1

Introduction to ActionScript

IN THIS CHAPTER

- What Is ActionScript?
- What Is ActionScript Good For?
- What ActionScript Can't Do
- Variables
- Objects and Object-Oriented Scripting
- Object and Frame Actions
- Dot Syntax
- Properties
- Methods
- Events and Event Handlers
- Functions
- Conclusion

What Is ActionScript?

ActionScript is the scripting language Flash uses to control its movies and the objects within those movies. If you want to do anything interactive in Flash, you'll need to use ActionScript. It allows you to execute different actions in a movie depending on what a user does or on what frame of the movie is being played.

ActionScript looks a lot like JavaScript, which Macromedia (the folks who wrote Flash) did on purpose. A specification called ECMA-262 was written to provide an international standard for the JavaScript language. ActionScript in Flash MX is based on the ECMA-262 specification, so if you've used JavaScript before, a lot of ActionScript will look familiar to you. If you haven't used JavaScript before, don't worry—you'll get it. Throughout this book, I'll be referring to *actions*. This is a general term, and an action roughly means "a chunk of ActionScript code that does something."

The biggest leap in the abilities of ActionScript occurred when Flash 5 was released. Flash 6 is the next version, and while the additions to ActionScript aren't as grand in this version, they do round out ActionScript into a full-fledged scripting language. Check out Chapter 2 to find out precisely what's new in Flash MX.

What Is ActionScript Good For?

Here's some of what you can do using ActionScript:

- Create multiplayer games.
- Create engaging, user-aware navigation.
- Send data to middleware, like PHP and Cold Fusion.
- Create and parse XML objects.
- Allow for a more responsive Web site.
- Make your site more accessible to readers.
- Make your site more international in scope (Flash MX supports Unicode).
- Communicate with JavaScript.
- About a billion other things. You'll see.

What ActionScript Can't Do

- ActionScript can't talk directly to a database—you'll still need to use middleware, like PHP, Cold Fusion, or ASP, to do that.
- ActionScript can't access the Window or Document objects that JavaScript can.

- You can't use exception handling with `try`, `throw`, or `catch`.

Variables

If you're completely new to programming, it'll take a little while to master the fundamentals, depending on how linearly you can make your brain work. One of the most basic programming concepts is the variable. This is the same variable you saw in algebra class in junior high. Some simple examples:

```
// "x" is the variable
x = 3;

// "message" is a variable that holds a string,
// i.e., usually text
message = "Please press the next button."
```

You'll probably use variables mostly to keep track of what the user is doing and what state certain movie clips are in. If this isn't clear right now, keep reading—as you see more examples, it should become clearer.

Objects and Object-Oriented Scripting

Both ActionScript and JavaScript are called *object-oriented* scripting languages. Let's go over what this means, since it's an odd concept if you haven't been exposed to it before.

Scripting languages organize information into groups called *classes*. You can create multiple instances of these classes, which are called *objects*.

Classes and objects are just ways to group together chunks of information. To use a real-world analogy, you could create a "bicycle" class. That bicycle has lots of different properties to it, such as how many gears there are, how big the tires are, and what color it is. You can also perform some actions with the bike: You can, among other things, pedal, brake, and turn (these are called *methods*).

In Flash MX, all movie clips are objects (also called *instances*) of the class MovieClip. Since all movie clips are

objects of the same class, they all have the same properties and the same methods (that'll make sense soon—hang in there!).

Buttons are also objects. While there are some significant differences between buttons and movie clips, which we'll cover later, I consider it—as do many other Flash folk—useful to think of buttons as a kind of movie clip.

Flash has a number of predefined objects you can access: Accessibility, Arguments, Array, Boolean, Button, Capabilities, Color, Components, CustomActions, Date, FStyleFormat, Key, LoadVars, Math, Mouse, MovieClip, Number, Object, Selection, Sound, Stage, System, String, Textfield, TextFormat, XML, and XMLSocket. If this is an overwhelming list now, don't worry—we'll spend much of this book looking at them. Those of you coming from Flash 5 will notice Macromedia has incorporated about twice as many predefined objects in Flash MX. This is a good thing. It means we have more tools in our programming toolbox to create cool stuff (and make money) with.

We'll be seeing all of these objects throughout this book. These objects are treated with excruciating detail in Appendix A, "ActionScript Reference."

