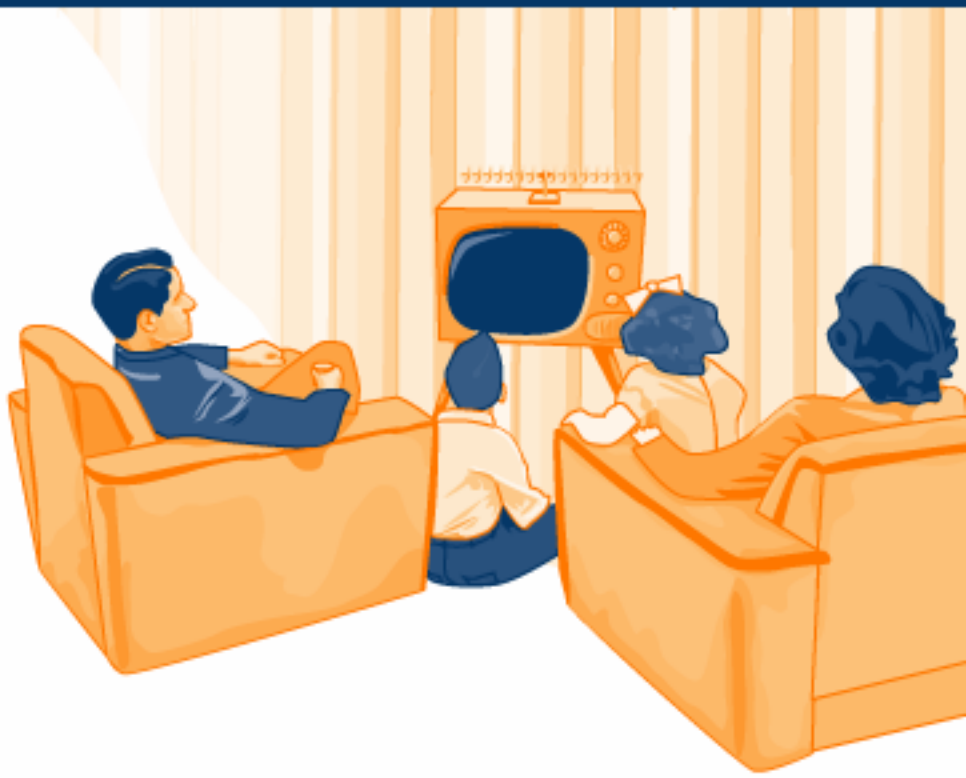
Flash MX 2004

sitepoint

The Flash Anthology

Cool Effects & Practical ActionScript



By Steven Grosvenor

Practical Solutions to Common Problems

The Flash Anthology

Cool Effects & Practical ActionScript

by Steven Grosvenor

The Flash Anthology: Cool Effects & Practical ActionScript

by Steven Grosvenor

Editor: Georgina Laidlaw
Managing Editor: Simon Mackie
Technical Director: Kevin Yank
Cover Design: Julian Carroll
Printing History:
First Edition: July 2004

Expert Reviewer: Oscar Trelles
Technical Editor: Matt Machell
Index Editor: Bill Johncocks

Notice of Rights

Notice of Liability

Trademark Notice

Published by SitePoint Pty. Ltd.

424 Smith Street Collingwood
VIC Australia 3066.

Web: www.sitepoint.com
Email: business@sitepoint.com

ISBN 0–9579218–7–X
Printed and bound in the United States of America

About The Author

Steven Grosvenor is Senior Systems Architect for a Managed Internet Security company in the United Kingdom, and cofounder of phireworx.com, a Fireworks resource site.

Steven is a contributing author of *Fireworks MX Magic* (New Riders), *Special Edition Using Fireworks MX* (Que), and *Fireworks MX Fundamentals* (New Riders). He has also authored numerous articles on the Macromedia Developer/Designer Center, and a variety of magazine pieces.

Endeavoring to juggle a hectic home life and work schedule, while somehow keeping on top of the latest advances in technology, he nevertheless finds time to maintain *Go Flash, Go!*: SitePoint's Flash Blog.

About The Expert Reviewer

Oscar Trelles is an interaction designer based in New York and an active member of the Flash community. An independent consultant, he has a particular interest in projects that allow scope for innovation and improvement of the user experience, as well as the practice of scalable Flash and Web development. Oscar runs a blog at http://www.oscartrelles.com/, where visitors discuss, among other things, Flash development.

About The Technical Editor

A man of many talents, Matt Machell has been a Web designer, technical editor, writer, and jewelry picker. He is currently contracting in the higher education sector, producing Websites for research centers. He likes music with loud guitars and games with obscure rules. He lives in Birmingham, UK, with his girlfriend, Frances, and a horde of spider plants.

About The Technical Director

As Technical Director for SitePoint, Kevin Yank oversees all of its technical publications—books, articles, newsletters and blogs. He has written over 50 articles for SitePoint on technologies including PHP, XML, ASP.NET, Java, JavaScript and CSS, but is perhaps best known for his book, *Build Your Own Database Driven Website Using PHP & MySQL*, also from SitePoint. Kevin now lives in Melbourne, Australia. In his spare time he enjoys flying light aircraft and learning the fine art of improvised acting. Go you big red fire engine!

About SitePoint

SitePoint specializes in publishing fun, practical and easy-to-understand content for Web Professionals. Visit http://www.sitepoint.com/ to access our books, newsletters, articles, and community forums.

To my late father, Frank, who gave me such insight into life and made me the person I am today; I am forever indebted to you.

To my beautiful wife, Paula, who has put up with endless nights and weekends while I toiled over this book, and my children: Verity, Alexander, and Eleanor, who had the patience of little angels. I thank you all for the various ways in which you helped me.

Table of Contents

Preface

Gone are the days when you could satisfy your clients with the creation of simple Flash effects using the timeline. Things have changed… a lot! Basic animated tweens aren't enough any longer—people expect more now, from scalable and reusable scripted animation, to external data interaction.

The problem for most fledgling developers is that, although they may know what they want to do, they don't know how to do it. Perhaps they see the ActionScript Reference Panel and Actions Panel and think, "Whoa, I'm not going near that!" This is where Flash turns most people off. New users tend to either keep to the fringes and miss out on the real power, or they leave the application alone completely. But, it needn't be that way. The use of ActionScript to create effects needn't be an overwhelming experience. In fact, from animation to video, ActionScript can help you achieve your wildest Flash goals. The process *can* be easy, provided the code's broken into manageable, bite-sized chunks.

Flash is about exploration. As you move through the examples in this book, you'll find new applications for each area we explore. With a little imagination and some more ActionScript, you'll have everything you need to create engaging, compelling effects. There's simply no need to shy away from scripted projects. Whether you're a developer or a designer, Flash has something for all levels of ability.

As I walk you though the world of scripted effects and techniques, you'll see exactly what's involved in creating each of the examples offered here. We'll pull apart the code and discuss modifications and ideas you can use to extend each of the effects we consider. These examples aren't dead-end effects created for their own sake. Each one has a variety of potential applications and deserves a place in your Flash arsenal. Soon, you'll be creating first class projects, surprised by how quickly your skills develop as you explore new possibilities and push the boundaries of your knowledge. So, buckle up and get ready to ride the Flash Anthology roller coaster!

Who Should Read This Book?

This book is aimed at beginning to intermediate Flash developers and designers who want to expand their understanding of Flash. The examples are formulated to give you a more comprehensive grasp of Flash's capabilities, encouraging you

to employ scripted techniques to create scalable, impressive effects in real-world situations.

If you understand JavaScript, you should find ActionScript a natural step. If you're not familiar with either technology, don't worry! You'll soon pick up the ActionScript syntax and logic as we analyze each example.

After reading a few chapters of this book, you'll probably be comfortable enough to start creating your own modifications to the techniques we discuss. By the end of the book, you'll realize just how many different possibilities there are for Flash in your Web projects. I hope you'll be inspired to apply and experiment with the technology as you continue to develop your skills.

What's in This Book?

There's no need to read the chapters in this book in the order in which they're presented—feel free to dip into whichever interest you. Of course, you can just as easily follow the book from start to finish for a well-rounded picture of Flash and its capabilities.

Chapter 1: ***Flash Essentials***

If you're new to Flash, this chapter will give you a solid grounding in the program's interface, as well as a fundamental understanding of the ActionScript dot notation. Tips and techniques for working with ActionScript and the timeline are also included. Finally, I'll walk you through a few organizational guidelines.

Chapter 2: ***Navigation Systems***

We jump straight into some nifty examples in Chapter 2, which focuses on navigation effects. We start by asking, "What makes an effective navigation system?" We review the planning of common navigation methods, then move on to examples of horizontal, vertical, gadget-based, and advanced systems. If you've ever wanted to build compelling Flash navigation, look no further than this chapter.

Chapter 3: ***Animation Effects***

The question of whether to use timeline-based or scripted animation is one every Flash developer asks at some point. We'll explore the principles behind these effects and attempt to put the timeline vs. script debate to rest once and for all. The basic building blocks provided in this chapter will have you creating stylish animations in no time!

Chapter 4: *Text Effects*

Text effects are part of every top Flash designer's repertoire. A thoughtfully conceived and carefully applied text effect can bring life and interest to an otherwise bland interface. This chapter explores these effects in full, discussing when and how they should be applied, and providing a variety of examples to help you build everything from simple animations, to advanced, three-dimensional text productions.

