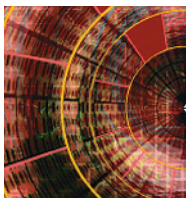
one

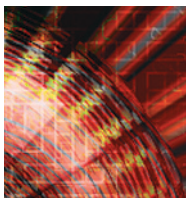
CHAPTER

ActionScript for Non-Programmers

For many of us, *making the transition from user to programmer—from working "manually" with an application program's interface to building scripts that automate complex operations—can be an intimidating prospect.* Coding, *the common name for writing scripts in a programming language, seems to imply a set of cryptic, inflexible rules that may take ages to learn.*



It's true that there is a lot to learn, and your scripts do need to follow the rules in order to work. But Flash MX provides an interface to ActionScript that does some of the work of creating scripts for you. Learning to write Flash MX scripts will be much less intimidating once you become familiar with the ActionScript environment and its core mode of operation. You'll gain this familiarity as you work through the examples in this book.



The most important tool that Flash MX provides for working with ActionScript is the Actions panel. It is your interface for creating simple to complex scripts. It allows you to drag and drop commonly used code lines into your script, as well as to enter code directly. In this chapter, you'll become familiar with the Actions panel and learn how to use its many useful and timesaving features. You'll learn how to take advantage of features like the Actions toolbox, the Reference panel, auto-formatting, code hints, and the View Line Numbers option. You'll also begin to learn some nuts and bolts about how ActionScript works.

Getting Comfortable with the Actions Panel

For many designers, the Actions panel can be somewhat intimidating. It's not as scary as it seems—even when you're working in the dreaded Expert mode. In this book, you'll be spending all of your time in Expert mode. Most of the coding in this book is infeasible (or awkward at best) in Normal mode.

The main difference between Expert and Normal mode is that in Expert mode, you can type in the code directly, as well as drag it from the Actions toolbox. In Normal mode, you can only drag and drop code from the Actions toolbox; you cannot enter code manually.

In many ways, this book is about making anyone who is not experienced with programming comfortable in Expert mode. You'll start to see the blank, white space in the Actions panel as a doorway to endless possibilities, rather than as a members-only door accessible only to a select few.

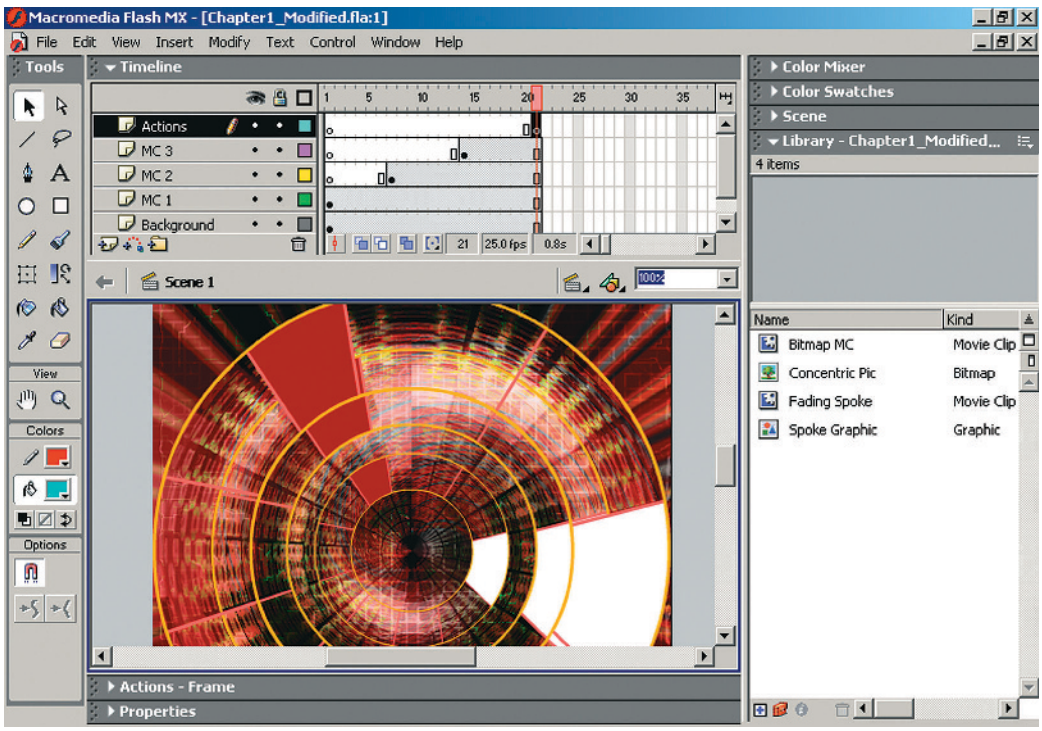
Adding ActionScript in the Actions Panel

This is a book in which you'll learn by doing. Throughout this book, you'll be building scripts, testing the results, and analyzing what the elements of each script do. By following the examples here, and by continuing to practice on your own, you'll develop firsthand knowledge of how ActionScript works. In this first example, however, you'll just start exploring the Actions panel. In the second half of the chapter, you'll try your hand at scripting.

1. Open the File

WWW.

Begin by opening the file named **Chapter1_Start.fla** in Flash MX (1.1). If you haven't already downloaded this book's sample movie clips from the Web, see the Introduction for a complete



1.1: The **Chapter1_Start.fla** *file in Flash MX*

description. Save the file to your local hard drive with the name **Chapter1_Modified.fla**. (To see a finished version of this movie, open **Chapter1_Final.fla**.)

Going from the bottom to the top of the timeline, you can see that the **Chapter1_Start.fla** file has five layers: Background, MC1, MC2, MC3, and Actions. The MC1, MC2, and MC3 layers contain the same movie clip, named Fading Spoke. The Fading Spoke movie clip fades a graphic symbol named Spoke Graphic from 100% alpha to 0% alpha.

2. Test the Movie

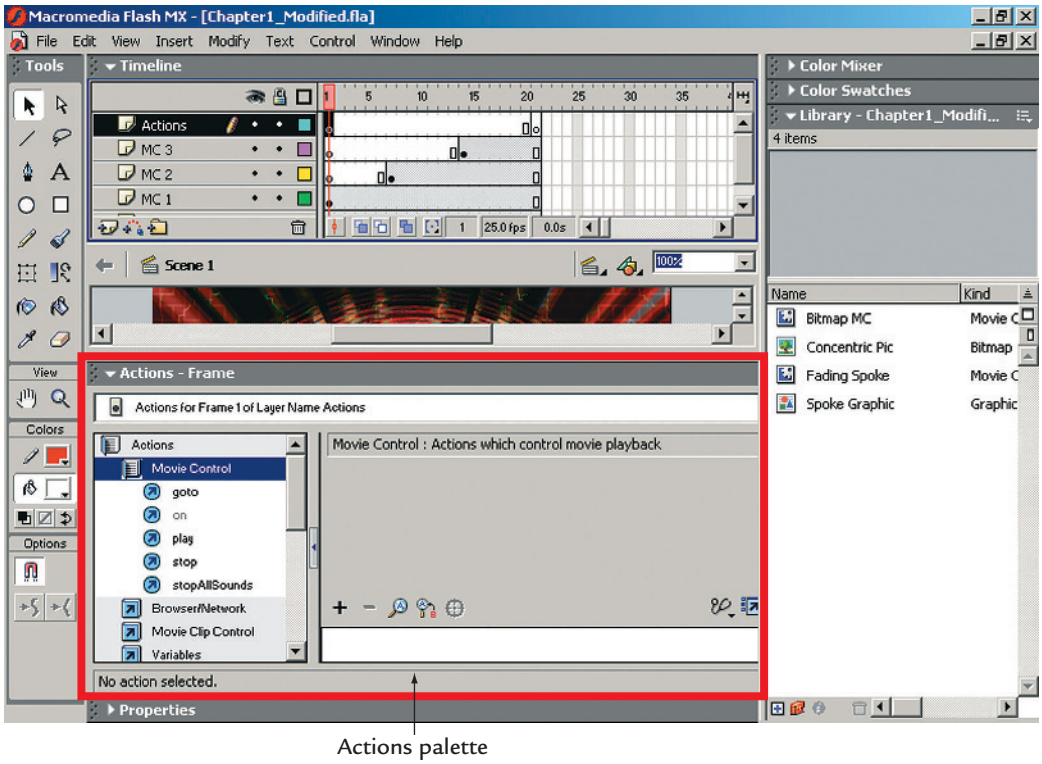
Test the movie before you enter any code. Press Cmd/Ctrl+Enter or choose Control ➢ Test Movie to see what the animation does.