Creating a Class

You don't have to restrict yourself to using only classes that Flash has provided, such as MovieClip. You can create your own classes using constructor functions. This is pretty advanced stuff, and if you can't think of why you'd want to create a new class, don't worry about it—usually, only advanced programmers build their own classes. This section is for them. Say you want to create a 1980s band:

```
function HairBand(p,s)
{
     this.hair = "big";
     this.hair_dye = true;
     this.number_members = p;
     this.number_synthesizers = s;
}

function Breakup()
{
```

```
    this.hair_dye = false;
    this.hair = "crew cut";
}

// Now, actually create two objects using
// the HairBand constructor function.
kajagoogoo = new HairBand(3,4);
softcell = new HairBand(2,1500);

// Create a method for a hairband
HairBand.kajagoogoo.partyover = Breakup;
```

Object and Frame Actions

Here's the basic structure of an action:

```
whenSomethingHappens(input variables)
{
    do stuff
}
```

We'll be elaborating on this basic structure significantly.

There are two kinds of actions: frame actions and object actions.

Object Actions—Movie Clips

Object actions are actions, or chunks of ActionScript code, that are attached to an object. Most of the time, an object is a symbol that's either a button or a movie clip. Graphic symbols can't have actions, nor can shapes you draw on the stage that aren't symbols (unless you create an empty movie and *then* draw in it. But more on that later). You can create your own objects, as we just saw.

An object action is associated with an instance of a symbol, not with the symbol itself. Here's an example (see Figure 1–1).

1. Load the movie *chapter1/fish_drag.fla*.
2. Control ➠ Test Movie.
3. Notice by positioning the cursor over each of the fish and dragging your mouse, you can move the fish on the left, but not the one on the right. In fact, no matter

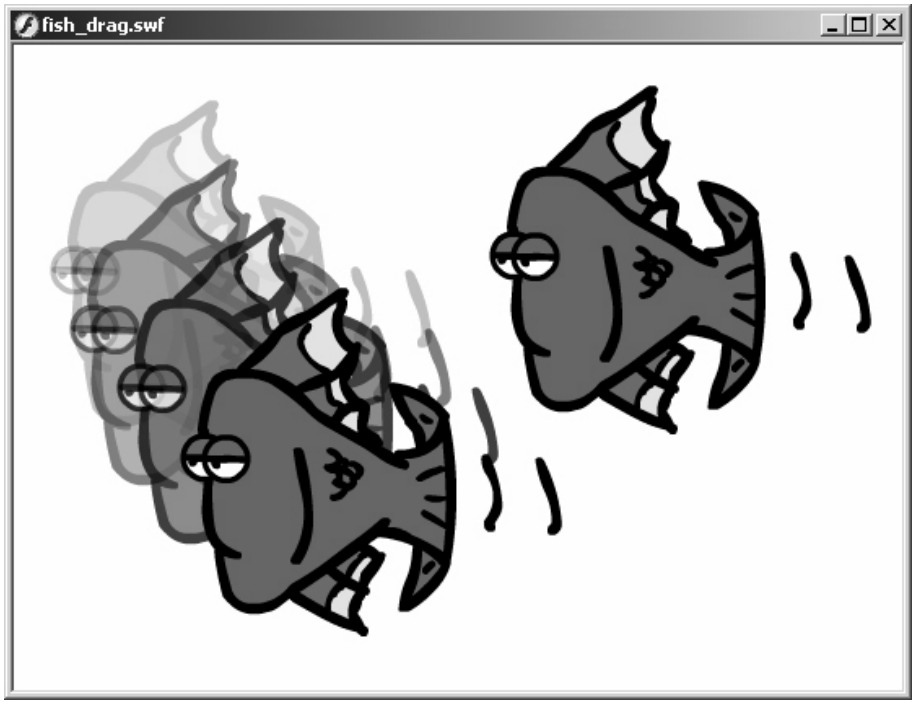

FIGURE 1–1 Dragging one of the fish

where you click, you move the fish on the left and the right one stays put.

Both fish are instances of the same movie clip symbol, but only one of them has an action attached to it. Let's see what that action looks like. Exit from the movie test and return to the Flash editor (the main program). There are many ways to configure this new user interface (UI) that Macromedia has implemented for Flash MX. To keep life easy, go to Window ➠ Panel Sets ➠ Developer and choose your screen resolution.

The Actions panel appears, as shown in Figure 1–2.