Chapter 5: *Sound Effects*

One of the most underemployed, yet effective additions to a project is sound. Appropriate sound effects can enhance the impact of movement, provide feedback to users interaction, and build atmosphere. In this chapter, we analyze when sound should be used, how to choose the right sound clip for the job, and how clips can be imported and exported easily. We explore volume and panning, then build a mini sound player, random track sequencer, and much more.

Chapter 6: *Video*

If you think video should be used only in DVD and similar presentations, think again! Here, you'll discover tips and techniques for video importing and exporting, for capturing and compression, and for being creative with video in Flash. You don't have to use full-frame video; we prove this point by harnessing the power of Flash's built-in components to produce effective and subtle video effects. If you always wanted to use video in your projects, but never knew how or when, you need look no further than this chapter for information and inspiration.

Chapter 7: *Flash Forms*

Flash isn't just about fancy animation and clever navigation—it also gives you the power to create some pretty amazing form-based applications. Flash allows you to achieve everything that can be done with HTML-based forms… and then some! In this chapter, we cover form function and data validation, and explore various methods for handling the data received through the forms we build. From there, we create several complete applications, using a variety of built-in components and validation techniques to produce functional, snazzy, and robust Flash forms.

Chapter 8: *External Data*

As a Flash developer, you'll often need to use dynamic data in your projects. There are many facets to working with external data, and this chapter covers several common examples. We'll import data from an external text file in an application that resembles Post-It Notes, interface with SQL Server 2000

through a Blog reading utility, and discuss the storage of user preferences using a Local Shared Object. Whatever your data manipulation requirements, you'll acquire the building blocks you need to get the job done!

Chapter 9: ***Debugging***

If your application is misbehaving and you don't know why, or you simply would like to watch your program as it progresses through complex function calls, this is the chapter for you. You'll learn to analyze what happens when good applications go bad, develop an understanding of simple, syntax-based bugs, and examine those nasty event-based problems that are so difficult to track. I'll explain the details of debugging Flash applications using the Debugger Panel and the `Error` class. We also review the `trace` function used to ensure that functions do what they're supposed to. If you have problems with Flash projects and aren't sure why, a read through this material will help you resolve them.

Chapter 10: ***Miscellaneous Effects***

This chapter sounds like a mixed bag, and it certainly is! Chapter 10 covers everything that can't be pigeon-holed within the other chapters. You'll find some fantastic examples involving CSS, graphing with commercial components, and search engine optimization.

The Book's Website

Located at http://www.sitepoint.com/books/flashant1/, the Website that supports this book will give you access to the following facilities:

The Code Archive

As you progress through this book, you'll note a number of references to the code archive. This is a downloadable ZIP archive that contains all of the finished examples and source files presented in this book. Simply click the Code Archive link on the book's Website to download it.

The archive contains one folder for each chapter of the book, with both the source `.fla` files as well as the compiled `.swf` files (for quickly previewing the various effects). Wherever possible, the example files have been saved in Flash MX format for the benefit of readers who may not yet have Flash MX 2004. Of course, many of the later examples use features specific to the new version, and will therefore require Flash MX 2004 to open.

Updates and Errata

No book is error-free, and attentive readers will no doubt spot at least one or two mistakes in this one. The Errata page on the book's Website will provide the latest information about known typographical and code errors, and will offer necessary updates for new releases of Flash.

The SitePoint Forums

If you'd like to communicate with me or anyone else on the SitePoint publishing team about this book, you should join SitePoint's online community[2]. The Flash and ActionScript forum[3], in particular, offers an abundance of information above and beyond the material in this book.

In fact, you should join that community even if you *don't* want to talk to us. There are a lot of fun and experienced Web designers and developers hanging out there. It's a good way to learn new stuff, get questions answered in a hurry, and just have a good time.

The SitePoint Newsletters

In addition to books like this one, SitePoint publishes free email newsletters including *The SitePoint Tribune* and *The SitePoint Tech Times*. Reading them will keep you up to date on the latest news, product releases, trends, tips, and techniques for all aspects of Web development. If nothing else, you'll get useful Flash articles and tips. If you're interested in learning about other technologies, you'll find them especially valuable. Sign up for one or more of SitePoint's newsletters at http://www.sitepoint.com/newsletter/.

Your Feedback

If you can't find your answer through the forums, or if you wish to contact us for any other reason, the best place to write is `<books@sitepoint.com>`. We have an email support system set up to track your inquiries. If our support staff members can't answer your question, they'll send it straight to me. Suggestions

[2] http://www.sitepointforums.com/
[3] http://www.sitepoint.com/forums/forumdisplay.php?f=150

for improvements as well as notices of any mistakes you may find are especially welcome.

Acknowledgements

I'd like to thank the following people, in no particular order, for their help during the development of this book.

A big thank you to Oscar Trelles for his refreshing outlook and alternative viewpoints; to Matt Machell for his input and last minute alterations; to Georgina Laidlaw for correcting my caffeine-induced grammatical and spelling mistakes; to Simon Mackie for steering the ship in the right direction at all times over the past few months; and to SitePoint for investing the time in helping me to develop this book. Thanks also to the many other people who have helped during the development of the book—you know who you are!

3

Animation Effects

The fads of Internet design may come and go, but one thing that will never change is that the more dynamic you make your Flash creations, the more engaging they are to the user. This dynamism can be counterproductive under certain circumstances, especially when multiple effects battle for the user's attention, or the effects are too garish. Identifying the key to effective animation is like the search for the Holy Grail. What some users think is a cool effect, others find patently uninteresting—and vice versa. Never lose sight of the importance of striking a balance between the interface and the animations you're attempting to produce.

Flash has always had as its nucleus animation and motion. This is, after all, what Flash was originally created for—the animation of objects over time. As new versions are released and the technology evolves, so do the capabilities of Flash's scripting language. What we could once achieve only with keyframes and tweening can now be accomplished in a few lines of ActionScript. Some developers find this reality difficult to grasp, but as we saw in Chapter 2, once you understand the basics, you can build on them with new experiments.

With very few exceptions, what can be done with keyframe tweening can also be achieved through ActionScript. But, what are the advantages of scripting? The answer's simple: portability, scalability, and manageability. You can affect an animation dramatically by tweaking an equation or a few variables in its ActionScript. This process is much easier than laboriously editing motion tweens, which can sometimes appear in their hundreds in large animated effects.

This doesn't mean that motion tweening is dead, however—not by a long shot. If you create simple motion tweens (for example, an effect that shows an object increasing in size), then script the effect multiple times and experiment with it via ActionScript, you can create some pretty amazing effects with a minimum of effort.

With Flash MX 2004, and the introduction of Timeline Effects, creating these motion tween building blocks takes even less work than it did before, as we'll see in the coming chapter. I'll give you the information you need to develop both classic effects you can be proud of and exciting new animations. You'll also learn the techniques involved in creating innovative Timeline Effects. It's virtually all ActionScript from here on, so have your calculator and pencil ready!

Animation Principles

If you are reading this book, then I can be pretty sure you have a copy of Flash MX or later. You probably purchased Flash because of the animation capabilities that lie at its heart. In the most basic form of Flash animation, we can smoothly transition an object's location or shape from point/shape A to point/shape B by altering the properties of that object at keyframes within the timeline. This used to be a cumbersome process in previous versions of Flash, but it's more accessible now. With a solid understanding of ActionScript and the dynamics of motion you can rapidly create animation effects that would have taken many hours to create with previous versions.

Hit the books!

What did you do with your old Physics and Math textbooks when you left school? Did you throw them away? Shame on you if you did—they can be an invaluable source of inspiration for creating mathematical and motion-related scripted animations in Flash. I'm a bit of a hoarder, which probably explains why I've still got mine!

There are many uses for Flash in creating animation. Perhaps you want to create a straightforward animation that moves an object from point A to point B. Maybe you're itching to build a more complex animation with a "real world" feel, easing objects into position or having them exhibit elastic characteristics. Both simple and advanced animations are possible in Flash via different methods: by hand, using complex keyframes and motion tweening, or with the help of ActionScript.

While the ActionScript method of animation may initially appear difficult, once you become comfortable with its methodologies for movement and learn the

nuances of its quick, effective methods, you'll soon be creating increasingly complex animations and building on your existing knowledge. If this is your first experience with ActionScript, you'll soon be surprised how easy it is to create scripted animation. This should inspire you to explore your own ideas and experiments, and take ActionScript to the limit.

Animation Overload

The ability to easily include animation techniques and effects within Flash movies is usually the reason people use this technology for animation development. However, inexperienced users may succumb to "animation rage," as they become a little *too* carried away with the power Flash puts at their fingertips. Over-the-top animation effects are the result—effects that, upon careless replication within the same movie, succeed only in creating an unpleasant experience, to say the very least!

It's easy to become trigger-happy and animate every element of your display, but this approach is a recipe for disaster. The effect that you set out to create will soon be lost, overwhelmed by all the others that surround it.

The key to an effective animation lies in maintaining a balance between the message you're trying to convey and what's happening on the screen. Even tipping the balance slightly can ruin your effect, so adopt the following guidelines as rules of thumb for creating successful animations:

Tame the Animation
: "Because you can" is not a good enough reason to animate something. Users tend to identify excessive animation as the mark of the amateur—though your site will certainly make an impression, it won't be a good one!

