances, the depths of A and B affect how the contents of each movie clip appear relative to one another. If A has a higher depth, then all the contents of A appear in front of all the contents of B, even if the contents of B have higher depths than the contents of A. That also means that you can add content to A at depth 1 and content to B at depth 1 without one overwriting the other.

Flash also uses a concept called levels. By default there is just one level in a Flash movie. It has the name `_level0`. You can only add levels by loading content using MovieClipLoader or the older global function `loadMovieNum()`. In general it's not advantageous to work with levels; levels are primarily a legacy concept going back to a time when Flash didn't have programmatic depths.

Setting Up Your Scripting Environment

In this book you use the Flash IDE for writing the majority of the code examples. When you are writing ActionScript in Flash you'll likely appreciate the capability to customize how the code is formatted and how Flash assists you in adding code. In the next sections you read about the Actions panel and the options available for customizing it.

Introducing the Actions Panel

When you want to add ActionScript within a Flash document, you'll need to use the Actions panel. You can open the Actions panel by selecting Window➪Actions or by pressing F9. From the Actions panel you can add ActionScript to a keyframe or a symbol instance. Flash will add the script to a keyframe if you've selected the keyframe from the timeline and it will add the script to a symbol instance if you've selected the instance on the stage. Adding code to symbol instances is no longer advisable. Prior to Flash MX it was the only way to add certain types of code. However, adding code to symbol instances has disadvantages in that it generally requires much more code, it doesn't scale well, and it can be difficult to keep track of where you've placed code. For those reasons it's recommended that when you add code to a Flash document, you always add it to a keyframe. A common mistake made by novices and experts

alike is to accidentally add code to a symbol instance when intending to add it to a keyframe. Before you attempt to add any code to the Actions panel, always make sure it says Actions – Frame in the title bar of the panel.

On the left side of the panel is the Actions toolbox and Script navigator. Both are seldom necessary or useful for anyone wanting to learn ActionScript beyond a very basic level. On the right is the Script pane where you can directly add ActionScript code. The Script pane is essentially a text editor.

Exploring ActionScript Preferences

Next you see how to customize the Actions panel. You can open the Preferences dialog by selecting Edit➪Preferences (see Figure 1-4).

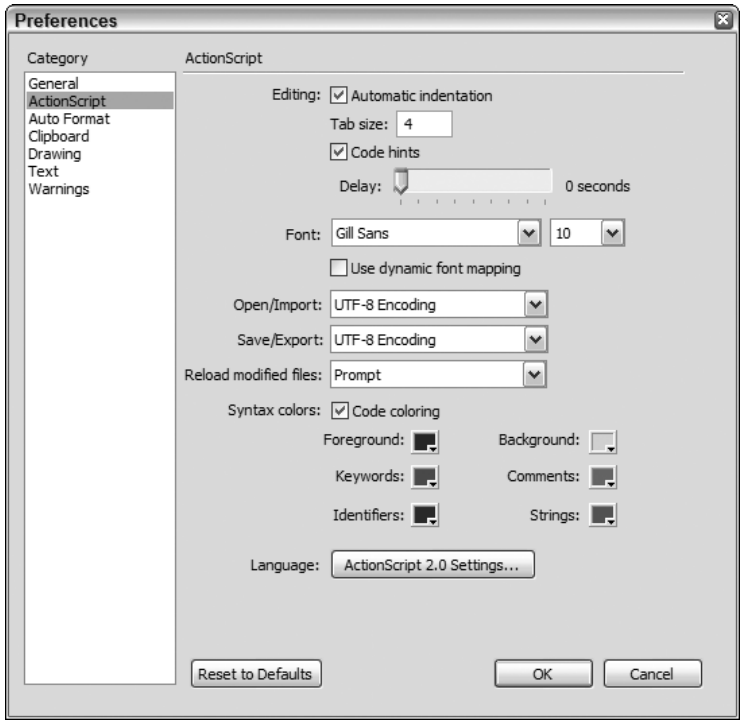

Figure 1-4

The following sections introduce you to some of the ActionScript preferences (select ActionScript from the Category list on the left side of the Preferences dialog).

Automatic Indentation

By convention code blocks are indented for readability. Code blocks are denoted by curly braces ({ and }). When you add a left curly brace ({) and a newline, Flash can automatically indent the next line. The Automatic indentation option is selected by default. If you don't want Flash to automatically indent your code, deselect the option.