Each instance of the animation goes through the alpha fade transition. The movie clips appear on the timeline in a staggered manner. The instance of the Fading Spoke movie clip on the MC1 layer is always visible, but the instance on the MC2 layer is visible only from frame 7 through frame 21, and the instance on layer MC3 shows up on frame 14 and plays through to frame 21.

You will see why we constructed the file this way in a moment. Now let's get ready to enter some simple ActionScript. Whenever you want to write or edit ActionScript, you will need to access the Actions panel. By default, the Actions panel is nested below the stage in Flash MX.

3. Toggle the Actions Panel

Close the Test Movie window and then open the Actions panel. To open the Actions panel, press F9 (1.2). This keystroke toggles the Actions panel between open or expanded and closed or minimized.



1.2: The F9 key toggles the Actions panel.

Notice that when you open the Actions panel, it partly obscures the stage; this makes the ability to toggle the panel on and off quite useful. You can toggle it on and off by choosing Window ➢ Actions.

The first time you open the Actions panel, it will be in Normal mode (as in 1.2). You will not be using the Normal mode in this book. Go directly to Expert mode.

4. Select Expert Mode

Click anywhere within the Actions panel to make sure it is selected, and then press Cmd+Shift+E (Mac) / Ctrl+Shift+E (Win) to go to Expert mode. (Using the keyboard shortcut for switching to Expert mode will have no effect if the Actions panel is not selected.) When you switch to Expert mode, you'll see that the white area on the right side of the Actions panel (called the *script window*) takes up much more space than it did in Normal mode.

On the left side of the Actions panel, you see the Actions toolbox (1.3). It contains a list of actions, organized in folders of categories—such as Operators, Functions, Constants, Properties, and so on. You can use the Actions toolbox to select actions. You won't be using the Actions toolbox for most of the examples in this book, because it's generally quicker to type commands than to work through the categories to select them. However, the toolbox can be useful until you're familiar with ActionScript syntax or if you need a reminder of which ActionScript commands are available. So let's take a few minutes to explore some of the useful aspects of the Actions toolbox.

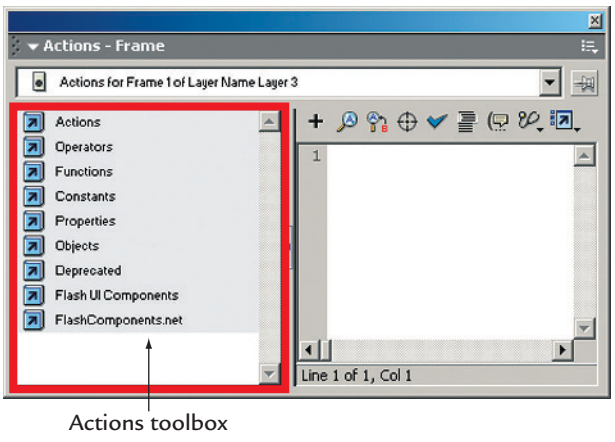
5. Expand Categories in the Actions Toolbox

Select frame 21 of the Actions layer. Click the Actions folder in the Actions toolbox to expand the category. You'll now see a list of subcategories such as Movie Control, Browser/Network, Movie Clip Control, and so on. Click the Movie Control folder to expand the Movie Control subcategory, and you'll see a list of actions for controlling a movie.

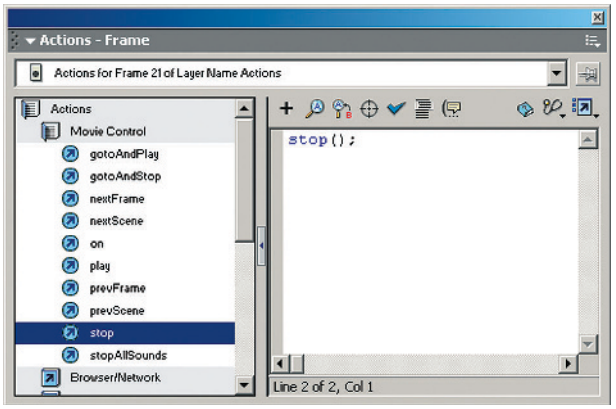
6. Double-click the Stop Action

Double-click the stop action in the Actions toolbox (1.4). Flash will add the following line to the script window on the right:

```
stop();
```



1.3: The Actions toolbox organizes actions in categories.



1.4: Double-click the stop action in the Actions toolbox to add it to the script window.

7. Test the Movie

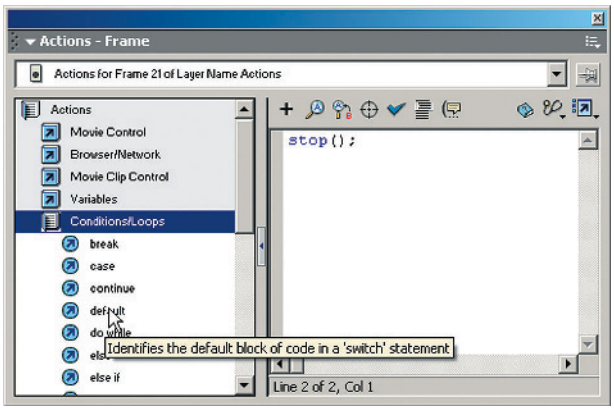
Now press Cmd/Ctrl+Enter to test your movie again. You'll notice that the spokes stagger onto the stage, but once they get there, they each cycle through the fade at a different time and keep repeating. That's because even though you've told the main timeline to stop in frame 21, each Spokes Fade movie clip timeline continues to cycle through the frames within each movie clip. Each animation started on a different frame of the main timeline, so even though they are all the same movie clip, they are going through their fading cycle at different times.

On the positive side, the value of using the Actions toolbox for entering ActionScript in the Actions panel is that you are assured of getting the syntax correct. The downside is that it would be very taxing on your time to need to hunt down the actions all the time. Usually, you'll want to use the Actions toolbox to enter code only if you're having trouble recalling the syntax of a certain line of ActionScript.

Getting Quick Information about Actions

A particularly useful aspect of the Actions toolbox is that it can give you some quick information about actions. Let's look at an action that's not as obvious as the `stop()` action (in terms of what it does or what it is used for) to demonstrate this feature:

1. Close the Test Movie window, and then click the Movie Control folder in the Actions toolbox to minimize it.
2. Click the Conditions/Loops folder.
3. Roll the cursor over the `default` action. Don't click; just place the cursor over the action and hold it there. After a second or two, you'll see some pop-up text explaining what the `default` action is used for (1.5). In this particular case, unless you know about `switch` statements, the explanation only brings up more questions.

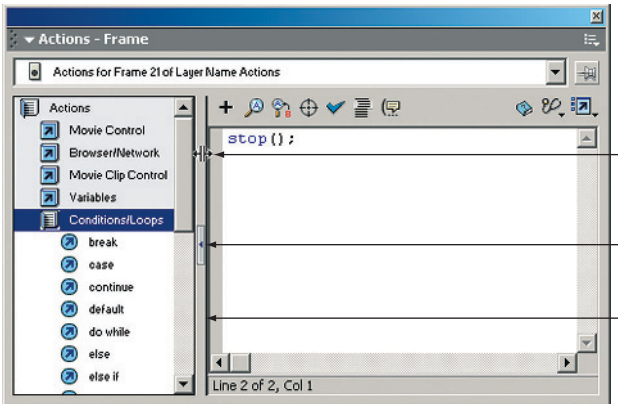


1.5: Roll your mouse over an action to get a brief explanation of what it does.