Err on the Side of Subtlety
: Effects that are exaggerated or garish will annoy users, especially if the animation is part of the main interface or navigation. Strive to create effects that are pleasing to the eye, not intrusive.

Consider the User
: Try to distance yourself from any effect you create; imagine you're a user viewing it for the first time. If you think it's "too much," then it probably is. If you don't like it, your users won't, either. Of course, you can't please all of the people all of the time, so try to strike a happy medium at which most visitors will be satisfied.

Stand back!

When previewing your movie, try standing several feet from the monitor. Believe it or not, this gives you a clear sense of the animation's movement across the screen. If you're too lazy to walk to the other side of the room, try squinting so that the screen blurs a little. You'll be able to detect the movement on the screen without the distracting details, which will help you identify whether the movie is over-animated.

Be Conservative with Animations

Yes, you can create cool animations with ActionScript, but you shouldn't include them all in one page, interface, or effect. You may lose focus by adding too many other animations to your design. Try to sprinkle animations through your designs, rather than deluging the user with an animation storm.

To Tween or Not to Tween?

A few years ago, Flash developers had no choice. To create animated effects in Flash, we used keyframes and motion or shape tweening. Now, we have the luxury of choice; we can script the motion, or create it via the traditional route. Both methods deliver benefits, as we'll see shortly, when we compare scripted animation with traditional tweening methods.

One point worth noting, however, is that with motion scripting, the entire movie need not be any longer than a single frame. The ActionScript, not the timeline, controls the animation, allowing for well-organized movies with uncomplicated structures.

Let's take a look at a simple animation technique with which you might already be familiar: linear motion. The most basic effect that you can create is movement from one point to another and, indeed, this may have been one of the effects you tried when you first opened Flash. Let's revisit it now.

Timeline Animation Example

It's easy to create this effect on the timeline. Let's walk through the steps involved.

1. Draw a simple shape (like a circle) on the stage and convert it to a movie clip symbol named `Timeline_Animation`. Position the symbol instance on the stage at (0, 0).

2. Select frame 10 within the main timeline, right-click, and select Insert Key-Frame (**F6**). Notice that the movie clip instance is copied into the new key-frame.

3. Select the instance of the movie clip in frame 10, and move it to (100, 100).

4. Select frame 1, right-click, and select Create Motion Tween.

Preview your movie. You've created a simple animation that moves your clip from one point to another. This is a simple animation; if the effect were more complicated, the timeline could quickly become messy and difficult to work with.

Creating simple motion using the timeline in this manner can also be accomplished within Flash MX 2004 and later versions via Timeline Effects (more on this in Chapter 4).

ActionScripted Animation Example

Let's take another look at this animation, but this time, let's build it in Action-Script.

1. Draw a simple shape (like a circle) on the stage and convert it to a movie clip symbol named `Scripted_Animation`. Position the symbol instance on the stage at (0, 0), and name the instance `scripted_animation`.

2. With the Actions Panel open and the first frame of the main timeline selected, add the following code:

```
var endX = scripted_animation._x + 100;
var endY = scripted_animation._y + 100;
var stepX = (endX - scripted_animation._x) / 10;
var stepY = (endY - scripted_animation._y) / 10;

scripted_animation.onEnterFrame = function ()
{
  if (this._x < endX) this._x += stepX;
  if (this._y < endY) this._y += stepY;
};
```

First, we set variables for the x and y endpoints (`endX` and `endY`) to equal the starting coordinates plus 100 pixels along each axis. We then use these values to calculate how much the object will have to move per frame along each axis (`stepX` and `stepY`) to reach its destination in ten frames. We then introduce an event

handler that moves the object along the two axes by the calculated distances until it reaches its destination.

This code takes the previous example a step further, though, because you can place this movie clip anywhere on the stage. Regardless of its starting location, the clip will move 100 pixels along each axis from its starting position.

You may be looking for more code to complete the effect, but that's it! Simple, isn't it? Of course, ActionScript becomes more complicated as you add more interesting effects, but this method certainly saves a lot of clutter on the timeline.

Animations built using the timeline and motion tweening are useful for testing and for implementation as part of a larger animation (for example, creating simple rotation for a loading animation). The real benefits of developing animations with ActionScript are scalability and the opportunity for dynamic movement in response to user input or other variables.

Once you start animating with ActionScript, it's difficult to stop—this method really does act as a springboard for your creativity. And, don't forget to save your experimental FLA files even if you don't use them straight away. You never know when you might need them!

Creating Function Libraries

Once the ActionScript bug has bitten you, you'll be infected permanently, and there's no known antidote! You'll create many FLAs over time, and will no doubt build up your own core set of scripts and methods. But, rather than reinventing the wheel every time you need to carry out a particular function, why not save your scripts in `.as` (ActionScript) files? These files can then be included dynamically in your creations as you need them.

I maintain a core of scripts that I've created over the past few years, and which I back up regularly. I'm always careful to sort my ActionScript files into a logical folder structure. That way, when I start a new project, I can go and grab my existing script files without any hassle.

Any scripts that are still in development, or that I haven't had time to finish, I place in a file called `unfinished.as`. This way, I don't lose the code or accidentally delete it, and I can come back to it later to finish or develop it further.

Hotmail for backups

If I lost all of my code snippets, I'd be very unhappy! And, even though I perform regular backups, I can never be sure of their integrity. For this reason, I set up a free mail account with Hotmail, and created an archive folder. Now, every month, I mail myself a ZIP archive of my `.as` files. This may seem a little extreme, but if you've ever lost your work in a hard drive or backup failure, you'll understand why I go to such lengths to protect my code.

Creating a Simple Function Library

A simple animation library can help you clean up your timeline and make things more manageable. To create your own library, follow these steps, or simply locate `Simple_Motion.fla` and `Simple_Motion.as` in the code archive:

1. Look at the code from the ActionScript animation example you completed above; specifically, look at the `onEnterFrame` event handler. We can write a function that does the same job for a specified `clip`, given `stepX`, `stepY`, `endX`, and `endY` values:

   File: **Simple_Motion.as**

   ```
   function SimpleMovement (stepX, stepY, endX, endY, clip)
   {
     if (clip._x < endX) clip._x += stepX;
     if (clip._y < endY) clip._y += stepY;
   }
   ```

   The structure of the `SimpleMovement` function is similar to the event handler, except that it accepts parameters to tell it exactly what to do (and what clip to do it to), instead of relying on predefined variables.

   Type the code for this function into a text editor (e.g., Notepad on PC, or BBEdit on Mac) and save it as `Simple_Motion.as`.

2. To use this file, add the following line of ActionScript to the root of any movie, in the first frame

   File: **Simple_Motion.fla** Actions : 1 (excerpt)

   ```
   #include "Simple_Motion.as"
   ```

This compiles the code from the `Simple_Motion.as` file into the SWF file when it is created, providing access to the `SimpleMovement` function we created above.

3. Alter the `onEnterFrame` event handler to use the imported function as follows:

File: **Simple_Motion.fla** Actions : 1 (excerpt)

```
scripted_animation.onEnterFrame = function ()
{
  SimpleMovement(stepX, stepY, endX, EndY, this);
};
```

Here, we've created a simple function call, passing the four variables defined on the root of the timeline, as well as the movie clip we wish to animate.

4. Preview you movie in Flash, and you'll see it works exactly as before.

Including the function in another project is as simple as saving the `.as` file to the directory containing the FLA you're working on, and adding the `#include` directive to the project. You can then use the function as often as you like.

Creating Master Libraries

When you're working on a series of projects that share a similar theme, you may find they also share bitmaps and vector and sound objects. If you've forgotten which FLA these shared objects reside in, you're left to choose between a time-consuming search or laborious replication.

To avoid this situation, I create what I call **master libraries** for my buttons, movie clips, and animations, which I name according to their content. For example, I might create an FLA file that contains all plastic- or glossy-looking buttons, and call it `Buttons - Plastic_Gloss.fla`. I would then save this in a master directory. When I need them, I simply select File > Import > Import to Library..., locate my FLA file and, presto! The buttons appear in the Library Panel for use in the current project.

Even after several months, you may come back to a project to enhance it or add extra functionality. If you can't remember where the source FLA files are, you're going to waste a lot of time. Using this procedure allows you to be smart with your time and resources, and maintain a consistent look and feel across projects.

I think that, by now, we've covered most of the best practices and methods for increasing productivity when you work with Flash. The practices I've outlined

here are only guidelines to make your life a little easier; they're not hard and fast rules. So, feel free to embrace as many or as few of them as you wish.

Now it's time again to "holster up" and get ready for a showdown with some very cool ActionScripted effects!

Random Motion

Have you ever wanted to create random movement for an object or a number of objects? There's a simple technique that will take a single movie clip, create many copies of the object, and randomly place these on the canvas. It then creates the illusion of constant random movement. Best of all, this technique is easily extensible, allowing you, for example, to dynamically alter many of the properties of the object, including opacity and scale.

If you'd like to see the finished product before you proceed, have a look at `Random_Motion.fla` in the code archive.

Setting the Scene

First, you'll need to create a new Flash movie to showcase your effect.