Tab Size

The default tab size is the equivalent of four spaces. You can change the value if you want to indent more or less when pressing the Tab key.

Code Hints

Code hints are a way that Flash can help you by providing a list of likely options based on what code you've provided so far. For example, if you type the name of a variable that Flash recognizes as a movie clip, it can automatically display a list of functionality associated with movie clips.

Code hints can be triggered manually by using suffixes after an underscore in the variable name. For example, type the following code into the ActionScript panel:

```
example_array.
```

After you type the dot a drop-down menu of available array methods is presented. You can read a list of suffixes Flash uses in a document entitled "About Using Suffixes To Trigger Code Hints" in the Help panel.

Although suffixes work, they are no longer the preferred method of triggering code hints. With ActionScript 2.0, you can use strict typing to control code hints. Strict typing tells Flash what type of data a variable points to. That, in turn, allows Flash to display code hints regardless of whether you use suffixes in the variable name. For example, type the following code into the ActionScript panel:

```
var example:Array;
example.
```

The drop-down list of available methods for an array is presented as before. However, this time you aren't required to use a specific suffix as part of the variable name. You learn more about ActionScript syntax, variable declaration, and strict typing in the following chapters.

You can also specify how long a delay Flash should use before displaying code hints. The delay is between the trigger (usually a dot following a variable name) and the display of the code hints. A short delay could mean that code hints might appear even if you don't want them to appear. If the delay is too long, you'll have to wait for the code hints to appear. The default value is 0 seconds, which means the code hints appear immediately after the trigger.

Font

Although font is generally a matter of personal taste, some fonts are not advisable for coding. Georgia, for example, is a font in which the lowercase o is identical to its zero. This can make debugging and reading code extremely difficult. You want to avoid fonts that cause this type of problem.

Serif fonts are generally better for distinguishing letters from numbers and they better define individual words. Monospace fonts are generally better for code as well. For that reason the default ActionScript font that Flash uses is a monospace serif font such as Courier or Courier New.

You'll want to experiment. If you have colors and a font you like in another editor, carry them over. You'll be glad you did. If you don't yet have a preference, take the time to try some out. A scripting environment that suits you specifically is more welcoming, easier to work with, and more conducive to your way of working.

Color

One of the basic Actions panel enhancements you can make is color — color of the background, keywords, strings, comments, and so on. You can reduce eyestrain by changing the colors to those that suit you. For example, I generally don't like to code on a stark white background because after many hours of coding, I can no longer look at the screen without squinting. To alleviate this, I give the Background color a newsprint shade, somewhere near #E0DBC7.

Because there are generally few keywords in code, I set them apart from the rest of the code by assigning the Keywords color setting to #0066CC, which is a blue.

Identifier words are the names of classes and data types you use to declare variables. As such, they appear very frequently in code. I want to be able to quickly scan the code for identifiers, so I set the Identifiers color to a red, #990000.

Foreground color applies to the text, which is the meat and potatoes of the code, including functions that you create and methods that aren't tied to specific objects (`clear()` rather than `this.clear()`, for instance). The default color is black, but I prefer a lower contrast, so I generally choose #333333, which is on the gray side of black.

Comments are like road signs in code. They can often warn you of certain code peculiarities, as well as show you how the code is used or fits within the larger project. I usually set the Comments color to orange (#CC6600) so that I can easily see alerts and comments placed in the code.

Strings are a special case. It is important to be able to scan the code quickly and see the difference between `this.n.clear()` and `this.n = "clear";`. I use #009900, a primary green, for my Strings color; it's easy to discern among other colors.

Of course, you'll want to choose your own colors, and you can always return to the Preferences dialog and change them.

ActionScript 2.0 Settings

The Language section at the bottom of the ActionScript preferences dialog has a button that leads you to ActionScript 2.0 settings. A new dialog box opens (see Figure 1-5) that enables you to assign values for the classpath.

The classpath is a list of locations where Flash can locate code in class files when compiling. A class package can reside anywhere, as long as you have a classpath specified so that the compiler can find it. That means you can build libraries of code that you can use in many projects.

The list of classpaths should contain a default classpath, which targets the classes that ship with Flash. Do not remove the default classpath. You will also see a classpath that is simply a period (`.`). It specifies that Flash also looks in the same folder as the Flash document for any classes that are referenced by your code.

Flash searches for classes in the order in which you list the directories in the classpath. Flash uses the first instance of a class it encounters to compile. You can add a new directory to the classpath by clicking the plus sign (+), and you can remove an existing directory from the classpath by selecting it from the list and clicking the minus sign (–). You can reorder the directories with the arrow buttons.