Flash MX provides information about ActionScript and specific actions in several ways. Later in this chapter, you'll use the Reference panel, code hints, and syntax coloring, all of which provide different kinds of information. While you're learning how to use ActionScript, these features will be invaluable. First, however, let's finish looking at the Actions toolbox.

Resizing and Closing the Actions Toolbox

If you plan to use the Actions toolbox regularly, you can minimize how much space it takes up. Hold your mouse over the gray bar that separates the Actions toolbox from the script window. Don't place it over the button with the left-pointing arrow (that's the minimize/maximize button, as you'll see in a moment). When you see the cursor change to two lines with arrows pointing left and right, you can click and drag to resize the window. Drag the bar to the left to make the Actions toolbox smaller (1.6).

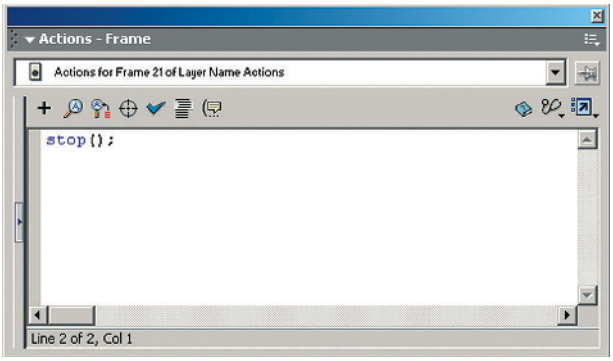


1.6: Resizing the Actions toolbox

NOTE Resizing the Actions toolbox is temporary. If you minimize and then maximize the Action toolbox, Flash does not remember the position of the resized Actions toolbox. Similarly, Flash will not remember the position of the Actions toolbox if you close and reopen Flash. However, minimizing the Actions toolbox is retained.

While the Actions toolbox has its uses, it will often be more trouble than it's worth. As you become more proficient with ActionScript, the space that it takes up will probably be more valuable to you than using it to enter actions or to find information about actions. Fortunately, you can close, or minimize, the Actions toolbox. To minimize the Actions toolbox, just click the small button in the middle of the bar that separates the Action toolbox from the script window (1.6).

When the Actions toolbox is not minimized, the arrow on this button will point left. When the Actions toolbox is minimized, the arrow will point right (1.7). You can expand the Actions toolbox by clicking the button again. You're going to leave the Actions toolbox minimized from now on.



1.7: Minimizing the Actions toolbox

Writing a Simple Script

At this point, this chapter's example consists of three instances of the Fading Spoke clip, each fading at different times. Now you'll make it more interesting by adding some code. What this simple demonstration program does, however, is not as important as what it can show you about the Flash MX interface to ActionScript. You'll use these few lines of code to become more familiar with the Actions panel. In particular, you'll explore two more features that provide information: the Reference panel and code hints.

In the first exercise, you applied some simple ActionScript to a keyframe on the timeline (by double-clicking the stop action in the Actions toolbox). In this exercise, you'll also use another way of applying ActionScript.

Keyframes on a timeline (both on the main timeline or on a timeline within a movie clip) are one of three places that you can assign ActionScript. The other places that you can apply ActionScript are directly on movie clips or buttons. As you will see in this exercise, when you apply ActionScript to a movie clip, you can control how that movie clip behaves programmatically.

Beginning the Script

In this example, you're going to write a relatively simple script that will control the Fading Spoke movie clips in two ways. First, the code will scale the movie clip instances, and then it will randomly rotate the movie clip instances. You will use the same script for all three instances of the Fading Spoke movie clip.

1. Select the Movie Clip

Move to frame 1 on the main timeline and select the Fading Spoke movie clip instance on the MC1 layer. You will apply ActionScript to this movie clip.

2. Assign ActionScript to Movie Clip

If the Actions panel is not open, press F9 to open it. Notice that the Actions panel is now labeled Actions - Movie Clip (1.8). This is an affirmation that you're now assigning ActionScript to a movie clip, rather than to a keyframe.

NOTE If the Actions panel is labeled Actions - Frame, you will be assigning the ActionScript to the keyframe on frame 1 on the MC1 layer, not the movie clip on frame 1. Make sure the Actions panel says Actions - Movie Clip before continuing with the exercise.

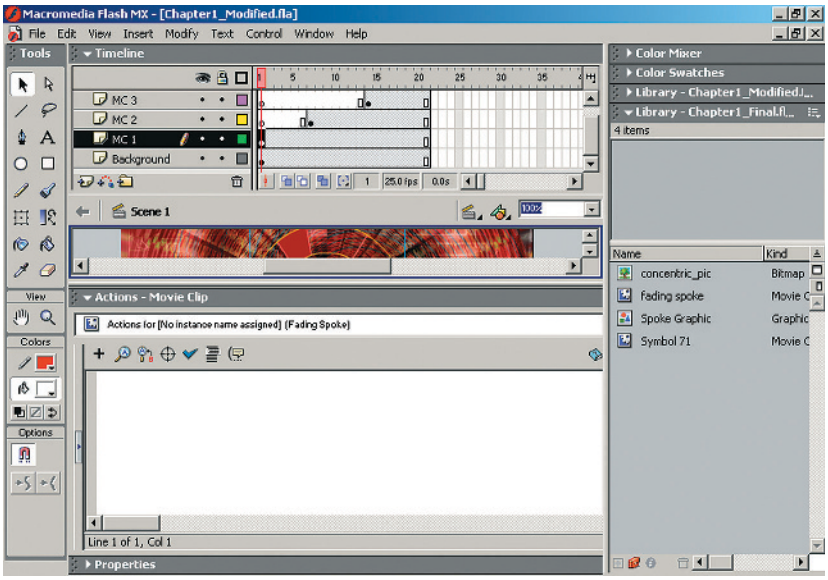
Enter the following code in the Actions panel:

```
onClipEvent() {
```

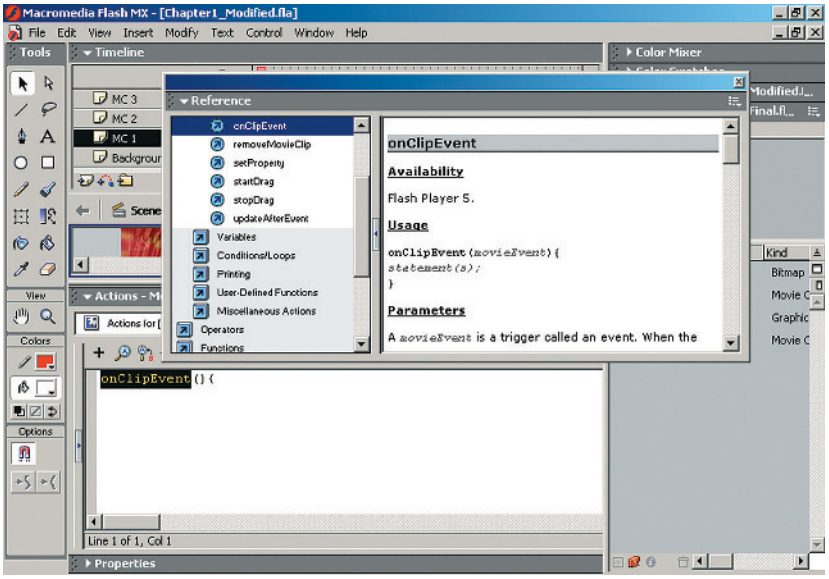
This is the first line of your code. Before you finish entering the code, let's look at a few useful features in the Actions panel.

Using the Actions Panel for Reference

If you are wondering what `onClipEvent()` does, Flash MX offers an easy way to find out. The Reference panel, accessed by clicking the Reference button in the Actions panel, lets you look up what a particular action does.



1.8: Make sure you that you are assigning the code to the movie clip instead of the keyframe.



1.9: When you click the Reference button with ActionScript code selected in the Actions panel, Flash MX displays the Reference panel entry for that command or other keyword.

1. Use the Reference Panel

Use your mouse to drag over just the text `onClipEvent()` (don't select the parentheses and the open bracket).