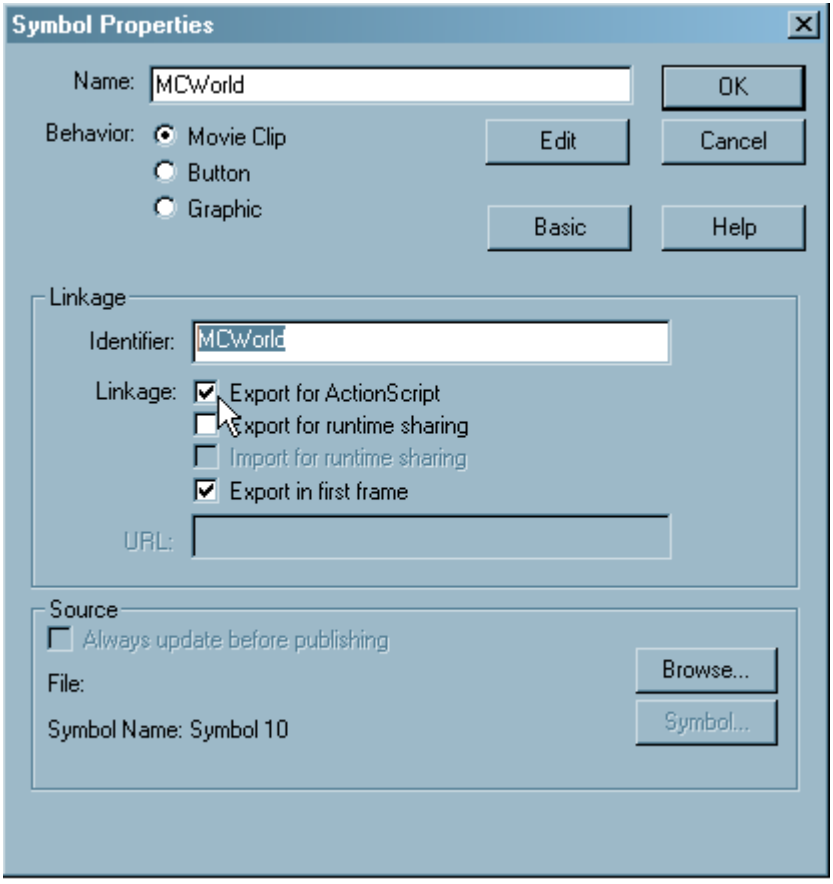
1. Select File > New to create a new Flash movie.

2. Select Modify > Document and set both the width and height of the movie to 300 pixels.

In order to randomly place and move copies of your object, you'll first need an object to use. In this example, we'll create a movie clip container named `MCWorld`, which will contain another movie clip, called `World`.

3. Select Insert > New Symbol, select Movie clip, and name the clip `MCWorld`. Click on the Advanced button to view the clip's linkage parameters, select Export for ActionScript, and name the identifier `MCWorld`, as shown in Figure 3.1.

Figure 3.1. Set linkage properties for the parent movie clip.

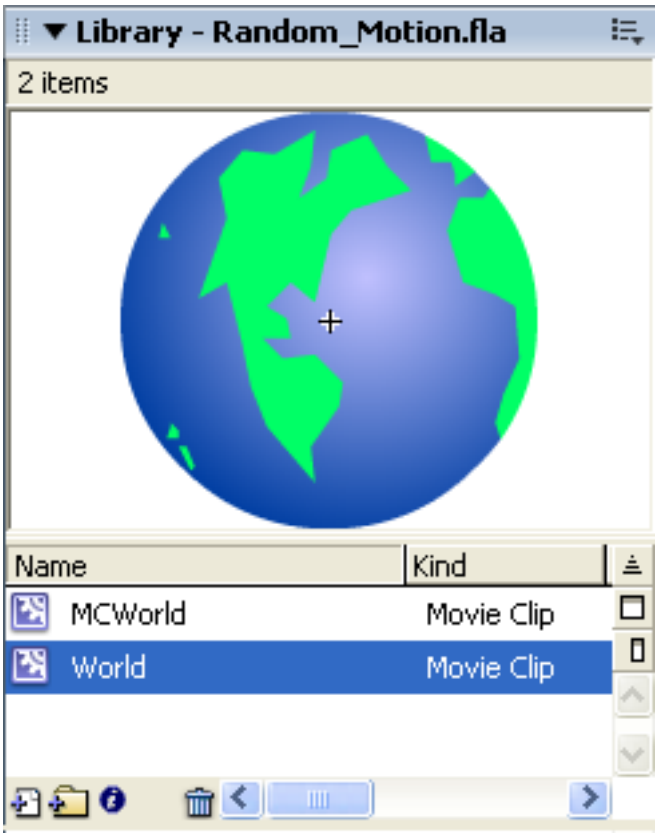


We select the Export for ActionScript button because, in a moment, we'll use ActionScript dynamically to create instances of this clip. To do that, we need to make it available to ActionScript by choosing this option and assigning the clip a unique identifier.

4. Create a graphic symbol that contains the object or image to which you want to assign random movement. Select Insert > New Symbol... and choose Graphic. Name this symbol `World`, then create the object either with the drawing tools, or by importing an image or other object from an external source.

The Library Panel should now contain a graphic symbol named `World`, as in Figure 3.2.

Figure 3.2. Add the child movie clip.



5. Double click the `MCWorld` movie clip to open it. Drag an instance of the `World` symbol into it and name the instance `World`.

Adding the ActionScript

Great! We've created a clip called `MCWorld` that contains a graphic called `World`. Now, we can begin to add the ActionScript that will control what takes place on the stage:

6. Select Layer 1 within the main timeline, and give it the name ActionScript. Expand the Actions Panel (select Window > Development Panels > Actions or press **F9**).

7. Add the following code within the Actions Panel. This code creates thirty instances of the `MCWorld` clip and places them on the canvas at random. It also randomly alters the clips' opacity.

File: **Random_Motion.fla** ActionScript : 1

```
var numObjects = 30;
for (i = 0; i < numObjects; i++)
{
  var randomObject = attachMovie ('MCWorld', 'MCWorld' + i, i);
  randomObject._x = random (300);
  randomObject._y = random (300);
  randomObject._alpha = random (100);
}
```

The key here is the `attachMovie` method, which lets you add a new movie clip to the current movie. The parameters we pass to this method are: the identifier we gave the clip in the library (`MCWorld`), a unique name for each clip instance (in this case, `MCWorld` with a number appended to it), and a number that indicates where to place the clip in the stacking order.

Also of note in this code is the `random` function, which returns a random integer between zero (inclusive) and the specified number (exclusive). So `random (300)` returns a number from 0 to 299. We use this function to generate the position on the stage and the opacity for each instance we create.

With our stage filled with randomly-positioned graphics, it's now time to move them around.

8. Double-click the `MCWorld` movie clip in the Library Panel to open it. Select Layer 1 and rename it ActionScript.

9. Add the following code within the Actions Panel. It uses `setInterval` (a standard JavaScript function) to move the graphic symbol instance (`World`) to a new random position within two pixels of its current position every tenth of a second.

File: **Random_Motion.fla** MCWorld : ActionScript : 1

```
setInterval (Randomizer, 100);
function ()
```

```
{
  var xShift = random (5) - 2;
  var yShift = random (5) - 2;
  World._x += xShift;
  World._y += yShift;
}
```

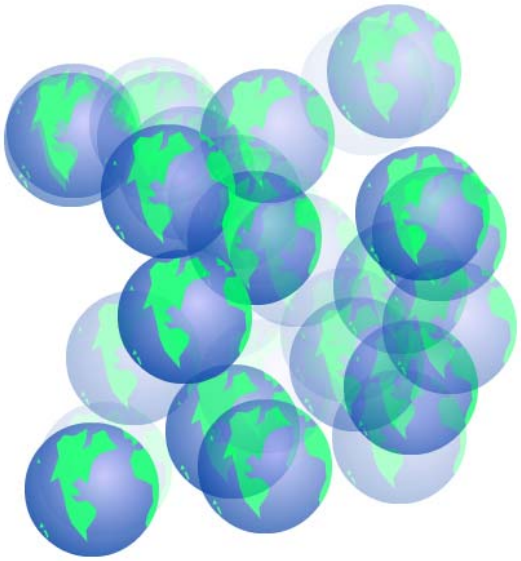
We could instead have used an `onEnterFrame` event handler to do this, but `setInterval` allows us to define the speed of the motion independent of the movie's frame rate, which can be useful in many circumstances.

Testing the Movie

You've created your movie clips; now, let's take the movie for a test-drive.

10. Select Control > Test Movie to preview your movie.

Figure 3.3. Preview the movie within Flash.



If everything has gone according to plan, you should see an animated random movie similar to the one shown in Figure 3.3, where the x and y coordinates of the graphics change every tenth of a second.

Random movement is simple to achieve using the basic building blocks outlined here. With experimentation, you'll realize that the possible applications of this technique are limitless.

Modifications

It's easy to modify the `Randomizer` function to alter the presentation of the movie. Changing various properties of the object at different points within the code can have quite dramatic effects—as we're about to see.

Flickering Opacity

Returning to the `Randomizer` function, let's add an extra line that alters the opacity of the graphic.

11. Locate the `Randomizer` function in the first frame of the `MCWorld` movie clip, and adjust it as follows:

File: **Random_Motion_Alpha.fla** MCWorld : ActionScript : 1 (excerpt)

```
function ()
{
  var xShift = random (5) - 2;
  var yShift = random (5) - 2;
  World._x += xShift;
  World._y += yShift;
  World._alpha = random(100);
}
```

12. Save and preview the movie.

Every tenth of a second, at the same time each graphic is given a little nudge, the opacity is now reset to a random value between zero and 99%, producing a flickering effect. How easy was that?