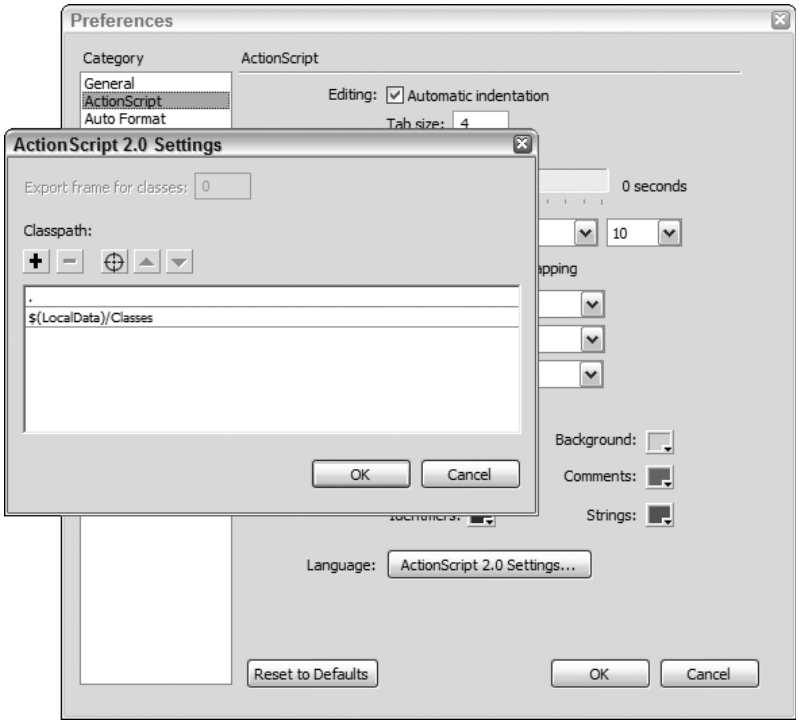

Figure 1-5

Exploring Auto Format Preferences

The Actions panel adds and indents characters when the Auto Format button is clicked. You can specify a few of those behaviors by first selecting Auto Format in the Preferences dialog box to display the appropriate page (see Figure 1-6).

The first two settings determine where Flash inserts the curly brace. If you check these options, the curly brace remains on the same line. If you leave these options unchecked, the curly brace is placed on the following line. If you've never used ActionScript before, you probably have no idea what this is about, but rest assured that you'll find out when you begin looking at functions and statements.

Just keep the Preferences dialog in mind for now because you'll probably want to come back to these settings once you start coding with ActionScript and become more familiar with how you prefer to work. Code examples in this book generally are formatted with none of the Auto Format options checked. The default settings are easy to read, though, and don't contain any surprises for fellow coders.