Click the Reference button, which is the button with a little blue book icon on the right side of the Actions panel toolbar panel (see the sidebar "Un-nesting the Actions Panel"). When you click the Reference button with an ActionScript keyword selected, Flash will open the Reference panel with that code already selected (1.9).

The left side of the Reference panel works similarly to the Actions toolbox in the Actions panel. You can select any action to get detailed information about it. The right panel displays information about that action.

You might have noticed that Flash MX does not come with an ActionScript reference guide. Now you know why: Flash MX has the reference guide built into it. In this example, you can see that the Reference panel displays a lot of information about `onClipEvent()`. If you scan through the information to the end, you'll notice some cross-linked topics that take you to information about related actions.

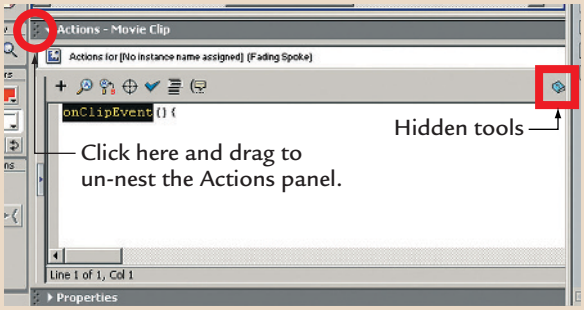
The Reference panel is particularly useful when you need information about various parameters for a specific action. For example, in this case, you'll be using the `load` parameter for the `onClipEvent()` action. Scroll down in the Reference panel until you see the `load` parameter and read up on what it does. But don't type `load` into the `onClipEvent()` action just yet. When you're finished, close the Reference panel.

Un-nesting the Actions Panel

At this point, you may run into a problem with the new Flash MX interface layout. Macromedia's redesign of the Flash interface in Flash MX is largely a substantial improvement for workflow and usability, but there are a few hitches here and there.

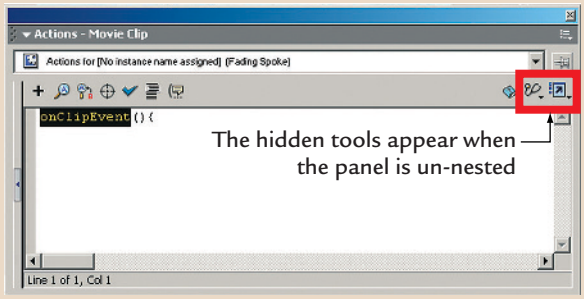
One problem crops up if you're working in 800×600 or 640×480 screen resolution. When working in lower screen resolutions, some useful tools in the Actions panel might become hidden. For instance, the screenshot below shows Flash MX at a screen resolution of 800×600. The illustration points out where some of the tools are hidden.

Fortunately, you can un-nest the Actions panel. To do this, click the small, gray, dotted area in the upper-left corner of the Actions panel and drag the Actions panel away from its nested position. This can be a little tricky because it's easy to inadvertently nest the Actions panel somewhere else within the Flash user interface. Try to drag the upper-left corner toward the center of the stage. This will keep it from nesting somewhere else.



After the Actions panel is un-nested, you can move it around by clicking and dragging anywhere along the top bar. If you want to re-nest the Actions panel, click in the upper-left corner along the gray bar (next to the minimize/maximize button) and drag the panel back into place. The figure below shows the Actions panel un-nested. Now you can see the buttons or features that were hidden earlier (outlined).

When the Actions panel is not nested, the F9 shortcut key toggles the visibility of the Actions panel on and off. You can also double-click the top bar to minimize the Actions panel as much as possible. Clicking the gray bar also minimizes the Actions panel, but not as much as double-clicking the blue bar above the gray bar.

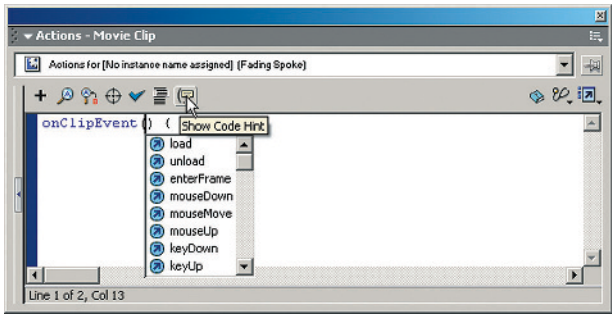


Please check outline placement for G0102. We didn't have a sample to follow.

2. View Code Hints

Now let's look at another useful feature in the Actions panel—code hints. Clicking the Show Code Hint button in the Actions panel toolbar produces a hint pop-up window, whose contents depend on the location of your cursor when you click the button. The Show Code Hint button is particularly useful if you need to know what parameters an action supports.

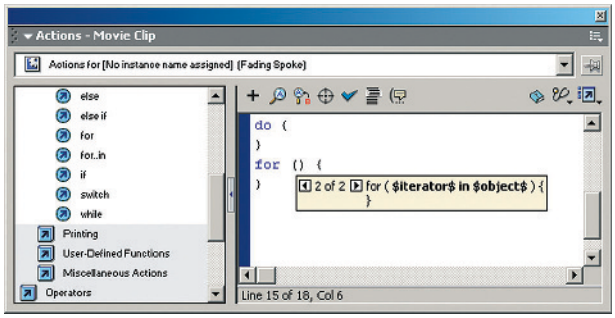
1. Place your mouse cursor between the opening and closing parentheses in the line of code in the Actions panel:
   ```
   onClipEvent(){
   ```
2. Click the Show Code Hint button in the Actions panel toolbar (1.10). In this case, the code hints show the various parameters available for the `onClipEvent()` action.
3. Double-click the `load` parameter to select it.



1.10: Click the Show Code Hint button to see code hints.

4. In some cases, clicking the Show Code Hint button will show you a series of hints. For instance, type the following in the Actions panel:

```
do {
}
for () {
}
```



1.11: Some actions have more than one hint associated with them.

5. Place your cursor between the parentheses after `for` and click the Show Code Hint button. The hint pop-up window will feature left- and right-pointing buttons that allow you to cycle through two hints for the `for()` action (1.11).
6. Delete the `do/for` code, leaving only the `onClipEvent(load) {` line of code.
7. Code hints don't always provide useful information. For example, type `getTimer()`, place your cursor between the parentheses, and click the Show Code Hint button. Flash will just repeat the action—`getTimer()`—in the pop-up hint window.
8. Delete the line `getTimer()`.

NOTE Usually, you will not see any hints if you click the Show Code Hints button when the cursor is anywhere outside the parentheses that go with an action that has parameters.

Completing the Script

Now that you have explored some of Flash's features for getting information—the Reference panel and code hints—let's return to the sample script.

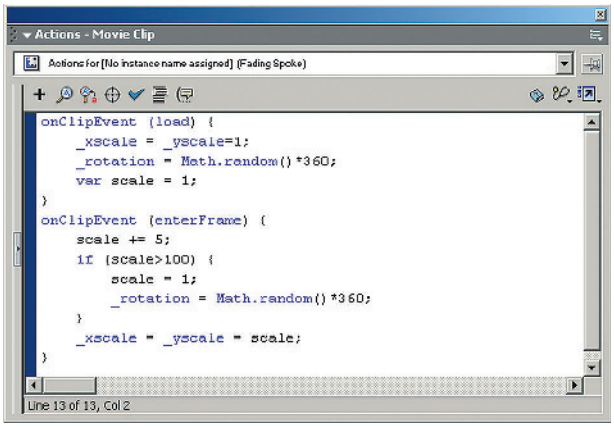
1. Finish the Code

Complete the code by adding the following ActionScript after the `onClipEvent(load)` line (1.12):

```
    _xscale = _yscale = 1;
    _rotation = Math.random()*360;
    var scale = 1;
}
onClipEvent(enterFrame){
    scale += 5;
    if (scale > 100){
        scale = 1;
        _rotation = Math.random()*360;
    }
    _xscale = _yscale = scale;
}
```

Within the `onClipEvent(load)` event handler, you're setting the scale of the movie clips in both directions (`_xscale` and `_yscale`) to 1%. You're also setting the rotation of the movie clip (`_rotation`) to a random rotation. This is done only once to set up the movie clips so they start out looking the way you want them to (scaled and rotated).