You can even insert additional object properties to, for example, alter the horizontal and vertical scale of the object. Adding the following lines to the above code will randomly scale the graphic objects:

```
World._xscale = random(100);
World._yscale = random(100);
```

Increasing the Redraw Rate

As I mentioned earlier, using `setInterval` to trigger the changes to our graphics disconnected this animation from the frame rate of the movie. To increase the rate of the animation, simply change the delay specified when calling `setInterval`:

File: **Random_Motion.fla** MCWorld : ActionScript : 1 (excerpt)

```
setInterval(Randomizer, 100);
```

Reducing this value will increase the redraw rate (how often the `Randomizer` function is called). Bear in mind that these values are counted in milliseconds. To change the redraw rate to one second, you'd set the value to `1000`. Keep in mind that decreasing the amount of time between redraws will increase the load on the CPU. If your movie uses a large number of objects, or objects that are complex, the user's computer might have a tough time keeping up with the changes.

Increasing the Number of Objects

To increase the number of objects initially drawn on the screen, simply change the `numObjects` variable in the main timeline code:

File: **Random_Motion.fla** ActionScript : 1 (excerpt)

```
var numObjects = 30;
for (i = 0; i < numObjects; i++)
{
```

The `for` loop uses this variable to control the number of objects created, so changing the number of objects couldn't be easier!

If those objects are complex, the CPU load will also increase proportionally. Be careful!

Altering the Random Shift Value

After the objects are originally placed on the canvas at random, the `Randomizer` function shifts the x and y coordinates of the graphics every tenth of a second to give an appearance of jittery nervousness. To increase or decrease this quality, simply locate and edit the following lines within the `Randomizer` function:

File: **Random_Motion.fla** MCWorld : ActionScript : 1 (excerpt)

```
var xShift = random(5) - 2;
var yShift = random(5) - 2;
```

To keep your graphics from wandering off the stage, make sure that the first number on each line is twice the second number plus one. This relationship ensures that the calculated shift values tend to average out to zero. Of course, the easiest way to maintain this relationship is to make it explicit in the code:

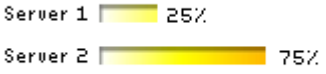
```
var nervousness = 2;
var xShift = random(nervousness * 2 + 1) - nervousness;
var yShift = random(nervousness * 2 + 1) - nervousness;
```

As you can see, once you become comfortable with editing the properties of objects, and randomizing their values, you can create some very interesting effects. The key to finding out what you can do is to experiment with values, explore the ActionScript reference, and have fun!

Simple Scripted Masking

In this example, we'll animate a mask from one point to another, based on input parameters we provide. I created this example to illustrate the traffic received by two Web servers hosting a number of Websites. The movie accepts two input parameters, then animates the mask accordingly. For this simple example, we'll populate the variables statically by declaring them in the root of the timeline. A more realistic scenario would see the movie embedded in a database-driven Web page and the variables passed to the movie dynamically. We'll cover importing external data in Chapter 8.

Figure 3.4. This simple scripted masking effect animates a mask between two points.

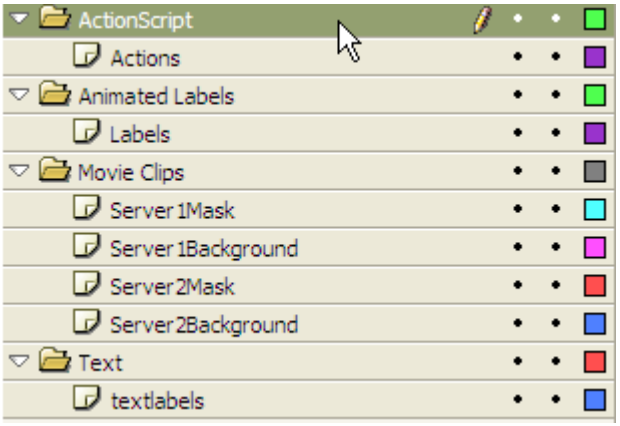


The finished product is shown in Figure 3.4. Let's look at how this effect is accomplished. To skip straight to modifying the effect , locate the file called `Simple_Animation_Masking.fla` in the code archive.

1. Create a new movie that's 200 pixels wide and 40 pixels high. Alter the frame rate to 24 fps for a nice, smooth animation.

2. Create the folders and layers shown in Figure 3.5 below.

Figure 3.5. Organize the layers and folders for the scripted masking effect.



We now need to create the background bar that we'll mask. In this example, I created a bar that's white on the left and gradually became red toward the right, indicating the increase in server load as traffic levels grow.

3. Add two static text fields within the `textlabels` layer and enter text that reads, `Server 1` and `Server 2`, to let users know what each bar represents. We don't need to convert these to movie clips, as we won't reference them in our ActionScript. Move them to (1, 0) and (1, 25) respectively.

4. Within the Server1Background layer, create a new rectangle that's 100 pixels wide and 9 pixels high. Select a gradient fill that changes from white on the left, through yellow and orange, to red on the right of the rectangle. Select the rectangle, then select Insert > Convert to Symbol.... Choose to create a Graphic named `Background`.

5. Name the existing instance of `Background backg1` and move it to (50, 3). Drag a second instance of the graphic from the Library Panel into the Server2Background layer naming it `backg2`, and move it to (50, 29). Lock these two layers; we don't need to modify them any further.

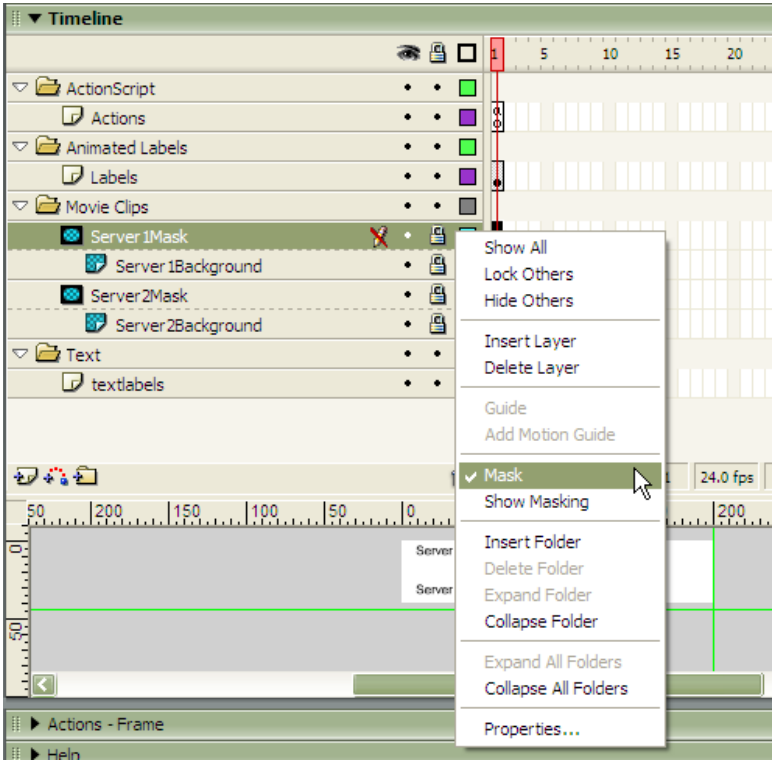
Now that we've created the backgrounds, we can build the masks we'll control via ActionScript:

6. Create a new rectangle, with no stroke and a solid white fill, that's 5 pixels wide and 9 pixels high (this exactly matches the height of the movie clip we will mask). Convert the rectangle to a graphic symbol named `ServerAnimation`, and place instances of the graphic in the Server1Mask and Server2Mask layers.

   Name the instances `server1Mask` and `server2Mask` respectively. (This is important as we will reference these clips in ActionScript later.) Move them to (50, 3) and (50, 29), so they're flush with the left edge of the `backg1` and `backg2` movie clips.

7. To achieve the desired effect, we need to set up the `server1Mask` and `server2Mask` graphics so that they work as **masks** for the background graphics beneath them. Locate the Server1Mask and Server2Mask layers, right-click on each, and select Mask (see Figure 3.6).

Figure 3.6. Convert the dynamic movie clips into masks.



When the movie runs, only those portions of the `Background` graphics in Server1Background and Server2Background that are covered by the `ServerAnimation` graphics in Server1Mask and Server2Mask will be visible.

8. We now need to animate the two mask graphics so that they reveal the appropriate portions of the `Background` graphics. Select the Actions layer, and, with the Actions Panel open, add the following code to the first frame:

File: **`Simple_Animation_Masking.fla`** Actions : 1 (Excerpt)

```
var server1load = 25;
var server2load = 75;

function animate (server, serverload)
{
  server.onEnterFrame = function ()
  {
```

```
    if (this._width <= serverload)
      this._width += 2;
  };
}
animate (server1Mask, server1load);
animate (server2Mask, server2load);
```

That's all the code we need to alter the rectangles to make the two bar graphs grow to their assigned lengths. Let's look at how it's done.

First, we create two variables with values that represent (as a percentage) how much of the `Background` graphic we want to display for each server. The math is kept simple because the `backg1` and `backg2` graphics are exactly 100 pixels wide.