I recommend turning on line numbers. That option is launched by clicking the View Options menu, located under the little pushpin on the right side of the Actions panel, as in Figure 1–3.

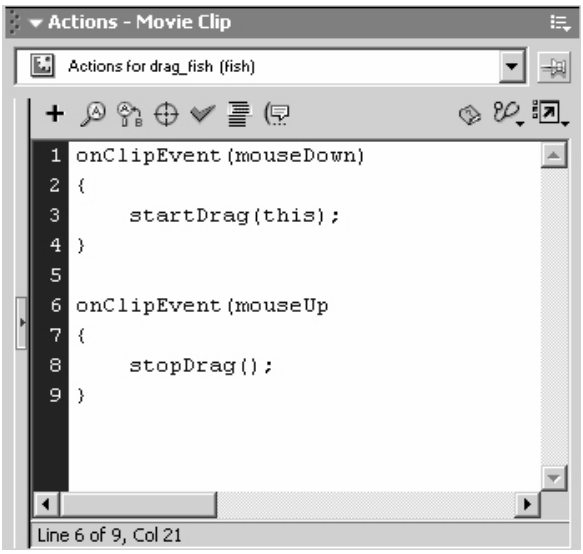

FIGURE 1–2 The Actions panel (Expert Mode)

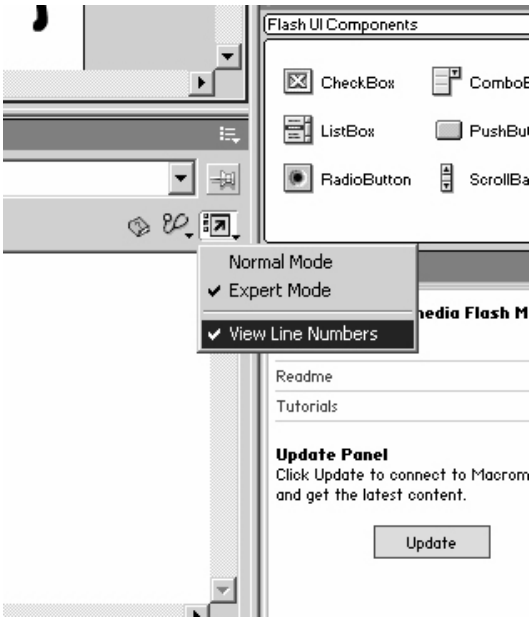

FIGURE 1–3 View Options menu

You can also see I've chosen Expert Mode (more on that later). The code that appears should be:

```
onClipEvent(mouseDown)
{
     startDrag(this);
}

onClipEvent(mouseUp)
{
     stopDrag();
}
```

These two functions never leave this object instance, no matter where this instance appears in the movie. The actions will happen to this object from the first frame of the movie all the way to the last frame of the movie. Object actions don't care what frame the movie is currently on (as long as the object actually exists in the movie at that frame).

The two functions you see are `onClipEvents`. `onClipEvent` is known as an *event handler.* An event is something that happens: for instance, a movie finishes loading, the user presses a mouse button, or the user hits the space bar. An event handler is a piece of Flash that is constantly looking for these events and lets ActionScript know when one of them occurs.

Since the fish symbol is a movie clip, each instance of that symbol gets an event handler called `onClipEvent` that will constantly look at the mouse and keyboard to see if the user is doing anything. If the user does do something, like pressing down the mouse button, say, the event handler looks at the ActionScript to see if that event exists anywhere in the code.

In this code, we're looking for two events, `mouseDown` and `mouseUp`. These refer to what the mouse button is doing. Is the mouse button currently being pressed down or was it just released?

If the mouse button was pressed, then `onClip-Event(mouseDown)` is called, and everything inside the first set of curly braces is executed. As it happens, there's only one thing to do:

```
startDrag(this);
```

`startDrag` is what's called a *method*. Briefly put, a method does something (as opposed to a *property,* which just holds a specific bit of information). We'll examine this method briefly now and in more detail later (it's darn useful).

`startDrag` causes the object in question to mirror the motion of the mouse cursor. Notice that we didn't use `startDrag()`, but rather used `startDrag(this)`. The `startDrag` method requires a target—that is, it needs to know what it should start dragging. The easiest way to reference the current object is just to call it `this`. You'll see `this` being used again in this book.