The `onClipEvent(enterFrame)` event is where the actual work takes place. You increase the scale in both directions by 5%, and then you check if you've scaled past 100%. If you have, you want to start over, so you set the scale back to 1%. For additional flair, each time you start the scaling over, you randomly rotate the movie clip again.



1.12: Complete the ActionScript for the Fading Spoke movie clip instance.

2. Test the Script

Press Cmd/Ctrl+Enter to test your movie. Notice the outer two spokes look the same, but the inner spoke grows more and rotates each time it starts at the beginning again. You'll learn more about the `onClipEvent(load)` and `onClipEvent(enterFrame)` event handlers that make this happen at the end of this chapter and in the following chapters.

Using Other Actions Panel Features

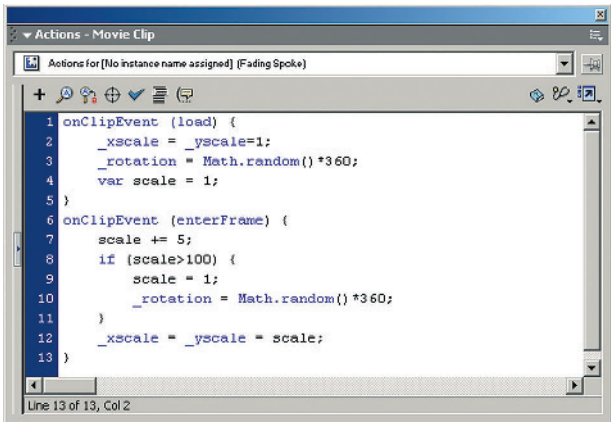
Now that you have all of your code, let's look at a few more features available in the Actions panel. You'll learn about line numbering, syntax coloring, and other Preferences settings, as well as the Find and Replace function.

Adding Line Numbers in the Actions Panel

Line numbers make it easier to refer to specific lines in the script. Click the View Options button in the Actions panel toolbar (the button on the far right side) and select View Line Numbers. This will add numbers before every line in your script. (1.13)

Line numbers are useful for several reasons:

- They make it easier for you to stay oriented when you're dealing with a very long script.
- They are helpful when you're debugging code. As you'll see later in this book, Flash lets you know which line in the script contains a problem.
- Line numbers make it easier to collaborate with other programmers and designers because you can reference the lines in the script using the numbers.



1.13: Adding line numbers to the Actions panel

In our example, you should have 13 lines of code, including the close bracket on the last line. The line `onClipEvent(load) {` should be on line 1, and the line `onClipEvent(enterFrame){` should be on line 6.

Setting Syntax Coloring

Look carefully at the script in the Actions panel and notice that some of the script is colored dark blue and some of it is colored black. Flash MX automatically colors actions for you as you write your script. If you prefer, you can specify colors used for syntax coloring.

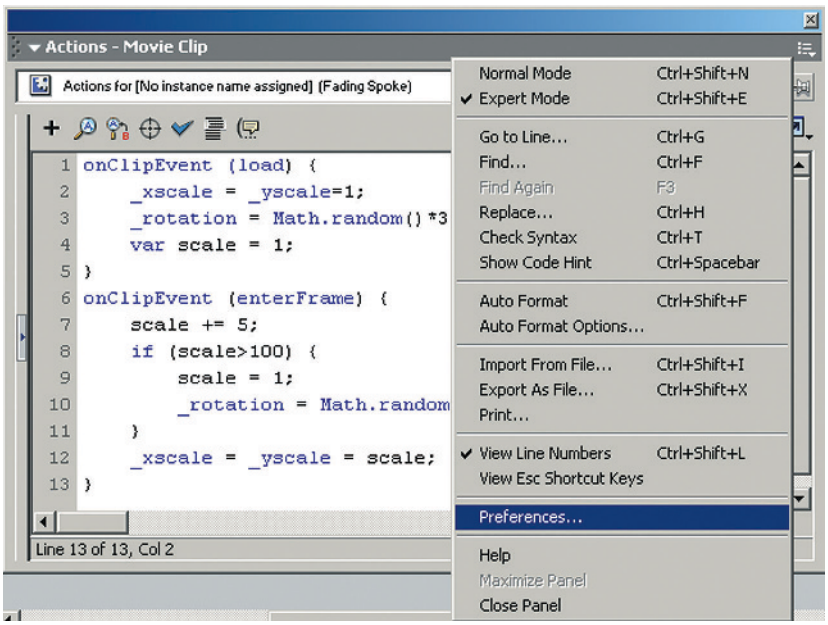
To see the Syntax Coloring options, open the Actions panel options menu (click the drop-down button in the top-right corner of gray bar that contains the label Actions - Movie Clip) and select Preferences (1.14). This opens the Preferences dialog box to the ActionScript Editor tab.

Notice that the top two choices on the Actions panel options menu are Normal Mode and Expert Mode. You can use these menu choices to switch between Actions panel modes.

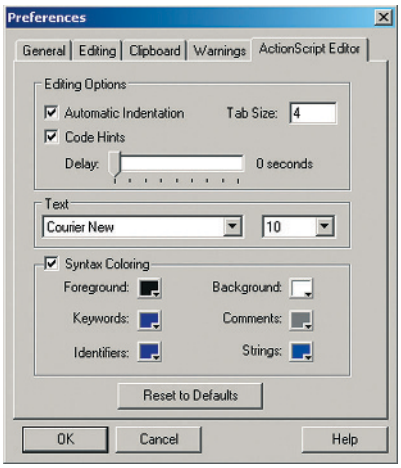
There are six options for Syntax Coloring (1.15). Two of the options are Foreground and Background. The foreground color is the color that you will see for everything in your script that is not a keyword, identifier, comment, or string. The background color is the color of the script window.

Notice that the default color for keywords and identifiers is the same (or very similar). Let's change that. Click the small color swatch next to the Keywords option and change the color to bright red (hex value #FF0000). Next click the small color swatch next to the Identifiers option and change the color to dark green (hex value #009900) (1.16). Click the OK button to apply the changes.

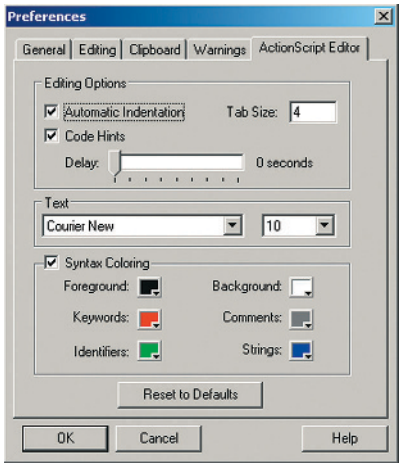
Look at your script in the Actions panel (1.17), and you'll see that two instances of `onClipEvent()` and `if` have changed to red. Red is the color that you specified for keywords, so Flash MX recognizes these words or actions in the script as keywords. Other actions have



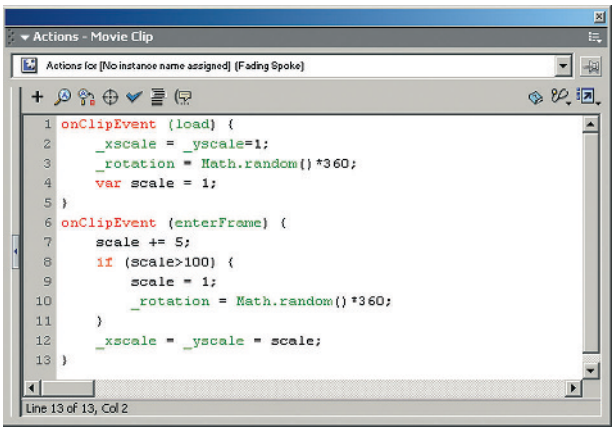
1.14: Select Preferences from the Actions panel options menu.



1.15: The Syntax Coloring default settings



1.16: Change the color for keywords to red and the color for identifiers to green.