The `animate` function takes a reference to one of our mask graphics and sets up an `onEnterFrame` event handler to increase its width by two pixels per frame up to a specified value. The code finishes by calling `animate` for each of the two mask graphics, passing each of the two server load values.

Save your movie and preview it. Notice how the two masks grow to sizes dictated by the `server1load` and `server2load` variables. It's a pretty cool effect that you can easily include in your projects, creating bar graphs or other visual displays of increases occurring over time.

Adding Text Labels

So far, we've managed to animate masks over time to create a slick, animated bar graph. This is great, but we don't know what values the bars represent. Let's add a text label to each graph to complete the effect:

9. Create a new graphic symbol named `serverinfo` containing a dynamic text field that's 34 pixels wide and 15 pixels high. In the Property Inspector for the text field, set the Var name to `serverload` (which we'll use to set the value to be displayed for each server). Also in the Property Inspector, click Character... and make sure that Embed font outlines for is set to Basic Latin (or All Characters in Flash MX).

10. Drag two instances of this new symbol into the first frame of the Labels layer of the Animated Labels folder. Name the instances `server1info` and `server2info`. Position them at (54, 1) and (54, 27) respectively.

11. Navigate back to the first frame of the Actions layer in the root of the movie. Insert the following code below the first two variables:

File: **Simple_Animation_Masking.fla** Actions : 1 (excerpt)

```
server1info.serverload = server1load + "%";
server2info.serverload = server2load + "%";
```

This code sets the `serverload` variable inside each of the two `serverinfo` symbols, controlling the text they display.

12. Save and preview your work.

You'll notice that the values for each of the bars now simply sit where we placed them. They look a little out of place, given the movement that is occurring. We'd better animate them so that they fit in.

13. Still in the first frame of the Actions layer, add the following function declaration:

File: **Simple_Animation_Masking.fla** Actions : 1 (excerpt)

```
function moveText (serverinfo, serverload)
{
  var startPos = serverinfo._x;
  serverinfo.onEnterFrame = function ()
  {
    if (this._x <= startPos + serverload)
      this._x += 2;
  };
}
```

This function works just like the `animate` function, but it moves the graphic we pass to it horizontally instead of setting its width.

Try some speedy text

For a slightly different effect that adds to the movie's visual appeal, you could have the text move more quickly than the bars by increasing the step size of the movement from two to four.

All that's left is to call this function for each of our text labels to kick off their animation.

14. Add the following code:

File: **Simple_Animation_Masking.fla** Actions : 1 (excerpt)

```
moveText (server1info, server1load);
moveText (server2info, server2load);
```

15. Save your movie and preview it.

That's it! This scripted animation of the text fields completes the effect and looks very cool!

Modifications

You can easily modify this effect to include more items. Simply create more bars, and reference them when the movie loads, to produce interesting graphs. This effect could also generate moving bars that slide into place as the movie is loaded. The direction in which you choose to take this effect really is up to you.

Raindrops Keep Falling on My Head

One of the quickest ways to create an animated effect is to take an animation and duplicate it multiple times on the stage. The success of this technique depends on the original animation being cool enough to warrant this kind of replication. In this example, we'll create a quick rainfall effect by duplicating a movie clip several times on the canvas. Sound simple? Let's look at how it's done.

To skip the details and jump straight into the effect, locate the `Duplication.fla` file in the code archive.

Figure 3.7. This simple raindrop effect is created using duplication.

Setting the Scene

First, we need to create the movie clip we'll reference in our control code.

1. Create a new movie that's 350 pixels wide and 400 pixels high. Increase the frame rate to 18 fps. Create three layers and name them Actions, Raindrops and Background.

2. Create a graphic symbol named `Raindrop` and use the drawing tools to draw a falling drop of water.

3. Create a new movie clip symbol, also named `Raindrop`. Make sure it's open for editing, then drag an instance of the `Raindrop` graphic from the Library

Panel onto the stage. Name the instance `Raindrop`. Position it at (0, 0) within the clip.

4. Create a new keyframe within the movie clip at frame 40. Select frame 1, right-click, and select Motion Tween. Shift back to frame 40 and move the raindrop graphic to the bottom of the stage—about (0, 390).

5. In the Library Panel, duplicate the `Raindrop` movie clip you created, and name this duplicate `RaindropSlow`. Edit the `RaindropSlow` movie clip, grab the end keyframe in the timeline, and drag it out to frame 80. This will produce a slower animation.

We will use these two Raindrop clips to create a subtle effect a little later. Now let's assemble the main scene:

6. Drag one instance each of the `Raindrop` and `RaindropSlow` movie clips into the Raindrops layer. Name them `raindrop` and `raindropSlow`, respectively.

7. Move the two clips so they sit near the top of the stage, but outside its left edge. I placed them at (-45, 10) and (-30, 10). If you can't see past the edge of the stage, you may have to choose View > Work Area first. The goal here is to have the drops as part of the scene, but not visible on stage.

8. Select the Background layer and add some rolling hills and a storm cloud to it using the drawing tools. To finish, lock this layer.

Adding the Control Code

All the graphics are created and in place on the stage; we just need to duplicate them a few times. If you were to preview the movie now, you'd see two single raindrops, one falling faster than the other, off the side of the stage. I think we'd better spice things up with a little ActionScript.

9. Navigate to the first frame of the Actions layer within the timeline and add the following code:

File: **Duplication.fla** Actions : 1

```
for (i = 0; i < 50; i++)
{
  var newDrop = raindrop.duplicateMovieClip ("raindrop" + i,
      i);
  newDrop._x = random (350);
```



```
  newDrop._y = random (20);
}
```

Here, we use the `duplicateMovieClip` method to create 50 copies of the `raindrop` clip that resides on the root of the timeline. If you want more raindrops, you can change the number `50` in the `for` loop to whatever you like. However, beware of the increased CPU load that comes with large numbers of movie clips.

As with the `attachMovie` method we saw earlier in this chapter, `duplicateMovieClip` requires parameters that set the instance name for the new movie clip (in this case, `raindrop` with a number appended to it) and its position in the stacking order.

After we duplicate the movie clip, we assign each duplicate a random horizontal location between zero and the right-hand side of the stage (x=350). We also shift each instance of the clip vertically using a random value between zero and twenty, to make it look as if the raindrops are falling out of different parts of the cloud.

10. Save your movie and preview it within Flash. You'll notice that, even though the raindrops appear to be randomly spaced along the x and y axes, they fall in a straight line. It certainly doesn't rain like this in my neighborhood! We can quickly remedy the situation by introducing more random elements to the code.

11. Replace the code in the Actions Panel with the following:

File: **Duplication.fla** Actions : 1 (excerpt)

```
for (i = 0; i < 50; i++)
{
  var newDrop = raindrop.duplicateMovieClip ("raindrop" + i,
      i);
  newDrop._x = random (350);
  newDrop._y = random (20);
  newDrop.gotoAndPlay(random(40) + 1);
}
```

Notice the extra line of code within this block, shown in bold. This little snippet may look insignificant, but it brings the animation to life. Using the `gotoAndPlay` method of each new clip, the animation is advanced to a random frame between 1 and 40 (remember, `random(40)` generates values from 0 to 39) and then played from that point.

12. Save and preview your movie. You'll notice that the raindrops now fall much more naturally than they did before.

Although this animation is simple to accomplish, it's effective in its execution. There are numerous ways to extend this example and make it more interesting. Let's take a little time to examine them now.

Add Some Randomness

To make this effect more engaging, we can modify the horizontal scale and alpha values of each drop of rain as it's created:

13. Modify the code in the Actions layer as follows:

File: **Duplication.fla** Actions : 1 (excerpt)

```
for (i = 0; i < 50; i++)
{
  var newDrop = raindrop.duplicateMovieClip ("raindrop" + i,
      i);
  newDrop._x = random (350);
  newDrop._y = random (20);
  newDrop._xscale = random (100);
  newDrop._alpha = random (50);
  newDrop.gotoAndPlay (random (40) + 1);
}
```

Save and preview the movie. You'll see raindrops with differing widths—from little, skinny drops to big, fat ones. Each drop's opacity value is also picked at random between zero and 50%.

And the Heavens Opened...

Remember the movie clip we created earlier that was slower than the original raindrop? We have a use for it now! Not all raindrops fall at the same speed, and we can use that clip to make our animation more realistic. Let's add some code that will include the second raindrop movie clip in our animation.

To edit this effect for your own needs, locate `Duplication_Modification.fla` in the code archive.