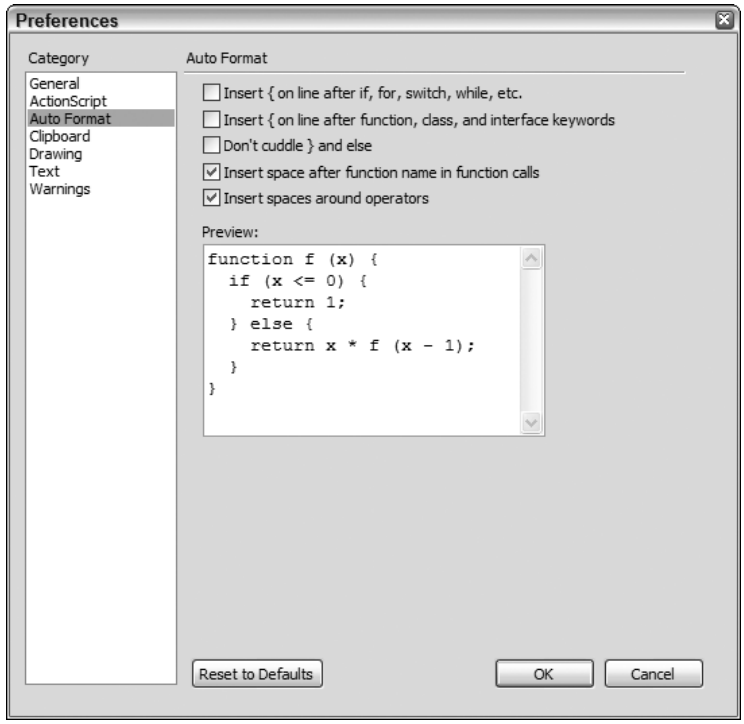

Figure 1-6

Publishing a Project

Once you've authored your Flash project you'll want to publish it so that you can deploy it. Flash has a Publish command that enables you to export to a variety of formats such as SWF, Quicktime, and even standalone executables. The Publish Settings dialog gives you the capability to select the formats to which you want to publish. It also enables you to customize the settings for each format. The Publish Settings dialog lets you automate the construction of JavaScript Flash Player detection as well as replacement images or HTML when a Flash player is not detected. The Publish Settings panel can help you target special Flash players such as Flash Lite and the Pocket PC player.

The Publish Settings dialog (see Figure 1-7) enables you to save and preserve profiles. This is handy for sharing the proper publish settings among developers or for storing a specific default profile you'd like to start with for all projects.

In the Publish Settings dialog you can select which formats you want to publish. By default the Flash and HTML formats are selected. When you select the Publish command (File➪Publish), Flash exports those formats that are checked in the Publish Settings dialog. You can use default paths to publish your assets. Alternatively, you can specify unique paths for each format. The Publish Settings are saved when you save your Flash document.

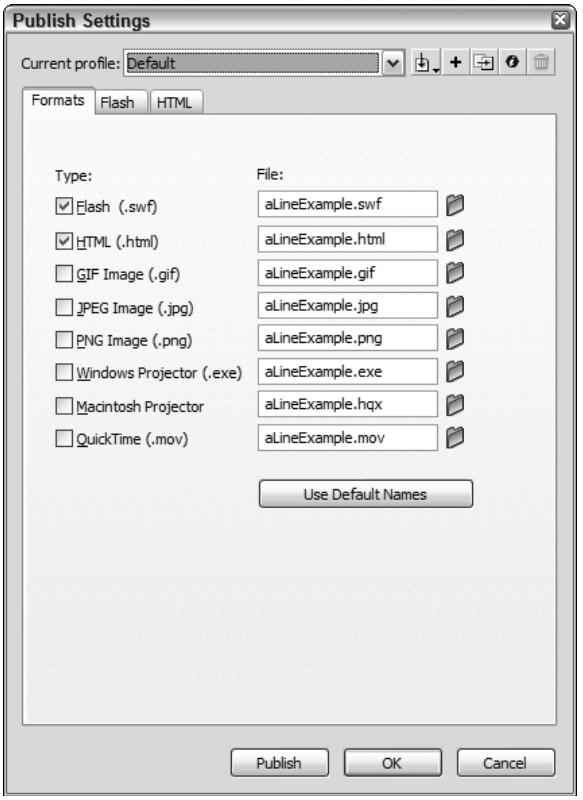

Figure 1-7

A new tab appears for each format you select (except the Projector formats, which don't have any special settings). By default, therefore, the Publish Settings dialog has three tabs — Formats, Flash, and HTML.

The Flash Tab

The Flash tab in the Publish Settings panel specifies the options for the SWF file. The version setting determines which players will be able to display your application. You should target a Flash version that best suits your audience and what you'd like your application to do. For example, if your Flash application takes advantage of Flash 8 text rendering because you require text to display at its best, you'll need to require Flash 8. However, if you're building a simple animation and interaction you may only need Flash 5. When you publish the FLA to SWF, the IDE will warn you if the code within the Flash document will not work in the targeted player specified in the Publish Settings panel.

Load order specifies how the first frame of the application should initially render. On slower machines this draw is more visible, so you should consider how you would like it to look. The default top to bottom, however, is sufficient.

The ActionScript version should match the ActionScript type you plan to use in your application. Although you can use ActionScript 2 output and use no version 2 code constructs, you should change this setting to at the very least alert future developers who might open your Flash document about the syntax used. The setting affects the compiler and compiler directives as well as error checking and strict type compliance.

The next group of options enables you to set attributes in the resulting SWF application. If you select Protect From Import as an attribute to include in your SWF, you are given the option of allowing the SWF to be accessed via a password. This can help protect your SWF file should anyone find it and attempt to load it into a new FLA or larger application.

Debugging allows the debugger to access the SWF file. Be careful to not have this setting selected on your actual deployment SWF application. This is just a faux pas, and wouldn't affect the application for regular users.