1.17: Flash MX changed the colors of portions of the script according to the Preferences dialog box settings.

been changed to green, including `load`, `_xscale`, `_yscale`, `random()`, and so on. Green is the color that you specified for identifiers, so Flash MX recognizes these words or actions in the script as identifiers.

Automatic syntax coloring is useful for several reasons. First, syntax coloring lets you know which parts of the script are keywords, identifiers, comments, or strings. Because you can look through your script according to the colors, you can easily find items. For example, if you want to be able to identify strings in large blocks of scripts, you could change the colors for strings to stand out more so they are easier to hunt down.

Another useful aspect of automatic syntax coloring is that it lets you know if you've gotten the spelling and capitalization right. If actions in your script change color as you enter each line, you know you've written it correctly. Conversely, if they don't change color when they should, you know something is wrong. For instance, say you want to add a comment to your script. If the comment doesn't change colors, you have a visual clue that your comment isn't written properly. (You'll learn about strings and comments in later chapters.)

In this book, you'll be using the default colors. Open the Preferences dialog box again and select Reset to Defaults to return to the default Syntax Coloring settings.

NOTE You may find it helpful to temporarily change the syntax coloring colors to check a script. Just open the Preferences dialog box and set the Syntax Coloring options on the ActionScript Editor tab as you want. When you're finished, revert to the defaults by clicking the Reset to Defaults button in the Preferences dialog box.

Setting Other Actions Panel Preferences

Select Preferences from the Actions panel options menu to open the Preferences dialog box to the ActionScript Editor tab again (1.15). Along with the Syntax Coloring options discussed in the previous section, there are several other useful options for working with the Actions panel.

The Automatic Indentation option refers to the format of your scripts. You might have noticed that Flash automatically indented your code for you, because this option is selected by default. You can change the Tab Size setting (from the default of four spaces) for a smaller or larger indentation.

Indenting your script makes it easier to read and navigate through your code. Indenting is also standard programming convention. For example, code that follows an open curly bracket ({) is indented until the closing curly bracket (}). This shows you at a glance that the indented code is linked together or on the same level.

The Text options let you change the font and font size for the code in the Actions panel. This can be useful for those late nights when your eyes are getting tired. Also, as you learned earlier, Flash can get a little cramped in smaller screen resolutions. If you want to be able to see the Actions panel and more of your stage contents, you might opt to work at a higher screen resolution (if your video card supports it) so that you can see more. The ability to increase the size of the font for the script can come in handy in that case.

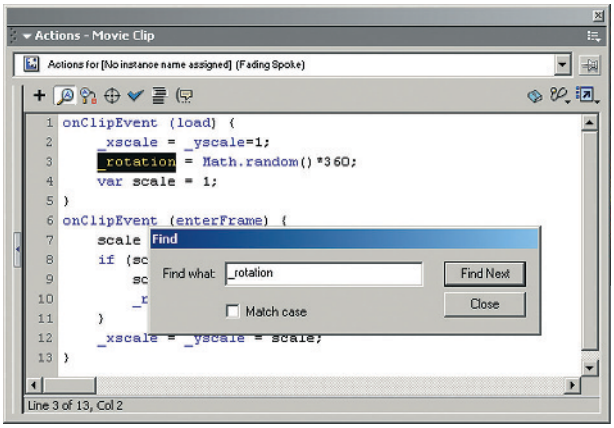
In this book, the examples use the default indentation and font settings, but you may set them according to your own preferences.

Using Find and Replace

There are two more useful features in the Actions panel that should be introduced here: the Find and Replace functions. As you would expect, these options allow you to locate certain code elements and replace them if you desire.

To see how the Find function works, click the Find button (the second one on the left, with the magnifying glass icon) in the Actions panel toolbar to open the Find dialog box. Enter **_rotation** in the Find What field, and then click the Find Next button. The first instance of `_rotation` in the Actions panel will be selected (1.18). If you click the Find Next button again, the next instance of `_rotation` will be selected.

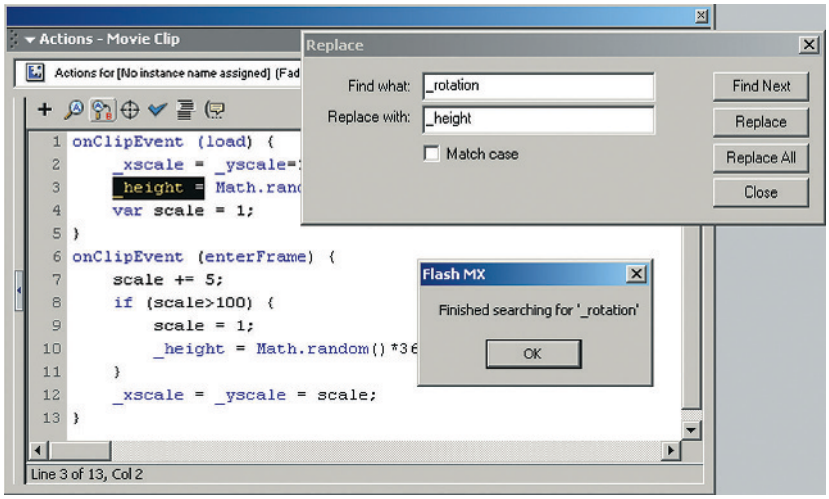
The Find feature is useful when you need to locate a particular item within a big block of code. However, if you want to also fix or change what you're looking for, the Replace feature might be a godsend. Close the Find dialog box and click the Replace button (to the right of the Find button) to open the Replace dialog box. Enter **_rotation** in the Find What field and **_height** in the Replace With field. Now click the Replace All button. Flash will very quickly replace all of the instances of `_rotation` with `_height` (1.19).



1.18: Use the Find feature to find the first instance of the `_rotation` *property.*

The Find Next and Replace buttons in the Replace dialog box allow you to choose when you want to replace instances. You can click Find Next to jump to the next instance of the script snippet. Then click Replace if you determine that you want to replace the instance, or click the Find Next button again to skip it and move onto the next instance.

The Replace feature is useful for several reasons. First, it makes it easier to correct mistakes. For example, let's say you have determined that there is a bug in your latest creation and the reason is that



1.19: Use the Replace feature to swap the instances of `_rotation` *with* `_height`.

you've typed an arbitrary variable that you created using two spellings. For instance, sometimes you typed *`MyVariable`* and other times you typed *`MyVariables`*. You can correct the problem quickly and easily by using the Replace feature.

Another slick thing you can do is use the Replace feature to experiment with scripts you've downloaded over the Internet. For example, you could replace a property with another property and see how it affects the code. That is, in fact, what you did when you replaced `_rotation` with `_height`.

As you become more experienced writing scripts, you'll see how useful the Find and Replace functions can be. For now, close the Replace dialog box and press Cmd/Ctrl+Z (Edit ➢ Undo) to undo the Replace results.