The other way to refer to the object is

```
startDrag(_root.drag_fish)
```

where `drag_fish` is the name of the instance of the fish symbol. The `_root` part means "start looking from the top of the movie hierarchy." If this is confusing, don't worry. It is covered later in this chapter in the section "Dot Syntax."

Object actions that are on a movie clip have to be inside of an `onClipEvent`. The events are:

- `load`
- `unload`
- `enterFrame`
- `mouseMove`
- `mouseDown`
- `mouseUp`
- `keyDown`
- `keyUp`
- `data`

Object Actions—Buttons

The only real difference between actions that are attached to buttons instead of movie clips is that the event handler for buttons is `on` instead of `onClipEvent`. Otherwise, they're pretty much the same. The events for `on` are:

- `press`
- `release`
- `releaseOutside`
- `rollOver`
- `rollOut`

- `dragOver`
- `dragOut`
- `keyPress`

Here's an example that I doubt will find its way into one of your movies, but illustrates the `on` event handler nicely.

1. Load the movie *chapter1/face_button.fla.*
2. Control ➠ Test Movie.
3. Notice when you press down on the mouse button, the fish appears.
4. Return to the Flash editor.
5. Click on the button symbol.
6. Open the Actions panel to see the actions.

Frame Actions

Frame actions are like object actions, except that the actions are associated with a certain spot in the timeline instead of an object. If a frame has some actions associated with it, those actions are carried when the playhead enters that frame.

A simple example would be stopping a movie at the last frame so that it doesn't loop, which Flash movies do by default.

1. Open *chapter1/fish_cruise.fla*.
2. Notice the layer *Actions*. Creating this layer isn't necessary to place actions on, but I find it useful.
3. Double-click on the small *a* on the timeline on frame 60 on the actions layer.
4. Open the Actions panel.

The Frame Actions panel looks like as shown in Figure 1–4 (notice it looks almost exactly like the Object Actions panel), with this code:

```
stop();
```

As you might predict, this command stops the movie in its tracks. It stays stopped unless some other action starts it up again.

Flash will place a frame action only on a keyframe. If you try to place a frame action on a regular frame, Flash will look

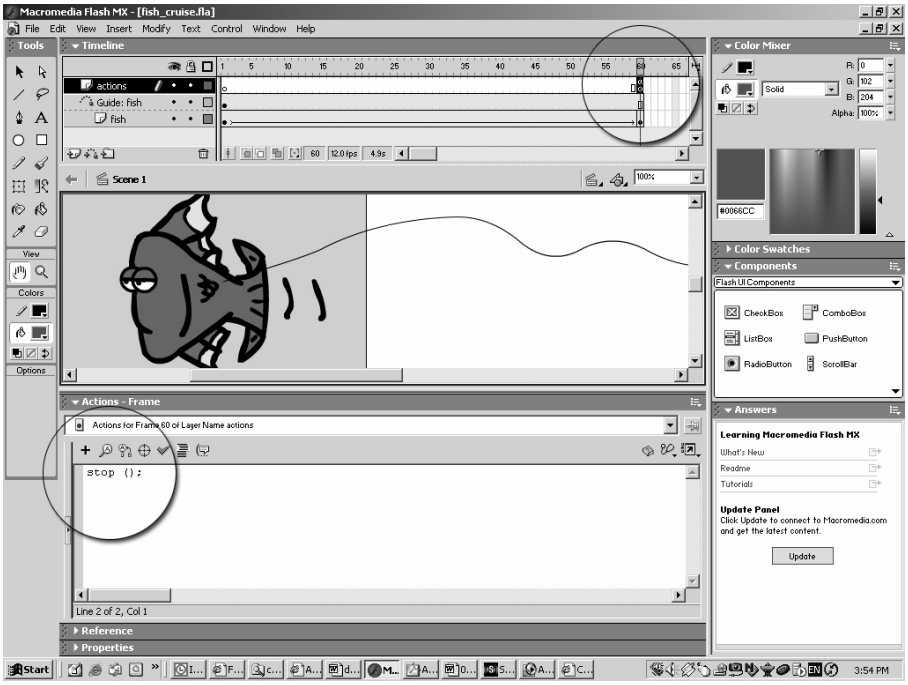

FIGURE 1–4 The timeline and the Frame Actions panel

backwards in the timeline until it finds a keyframe and place the frame action on that keyframe, not on the regular frame you clicked on. I recommend placing all of your frame actions on a separate layer—it makes organization much easier.