Compress Your Movie is a handy setting that enables you to save 10, 20, or even 30% of your SWF file size. The SWF player has a built-in decompressor to decompress the file when it's loaded into the player. This attribute is very handy for rich applications with a lot of skin attributes, ActionScript, and embedded content, but less so for fully dynamic SWF files, which import almost everything at runtime.

JPEG Quality is an often overlooked setting that can help cut down on overall SWF size. This setting affects any bitmap that does not have an individual quality setting applied to it. Open the Properties panel for each bitmap in the Library to specify the quality attribute for individual images.

The two Audio settings affect only audio that has been placed directly upon a timeline from the Library.

The HTML Tab

When publishing an SWF file it is often advantageous to define an HTML wrapper for it. Although most browsers will open the Flash plug-in and play SWF content as is, it is difficult to know if the Flash version installed on the user system is new enough for your SWF content. The HTML wrapper can include JavaScript and object tag plug-in version checking to automate the process of checking for the correct version of Flash.

The HTML you choose can also be specific to a device or target browser. For example, mobile devices have their own markup for inserting plug-in objects upon a page. Just as the Flash Player version is carefully considered, so to should the target HTML be considered for compliance with end user systems. You can use the drop-down menu to select the HTML wrapper for your target system. The next section covers some simple JavaScript detection of the Flash Player version.

In the HTML Publish Settings panel (see Figure 1-8), you can select playback states and attributes to be placed in the object and embed tags automatically. You can, of course, create your own HTML, and change the attributes any way you like.

As with any Publish Settings selection, you should test each decision to see how it affects performance and cross-browser compatibility.

It is possible to create custom HTML templates.

The Image Tabs

If you return to the Publish Settings Formats panel, you see the three types of images that can be created when the SWF file is compiled. Each of these file types has options for setting image attributes when the images are created. The attributes include size, quality, smoothing, and more. As with all publish settings, trial and error is the best way to understand how the settings affect the image output. The goal in publishing images is to create the highest quality image with the lowest possible file size. This is always a compromise and dependent upon the complexity of the content within the image.

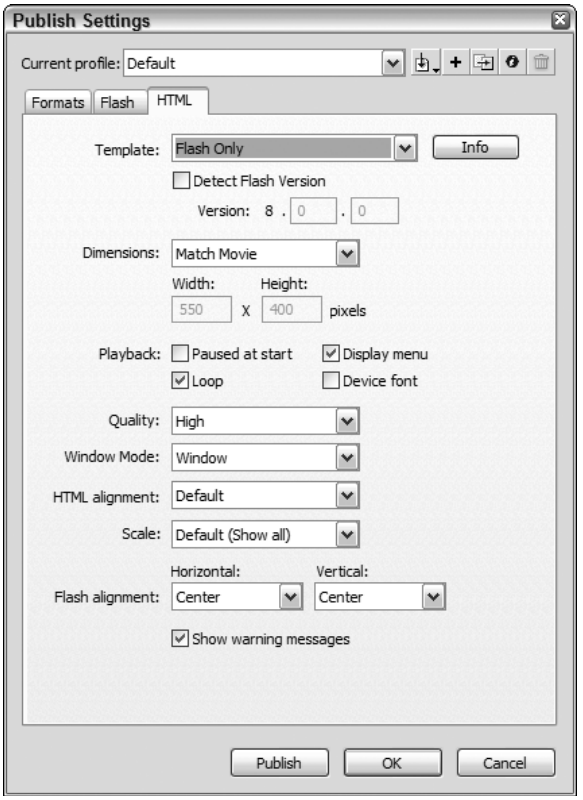

Figure 1-8

For example, Figure 1-9 shows the PNG tab in Publish Settings panel.

Options for Detecting the Flash Player

Once you have created the SWF file you'll want to be sure that all that hard work can be viewed on as many systems as possible. You have several options to choose from. The most robust, and easiest to use, is the Flash Detection Kit. Flash 8 integrates the kit into the HTML Publish dialog. The kit is also available as download. The downloadable kit can help implement the kit within servlet pages, as well as legacy HTML pages that require the detection update.

The default Flash Only HTML template provided by the Publish dialog can display your content when a Flash player is known to exist on a system. This template is fine for testing your SWF locally. However, using no detection leaves the plug-in issue resolution to the browser. Internet Explorer utilizes an alert message to ask the user if he would like to install or update the Flash player. It then automatically begins the install process using a simple certificate-based installation. Due to malware and other concerns, many users simply reject a certificate when presented. On all other browsers, however, a plug-in issue is often a simple message with no resolution or a link about where to go to get help about plug-ins. This is hardly acceptable, or user friendly.