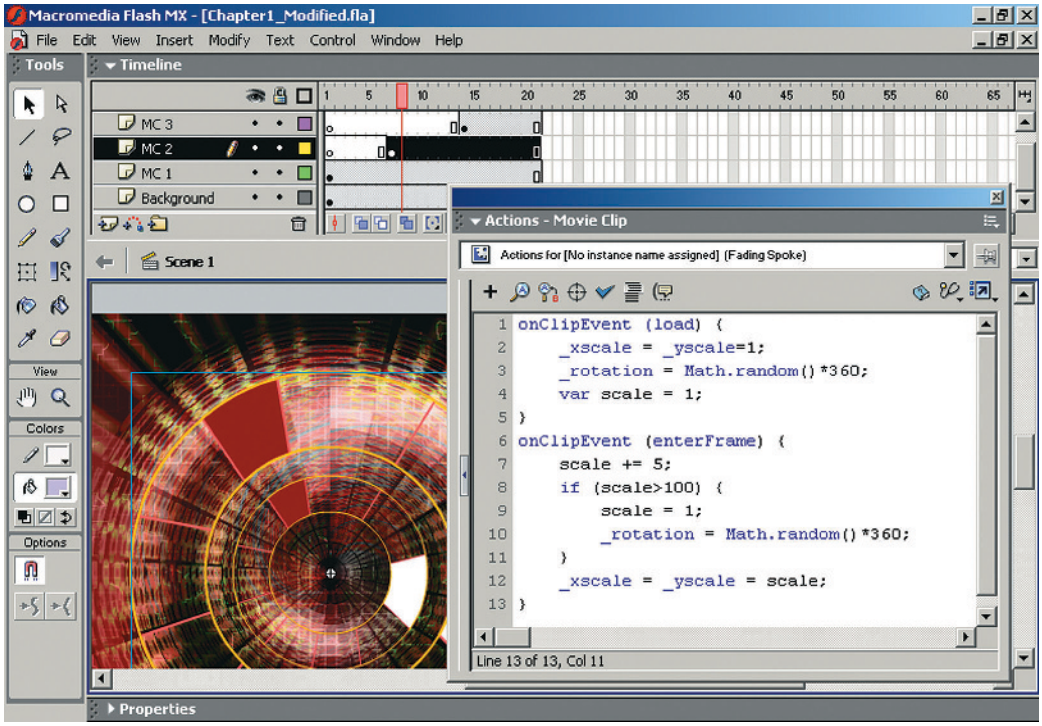
Creating Two Movie Clips

Now that you're familiar with the Actions panel, let's copy the code you've been working on from one movie clip to another. Having two movie clips that use the same code will make it a little easier to learn about how the code works.

1. Copy the Code

The first task is to duplicate the existing code. You'll copy and paste the code on one layer onto another layer.

1. Select the Fading Spoke movie clip that has the ActionScript, and then select the Actions panel.
2. In the Actions panel, press Cmd/Ctrl+A to select all the contents. Then press Cmd/Ctrl+C to copy the code.
3. With the Actions panel still open, select frame 7 on the MC2 layer in the main timeline. Notice that the Actions panel has the label Actions – Frame in the upper left.
4. Click the instance of the Fading Spoke movie clip that's on frame 7 of the MC2 layer. The Actions panel's label now has the label Actions – Movie Clip.



1.20: Copy the code from the Fading Spoke movie clip instance on the MC1 layer to the Fading Spoke movie clip instance on the MC2 layer.

5. Click in the Actions panel and press Cmd/Ctrl+V to paste in the code (1.20).

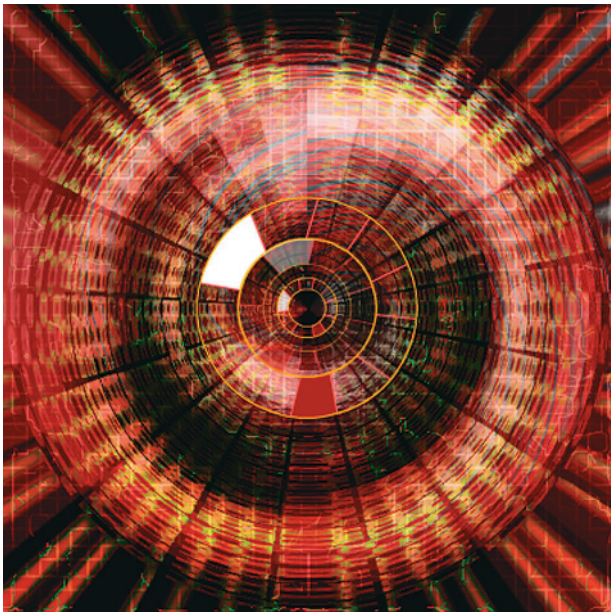
   You've just copied and pasted the code from one movie clip to another. When the movie runs the code, both movie clips will run independently of one another. In other words, the code that is assigned to the Fading Spoke movie clip on the MC1 layer will run independently of the code that is assigned to the instance of the Fading Spoke movie clip that's on the MC2 layer.
6. Paste the code onto the Fading Spoke movie clip instance that is on the MC3 layer (which becomes visible on frame 14).

When you copy and paste code, or when you write new code that you're not sure about, try clicking the Auto Format button on the Actions panel toolbar (the button to the right of the button with a check mark). If your code is not syntactically correct, Flash will display a dialog box with an error message that says, "This script contains syntax errors, so it cannot be Auto Formatted. Fix the errors and try again." This is an easy way to check your code and make sure you haven't missed something like a closing bracket or a semicolon at the end of a line.

7. Press Cmd/Ctrl+Enter to test the movie (1.21).

Now the movie clips rotate randomly, and their sizes incrementally increase, creating a more interesting effect. However, notice that the Fading Spoke animations are concentrated in

the center of the stage. They do not take up the whole stage area. Let's fix this.



1.21: The Fading Spoke movie clips are a little too small for the stage size.

2. Fix the Scaling

To adjust the animation size, you'll edit the code in the movie clip instances. After you make these changes, each movie clip will have slightly different code.

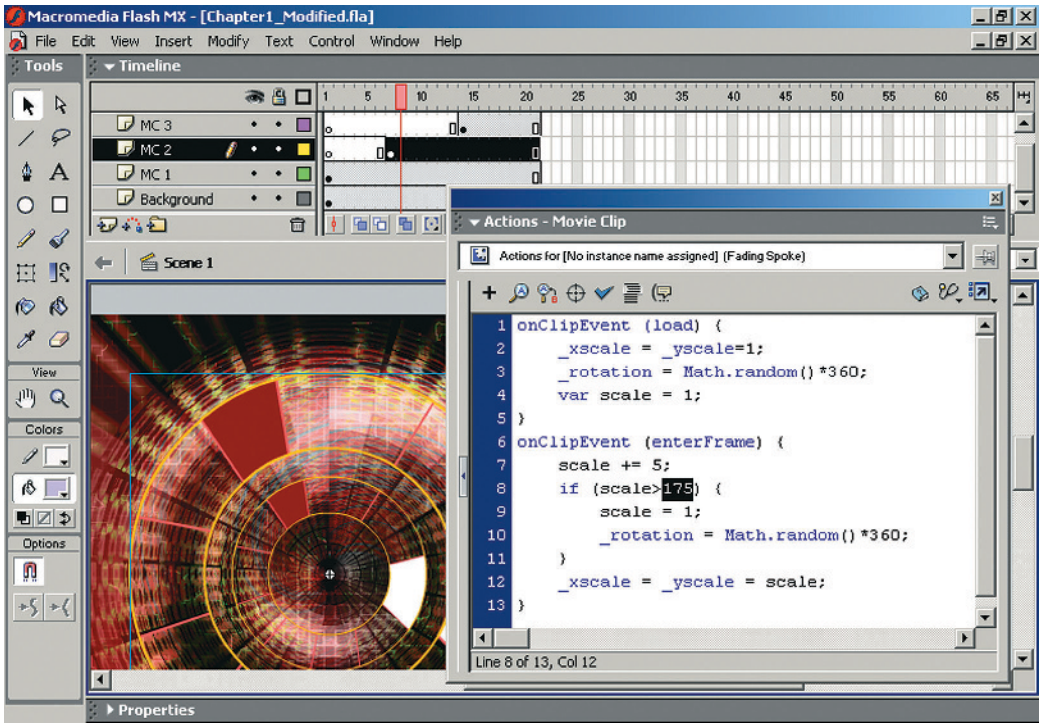
1. Close the Test Movie window.
2. Select the Fading Spoke movie clip instance that appears on frame 7 of the MC2 layer and locate the following line of code:

```
if (scale > 100){
```