14. Add the following code below the existing code in the first frame of the Actions layer:

File: **Duplication_Modification.fla** Actions : 1 (excerpt)

```
for (j = i; j < i + 100; j++)
{
  var newDrop = raindropSlow.duplicateMovieClip (
      "raindropSlow" + j, j);
  newDrop._x = random (350);
  newDrop._y = random (20);
  newDrop._xscale = random (100);
  newDrop._alpha = random (25);
  newDrop.gotoAndPlay (random (80) + 1);
}
```

This works just like the previous block of code: the `for` loop creates a number of duplicates of the movie clip, then uses the `random` function to set values for various properties of the duplicates. Here are the differences in this second block of code:

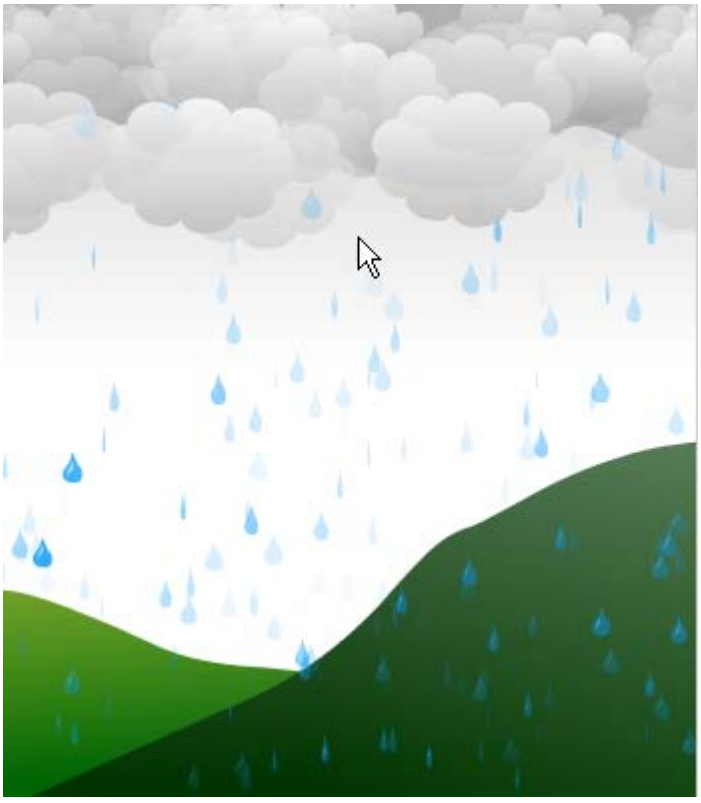
- Our `for` loop counts to 100, instead of 50, so we'll create twice as many duplicates.
- We create duplicates of the `raindropSlow` clip this time.
- We use a new counter variable for this loop (`j`), and add it to the count at the end of our previous loop (`i`) when setting the stacking order of the new duplicates. This ensures that all the drops get their own place in the stacking order. (Otherwise, the slow raindrops would replace the fast raindrops on the stage!)
- We set alpha values between zero and 25% for our slow raindrops, to make them appear further away.
- Because `RaindropSlow` is 80 frames in length instead of forty, we have adjusted the value we pass to `random` on the last line.

That's it for the raindrops! Now let's see if we can further enhance the scene with some more complicated effects.

Creating a Parallax Cloud Structure

Like raindrops, the movement of clouds is extremely random—each cloud moves at a different speed. With some clever math and a few simple cloud movie clips, you can create an interesting effect that adds depth to the scene (see Figure 3.8).

Figure 3.8. By choosing the right random values, you can create this smooth cloud movement.



We'll start where we left the example in the previous section. To jump straight to the finished product, locate in the code archive.

15. Create two new movie clip symbols, named `LargeCloud` and `SmallCloud`, and place an image that resembles a cloud in each. Make the cloud in the `SmallCloud` clip about half the size of its counterpart in `LargeCloud`.

16. Create above the RainDrops layer a new layer called Clouds, and drag instances of the two new movie clips into this layer. Name them according to their master movie clips (`largeCloud` and `smallCloud`, respectively). Again, place them off the side of the stage.

We'll use these two clouds in a manner similar to our work with the two raindrops, duplicating them with ActionScript code to create a random scene. To add a feeling of depth to our cloud structure, we'll create a **parallax** effect. This involves making faraway objects (our small clouds) move more slowly than nearby objects (our large clouds), which creates a sense of perspective and depth.

17. Add the following code to frame 1 of the Actions layer, beneath the existing code:

File: **Duplication_Modification_Clouds.fla** Actions : 1 (excerpt)

```
for (i = j; i < j + 60; i++)
{
  var newCloud = smallCloud.duplicateMovieClip (
      "smallCloud" + i, i);
  newCloud._alpha = random (100);
  newCloud._x = random (450) - 100;
  newCloud._y = random (60) + 10;
  newCloud.step = random(4);
  newCloud.onEnterFrame = cloudStep;
}
for (j = i; j < i + 30; j++)
{
  var newCloud = largeCloud.duplicateMovieClip (
      "largeCloud" + j, j);
  newCloud._alpha = random (100);
  newCloud._x = random (450) - 100;
  newCloud._y = random (40) - 20;
  newCloud.step = random(4) + 2;
  newCloud.onEnterFrame = cloudStep;
}
function cloudStep()
{
  if (this._x >= 350) this._x = -100;
  this._x += this.step;
}
```

As you can probably figure out by examining the code, we're creating 60 duplicates of the small cloud and thirty duplicates of the large cloud. Our loops continue to use the `i` and `j` variables so that the clouds are added to the top of the stacking order and the raindrops appear to come from behind or within them.

For each cloud we assign a random opacity between 0% and 99%, and a random horizontal position between -100 and 350. Remember that this is the position

of the left edge of the cloud, so we need those negative values to allow for clouds partially obscured by the left edge of the stage.

To develop the sense of depth even further, and to ensure our small clouds aren't obscured by the large clouds, we make our small clouds sit lower on the stage (with random vertical positions from 10 to 69) than our large clouds (from -20 to 19). With the small clouds closer to the horizon, they will seem further away.

Now for the crux of our parallax effect: the motion of the clouds. All of our clouds will move across the stage from left to right. Each cloud will have its own randomly assigned step size, which indicates the number of pixels per frame it should move. For the small clouds, we generate step sizes from zero to three pixels, while the large clouds will get step sizes from two to five pixels. We store each cloud's step size into a variable called `step` within the cloud's movie clip (`newCloud.step`).

Finally, we add an `onEnterFrame` event handler for each of the clouds, all of which will use a common function called `cloudStep`. This uses the clip's step size to move it to the right until it reaches a horizontal position of 350 pixels, at which point it's sent back to -100.

18. Save the movie and preview it. To see how the effect looks without the objects running off the stage, export the movie to a SWF file and double-click it to view the movie in Flash Player.

That's a pretty cool effect! All the clouds move at different speeds, so the effect doesn't look "manufactured." But there is still more we can add to this scene…

Creating a Lightning Flash

That storm we just created looks pretty good, but it could do with some sheet lightning to add that finishing touch.

We'll pick up where we left off in the previous section. To jump straight to the end of this example, locate `Duplication_Modification_Clouds_And_Flash.fla` from the code archive.

19. Create a new movie clip named `LightningFlash` and add a rectangle that fills the stage (350x400 pixels). Give it a white gradient fill, fading from 100% opacity at the center of the rectangle to 0% opacity at the bottom edge of the rectangle.

20. Back in the main timeline, create a new layer named Flash above the Clouds layer and drag an instance of the new movie clip into it. Name the instance `flash`. You can lock and hide the layer—we won't need to edit it again.

Now that we've created the movie clip, we can add the control code:

21. Once again, add the following code to the end of frame 1 of the Actions layer:

File: **Duplication_Modification_Clouds_And_Flash.fla** Actions : 1 (excerpt)

```
flash._alpha = 0;
flash.onEnterFrame = function ()
{
  var flashControl = random (10);
  if (flashControl >= 9 ||
      flash._alpha > 0 && flashControl >= 5)
  {
    flash._alpha += random (65);
    if (flash._alpha > 65)
    {
      flash._alpha = 0;
    }
  }
};
```

We start by setting the `flash` movie clip's opacity to zero, so that the lightning doesn't appear until we want it to.

Next, we tackle the `onEnterFrame` event handler, which will control the opacity of the lightning for each frame of the movie.

Even in the worst storms, lightning is an intermittent thing. After some fiddling, I've decided I want a 10% chance that a flash of lightning will occur in any given frame. So I use `random` to generate a number (`flashControl`) between zero and nine and write my code so that it initiates a lightning flash whenever that number is nine.

Within the body of the `if` statement, the code goes on to add a random value between 0 and 64 to the opacity of the `flash` movie clip. Once this takes place, we want the lightning to continue to grow in brightness until it hits the maximum brightness for a flash of lightning (which, after some experimentation, I've decided is 65% opacity. At that point, we set the opacity back to zero and wait for the next flash of lightning.

When a lightning flash occurs, we don't want to wait for that one-in-ten chance of lightning to make the existing flash brighter. And, at the same time, we can add some variety to our lightning flashes by not having them grow brighter with every frame. That's why the condition in the `if` statement is so complex—if a lightning flash is in progress (which we detect by checking if `flash._alpha` is greater than zero), then we allow the flash to grow brighter 50% of the time (whenever `flashControl` is five or greater).

22. Save the movie and preview it in Flash.

The conditions we've employed here, including the use of random values, cause the `_alpha` value of the `flash` movie clip to flash in and out, producing quite a stormy scene. Feel free to experiment with the probabilities I've put in place to make the storm more or less severe.

Creating User-driven Motion

In this example, we'll create motion based on user input. When the user clicks one button, objects slide into place; when another button is clicked, the objects slide back to their starting points (see Figure 3.9). This example builds upon previous animation examples and is fully scripted—there's not a motion tween in sight!

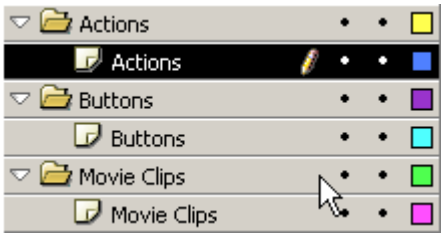
Figure 3.9. This effect is driven by user input.