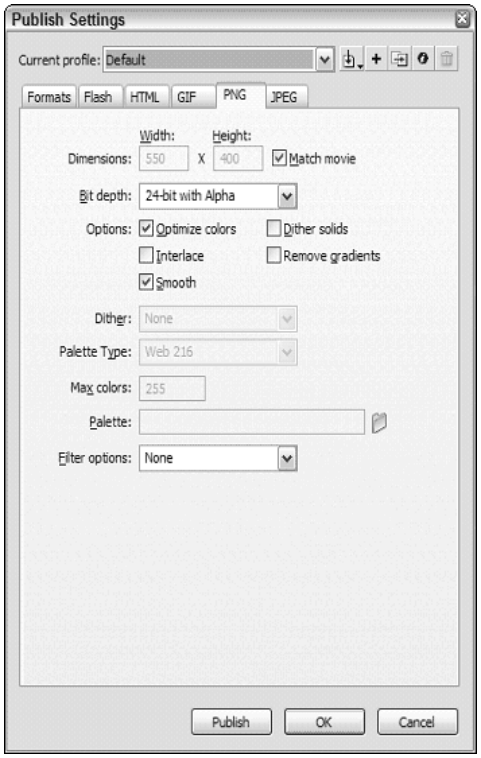

Figure 1-9

The detection kit exports a JavaScript-based SWF wrapper within an HTML page. After the kit has detected that Flash exists and the Version Is passes `true`, the JavaScript uses the `document.write` method to print the actual object and embed tags.

In the event that the Flash player is too old to play your SWF file, an alternative method is called that writes an image tag and specifies a `.gif` image.

If the browser does not have JavaScript enabled, the kit has a third and final method, which uses the noscript html tag to write an image tag and specify a `.gif` image.

Generally, I allow Flash to publish the kit in this way and then edit the output HTML file to behave the way I want. In some cases, replacing SWF content with GIF is acceptable, but generally not for SWF applications. In these cases it is better to use the `document.write` method to print HTML with instructions on how to resolve a plug-in update. This enables users to update their systems, and understand why the SWF application may not have worked on their first visit.

For any plug-in detection, it is better to display what you can right away and seamlessly integrate each level of compliance into your page design. Then gracefully deal with updates without alarming warnings and zero content displays.

Summary

This chapter serves as a brief introduction to the IDE. The IDE has many facets that you will discover and use. You might wonder why these weren't covered, but this book focuses on ActionScript and it is the intention of this chapter to introduce you to the IDE so that you can begin and complete the exercises throughout the book. In each example you'll see small shortcuts and work flows that weren't covered here. Keyboard shortcuts can speed your work a great deal, and each operating system has slightly different keyboard shortcuts. If a shortcut encountered in the book doesn't seem to be working, try using the manual menu system to find the action. In most cases, the keyboard shortcut is listed right next to the action in the menu system. Additionally, Appendix B is all about keyboard shortcuts.

The Flash IDE is extremely malleable. You may find that the work flows presented here are completely foreign to your method of working. Because the IDE is so flexible, I encourage you to deviate when possible and explore the IDE to see what it is capable of. The IDE also runs on basic settings files found in the application settings folder on your operating system. Sometimes if you wish you had a particular option, or could remove or simplify an aspect of the IDE, it isn't difficult to go poking around the Application Settings to see what you can find.

Exercises

1. Create a new FLA (Macromedia Flash document) by selecting File➪New and choosing Flash Document from the New Document panel. Create a Layer, and place a vector drawing or image object onto the stage. Place a simple `Hello World` ActionScript on the frame. Rename the layer `test` and save your FLA. Open a new FLA. Return to your original FLA, right-click (option+click) the frame with the `Hello World` ActionScript, and select Copy. Paste the frame into your new SWF. Examine the results.
2. Open a new FLA and open the Library. Add a symbol using the panel's context menu. Add a button or Movie Clip. Double-click the instance you just created and create some graphics within it. Open a new FLA file. Open the new FLA file's library. Return to the original FLA file's library and drag the object instance you created to the new FLA. Notice that the symbol now resides in both libraries.
3. Open multiple FLA files. Move between the FLA files by using the tab system at the top of the IDE. Minimize or un-maximize one of the FLA files. Note how the IDE enters a collapsible window mode.