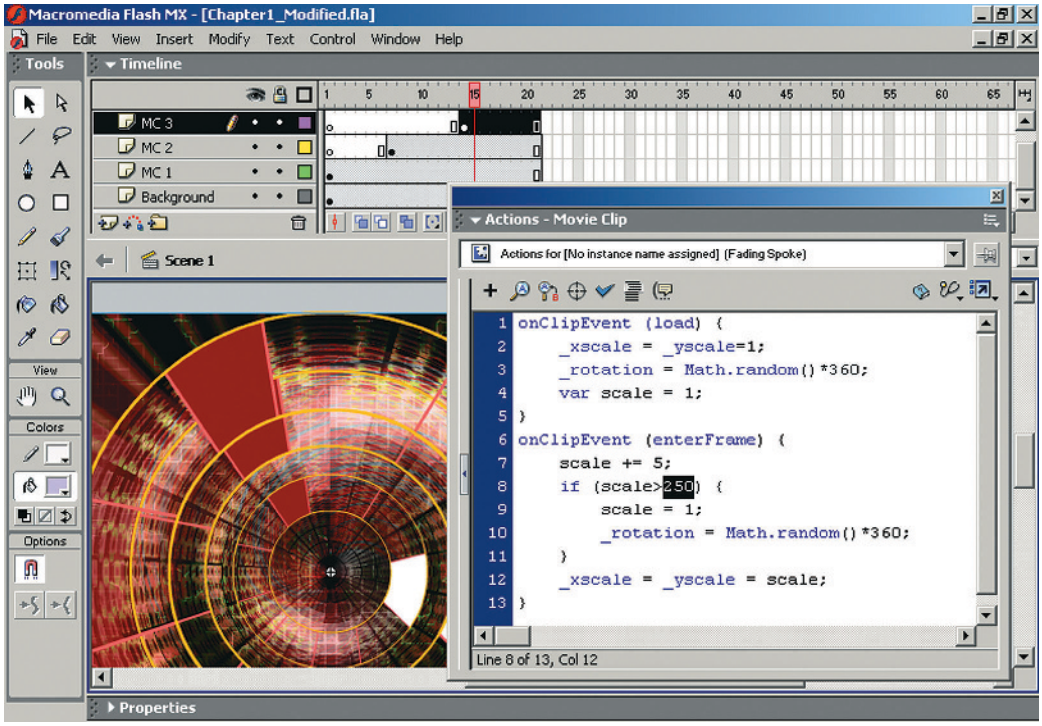
3. Change `100` to `175`, so that the code now looks like this (1.22):

```
if (scale > 175){
```

4. Select the Fading Spoke movie clip instance that appears on frame 14 of the MC3 layer and edit the same line of code (1.23). Change `100` to `250`.
5. Press Cmd/Ctrl+Enter to test the movie (1.24). Now the Fading Spoke movie clip instances fill the pages as they animate.
6. Close the Test movie window.



1.22: On the MC2 layer, increase the number from `100` to `175` within the `if` statement.



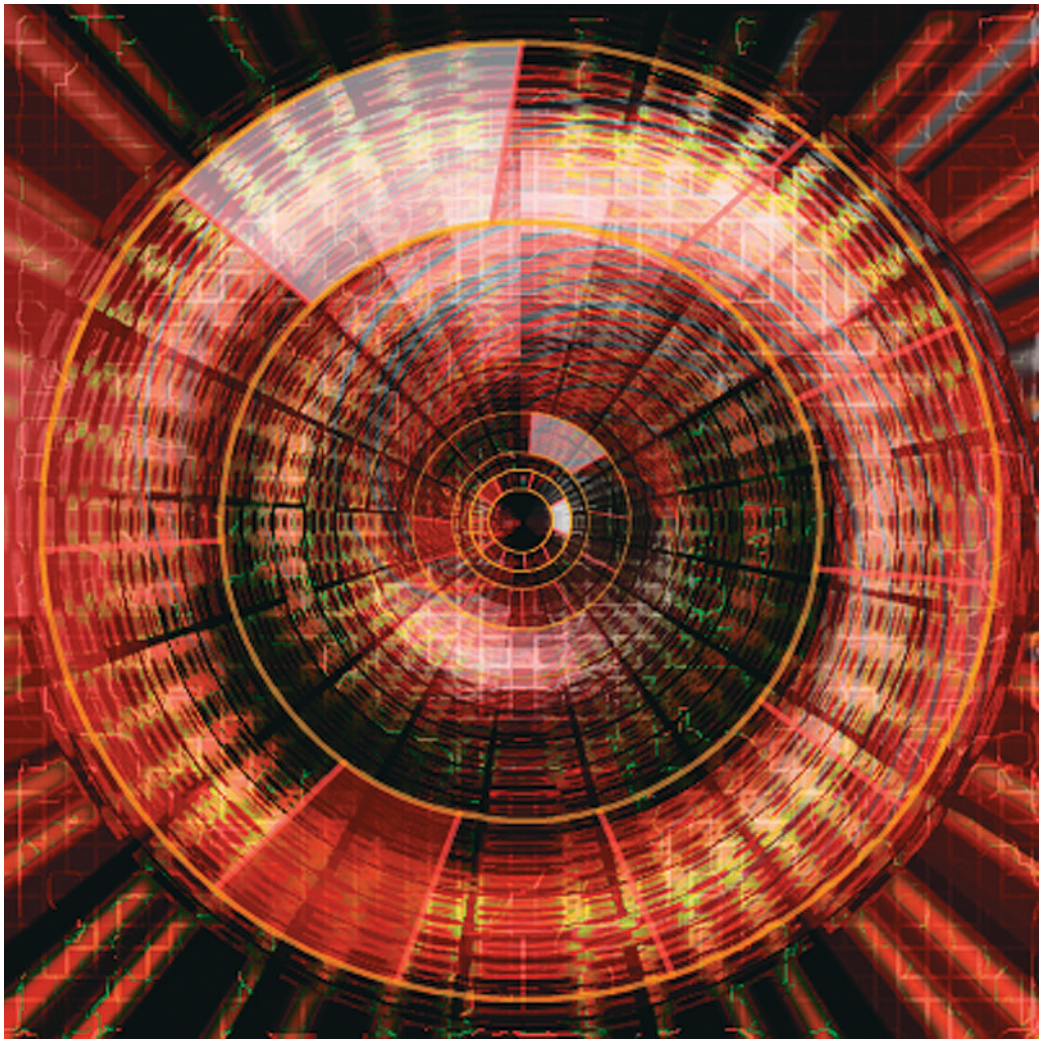
1.23: On the MC3 layer, increase the number from `100` *to* `250` *within the* `if` *statement.*

The main point of this example is to demonstrate how each movie clip uses its own code (it doesn't matter whether it's the same code or different code). This is a simple example of so-called object-oriented code in Flash. Each object acts independently of the other, and the code on each object executes independently of the code on the other objects.

Now let's take a closer look at what the code is doing in this example. There are two main sections of the code: the `onClipEvent(load)` section and the `onClipEvent(enterFrame)` section. The curly brackets after each `onClipEvent()` is where the real action takes place. Curly brackets are used in Flash to designate the beginning and ending of chunks of code. These cannot be used arbitrarily.

The `onClipEvent(load)` is executed when the movie clip is first drawn on the stage, and it is executed only once. Within the `onClipEvent(load)` is the initialization code. Since it only happens once when the movie clip is being loaded, this is where you put anything that needs a value to start with. It is also where you set how you want the movie clip to look when the user first sees it. In this example, you set the scaling of the movie clip to 1%. You also rotate the movie clip about itself by some random amount. This is how the user will first see the Fading Spoke movie clip: very small and rotated.

The `onClipEvent(enterFrame)` code is executed at the frame rate. In this case, the frame rate is 25 frames per second (fps), so this code will be executed 25 times per second. The `onClipEvent(enterFrame)` is where the actual work takes place, because it is repeated over and over. In this example, you tell the Fading Spoke to get bigger, but only up to a certain point. When that certain point has been reached, you reset the Fading Spoke to very small (1%) again, and then rotate it in some other direction.



1.24: The Fading Spoke movie clips instances now scale upward enough to fill the stage.

Conclusion

Flash MX has several built-in features that give you quick access to information on actions. Don't think for a moment that you're somehow cheating or that you're in some way less of a programmer if you use the Reference panel or Show Code Hint button, or even if you select actions from the Actions toolbox. Professional programmers use programming reference guides all of the time. Macromedia has just made it easier for them and you to access the information you need.

Now that you've explored the Actions panel and its tools, you're ready to put ActionScript to use. In the next chapter, you'll learn more about Flash MX's features for controlling movie clips.