If you don't feel like creating this effect from scratch, locate `User_Drive_Motion.fla` in the code archive.

Setting the Scene

1. Start by creating a new movie that's 400 pixels wide and 185 pixels high, and has the folder and layer structure shown in Figure 3.10.

Figure 3.10. Create the folder structure for the user-driven motion effect.



2. Create a new movie clip named `Animation_Base`. In it, place a 190x10 pixel rectangle. Feel free to replace this rectangle with whatever you wish, as we'll be using it only to illustrate the effect of this animation.

3. Drag four instances of the `Animation_Base` movie clip into the Movie Clips layer, naming them `line1`, `line2`, `line3`, and `line4`, and positioning them at (10, 10), (10, 20), (10, 30), and (10, 40) respectively.

4. Set the Color for each `Animation_Base` instance to an Alpha value of 0%, so that the clips start off invisible.

5. Create two new button symbols called `Trigger_Show` and `Trigger_Hide`, and create instances of them in the Buttons layer called `trigger_show` and `trigger_hide`, respectively. Don't be too concerned about the way they look—you can redesign them later when you modify the effect for your own use.

Adding the ActionScript

Now, we'll create a function that will take care of all the movement in this effect, including the speed at which the objects move into and out of position:

6. Select the first frame of the Actions layer, and add the following code within the Actions Panel:

File: **User_Drive_Motion.fla** Actions : 1 (excerpt)

```
function MoveTo (clip, fadeType, xTo, yTo, speed)
{
  clip.onEnterFrame = function ()
  {
    this._x += (xTo - this._x) * speed;
    this._y += (yTo - this._y) * speed;
    if (fadeType == "in" && this._alpha < 100)
    {
      this._alpha += 5;
    }
    else if (fadeType == "out" && this._alpha > 0)
    {
      this._alpha -= 5;
    }
  };
}
```

The `MoveTo` function accepts the following parameters:

`clip`	The clip we're animating
`fadeType`	The type of fade effect to display (`"in"` or `"out"`)
`xTo`	The final horizontal position for the animation
`yTo`	The final vertical position for the animation
`speed`	The speed of the clip's movement (between 0 and 1 for smooth animation)

Looking over the code, it should be pretty obvious what `MoveTo` does. It sets up an `onEnterFrame` event handler for the specified movie that will ease the clip to the specified coordinates while fading its opacity in or out.

The effect will be triggered by the `trigger_show` button, and reversed by the `trigger_hide` button. So all we really need to do is call `MoveTo` with the proper parameter values whenever one of these buttons is clicked.

Adding the Button Trigger Code

Now, we'll add the button trigger code that will make the function work.

7. Add the following code beneath the function declaration we created in the previous section:

```
trigger_show.onPress = function ()
{
  MoveTo (line1, "in", 50, 10, 0.3);
  MoveTo (line2, "in", 100, 20, 0.3);
  MoveTo (line3, "in", 150, 30, 0.3);
  MoveTo (line4, "in", 200, 40, 0.3);
};
```

   When the `trigger_show` button is pressed, the `MoveTo` function will be called, moving the clips to their new positions. In this example, we move the movie clips horizontally, as the vertical coordinate matches their starting positions on the stage.

Let's now add the code that will be called when the `trigger_hide` button is pressed, returning the clips to their resting states:

8. Add the following code:

```
trigger_hide.onPress = function()
{
  MoveFromTo(line1, "out", 10, 10, 0.3);
  MoveFromTo(line2, "out", 10, 20, 0.3);
  MoveFromTo(line3, "out", 10, 30, 0.3);
  MoveFromTo(line4, "out", 10, 40, 0.3);
};
```

9. Save and preview your work.

The effect you see is smooth and crisp. Clicking the first button moves the objects into place, while clicking the second option tucks them away.

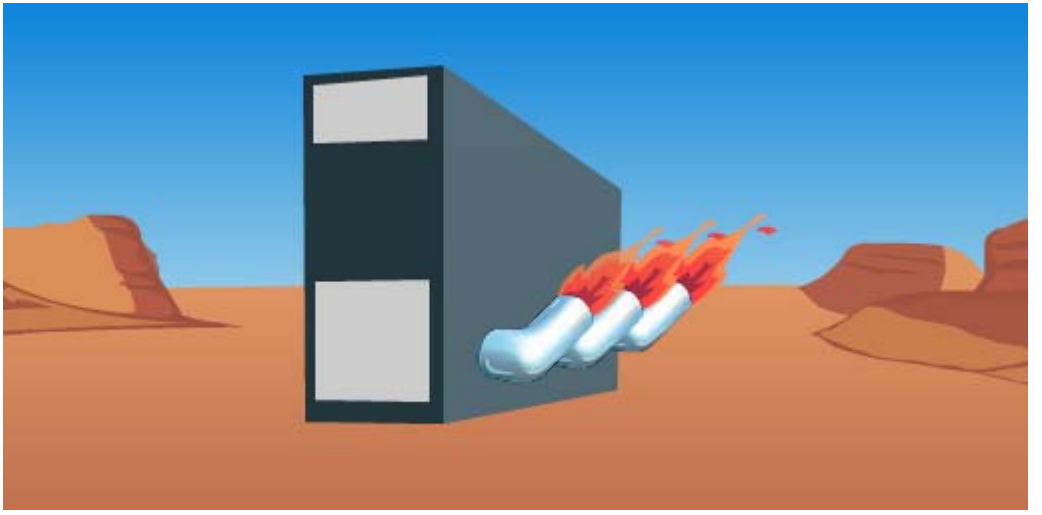
Looks like a menu to me!

You could easily build this effect into a navigation element by nesting buttons within the movie clips and altering the passed parameters to suit your needs.

Subtle Flame Animation

With the right balance of subtle animation, you can create effects that are both interesting and visually appealing. In this example, we'll create a realistic flame animation for our hot-rod server (Figure 3.11).

Figure 3.11. Use random movement to create subtle animation effects.

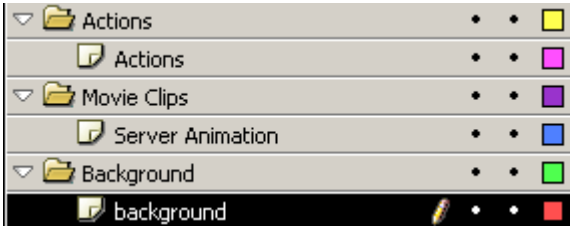


You can grab the finished version of this example and edit it for your own needs by locating `Flaming_Server.fla` in the code archive.

1. Create a new movie that's 600 pixels wide and 300 pixels high; set the frame rate to 24 fps.

2. Create the layer and folder structure shown in Figure 3.12.

Figure 3.12. Use this folder and layer structure.



3. Create a background graphic that will serve as the backdrop for the effect within the background layer. In Figure 3.11, I've used a Grand Canyon background.

4. Create a new movie clip named `Flame`, and create three keyframes. Using the pen tool, create three different flames, one on each keyframe, as shown in Figure 3.13. We'll use these to create the random flame effect.

Figure 3.13. Create three different flames for a subtle animation effect.

5. Drag three instances into the first frame of the Server Animation layer, naming them `flame1`, `flame2`, and `flame3`.

6. Add the following code to the first frame of the Actions layer on the root of the timeline:

```
function randomBetween (a, b)
{
```

```
  return Math.min (a, b) + random (Math.abs (a - b) + 1);
};
```

What we have here is a useful function that will provide a random integer between the two given numbers (inclusive). Work through the math if you like—`Math.min` returns the smaller of two numbers, while `Math.abs` returns the absolute (positive) value of a number. We'll be using this function throughout this example to create additional subtlety and randomness.

7. Add the following code:

```
Flame1.gotoAndPlay (randomBetween (1, 3));
Flame2.gotoAndPlay (randomBetween (1, 3));
Flame3.gotoAndPlay (randomBetween (1, 3));
```

This randomly starts the animation of each flame between frames one and three.

Now, we have a subtly alternating effect. The next section of code adds a little more randomness—in the real world, flames aren't always the same length. Let's make the size of each flame change at random.

8. Add the following code:

```
Flame1.onEnterFrame = function ()
{
  this._yscale = randomBetween (100, 140);
  this._alpha = randomBetween (60, 100);
};
Flame2.onEnterFrame = function ()
{
  this._yscale = randomBetween (100, 180);
  this._alpha = randomBetween (60, 100);
};
Flame3.onEnterFrame = function ()
{
  this._yscale = randomBetween (100, 200);
  this._alpha = randomBetween (60, 100);
};
```

This assigns an `onEnterFrame` event handler to each of the flames. These handlers randomly modify the opacity and vertical scale of the clips for each frame of the animation. Feeding different values to `randomBetween` gives a different quality to each of the flames.

9. You're done! Preview the movie in Flash.

In the completed example that I've created for you, I've put the flame movie clips behind three exhaust pipe graphics and rotated them to make them look realistic when bolted onto the side of a server box, which is a simple graphic symbol.

Conclusion

This chapter has been an adventure in ActionScript animation techniques, and we've covered a lot of ground—from simple animation, to scripted masking and complex duplication techniques. I hope you'll grab the source files from the code archive and rip them apart! Use the techniques that you've learned here to improve your own projects and create new effects.

In Chapter 4, we'll apply these same techniques to create some truly intriguing text effects—effects that really harness the new powers of Flash MX 2004!