Dot Syntax

ActionScript uses what is called *dot syntax*. For example, if you have a movie clip called *red_shirt* inside of the movie clip called *santa_claus,* then one way to access that object is

```
_root.santa_claus.red_shirt
```

`_root` is the base of all Flash movies. If you want to find out where *red_shirt* is on the stage, you could use

```
xPosition = _root.santa_claus.red_shirt._x;
```

`_x` is a property that returns the horizontal position of the object. We'll talk more about properties soon.

If you're familiar with JavaScript, dot syntax will look familiar to you: for example,

```
document.myForm.textBox.value = "Try again!";
document.image['nav'].src = 'images/clickme.gif';
```

If you're familiar with Flash 4, you're used to the *slash* syntax. Hopefully, you'll find dot syntax a little more intuitive and easier to use.

Properties

A *property* is a piece of an object. Most objects are simply collections of properties. Examples of some movie clip properties are:

- how wide it is (`_width`)
- where it is on the stage (`_x` and `_y`)
- what frame is currently being played (`_currentframe`)
- how transparent it is (`_alpha`)
- its name (`_name`)
- whether it's visible or not (`_visible`)

Most properties can be read and altered. For example, let's see how wide a movie clip called *clue* is:

```
clueWidth = _root.clue._width;
```

If we don't like the value of `clueWidth`, we can change the width of *clue* like this:

```
_root.clue._width = 110;
```

or

```
_root.clue._width = _root.clue._width - 40;
```

All movie clip properties start with an underscore (_). That's just how Flash is. As we continue through the book, almost every example will involve lots of properties, so if you're not clear on the concept yet, you will be soon.

Methods

A method is something an object can do or something you can do to an object. For example, here are some things methods can do:

- Stop a movie clip.
- Go to a certain frame and start playing there.
- See if a movie clip is over another movie clip.
- Hide the mouse cursor.
- Calculate a cosine.
- Set the volume of a sound being played.

Let's go to frame 10 of the *clue* movie clip and start playing from there:

```
clue.gotoAndPlay(10);
```

The method here is `gotoAndPlay`. All methods live inside objects—they don't exist on their own.

NOTE Even when you see methods that look like they don't belong to an object—for example, a frame action whose only line of code is `stop()`—it's understood by Flash that the object in question is the movie clip the frame is in: It will always assume `this.stop()`. If the frame is in the main timeline, then the default timeline is `_root`, resulting in `_root.stop()`.

Remember we said that all movie clips you make are objects based on the master MovieClip class? Well, that class has a whole bunch of properties and methods associated with it, and when you make a MovieClip object, all those properties and methods become a part of your movie clip. This is a good thing—those properties and methods are vital to creating interactive Flash, and it would be a pain to have to create all those properties and methods yourself.

Events and Event Handlers

In the interactive world of ActionScript, events are happening all the time. However, your movie wouldn't know about any of

those events if it weren't for the event handlers that are triggered when an event happens.

There are two ways to handle events. We've already seen it for buttons:

```
on(press)
{
  do something
}
```

Here's what an event handler for movie clips looks like.

```
onClipEvent(enterFrame)
{
  do more stuff
}
```

The other way to handle this is a little different:

```
myButton.onPress = showFishFunction
```

In this case, we're using the `onPress`, instead of the `on`, event handler. If the user presses the button, then the `showFishFunction` is called. We'll see more of this kind of event handling later in the book.

Buttons, movie clips, and the Mouse object all have their own event handlers.

What's a function? I'm glad you asked.

Functions

If you're getting fancy and writing some complicated Flash, you may find that the objects, methods, and properties that Flash provides don't quite meet your needs. Fortunately, Flash provides a way for you to create your own objects, properties, and methods using *constructor functions*.

If the predefined functions Flash provides don't quite meet your needs, you can create your own functions (if you've done work in JavaScript, the concept of functions will be quite familiar to you).

A function is a set of instructions that's executed only at certain times. We'll be using them in later chapters. We used functions when we created HairBand earlier in this chapter.

Functions can look (and act) a lot like methods. You'll see some of this overlap later as well.

Conclusion

If you're brand new to ActionScript, much of the preceding chapter may have sounded like gobbledygook. If that's the case, don't worry—we'll go through a lot of examples that will make this theoretical talk more concrete. If you're a grizzled scripting veteran, you probably realize that you're already on familiar ground, and ActionScript will come easily for you.