

Flash  
MX 2004



# *The Flash Anthology*

## *Cool Effects & Practical ActionScript*



By Steven Grosvenor

Practical Solutions to Common Problems

# **The Flash Anthology: Cool Effects & Practical ActionScript (Chapters 1, 3, and 5)**

---

Thank you for downloading the sample chapters of Steven Grosvenor's book, *The Flash Anthology: Cool Effects & Practical ActionScript*, published by SitePoint.

This excerpt includes the Summary of Contents, Information about the Author, Editors and SitePoint, Table of Contents, Preface, 3 chapters of the book and the index.

We hope you find this information useful in evaluating this book.

[For more information or to order, visit sitepoint.com](http://sitepoint.com)

## Summary of Contents of this Excerpt

Preface .....	ix
1. Flash Essentials .....	1
3. Animation Effects .....	87
5. Sound Effects .....	203
Index.....	433

## Summary of Additional Book Contents

2. Navigation Systems .....	27
4. Text Effects .....	127
6. Video Effects .....	247
7. Flash Forms .....	293
8. External Data .....	355
9. Debugging .....	393
10. Miscellaneous Effects.....	415
Resource Sites for Flash .....	431



# **The Flash Anthology**

**Cool Effects & Practical ActionScript**

**by Steven Grosvenor**

---

# The Flash Anthology: Cool Effects & Practical ActionScript

by Steven Grosvenor

Copyright © 2004 SitePoint Pty. Ltd.

**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

All rights reserved. No part of this book may be reproduced, stored in a retrieval system or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical articles or reviews.

## Notice of Liability

The author and publisher have made every effort to ensure the accuracy of the information herein. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors and SitePoint Pty. Ltd., nor its dealers or distributors will be held liable for any damages to be caused either directly or indirectly by the instructions contained in this book, or by the software or hardware products described herein.

## Trademark Notice

Rather than indicating every occurrence of a trademarked name as such, this book uses the names only in an editorial fashion and to the benefit of the trademark owner with no intention of infringement of the trademark.



Published by SitePoint Pty. Ltd.

424 Smith Street Collingwood  
VIC Australia 3066.

Web: [www.sitepoint.com](http://www.sitepoint.com)

Email: [business@sitepoint.com](mailto: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](http://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

<b>Preface</b> .....	<b>ix</b>
Who Should Read This Book? .....	ix
What's in This Book? .....	x
The Book's Website .....	xii
The Code Archive .....	xii
Updates and Errata .....	xiii
The SitePoint Forums .....	xiii
The SitePoint Newsletters .....	xiii
Your Feedback .....	xiii
Acknowledgements .....	xiv
<b>1. Flash Essentials</b> .....	<b>1</b>
Why Use Flash? .....	1
What's New In Flash MX 2004? .....	3
Flash MX 2004 Features at a Glance .....	3
Comparing Vectors and Bitmaps .....	6
Interactivity Unlimited .....	7
ActionScript Uncovered .....	8
ActionScript Dot Notation .....	10
Actions Panel Unraveled .....	12
Actions Panel Framework .....	14
Optimizing the Actions Panel .....	14
Configuring Auto Format .....	15
Code Commenting .....	16
ActionScript Coding Tips .....	18
The Timeline, Simplified .....	20
Label Me Up .....	22
Organize Your Layers .....	22
Timeline Effects (New in Flash MX 2004) .....	23
Wrapping Up the Basics .....	25
<b>2. Navigation Systems</b> .....	<b>27</b>
What Makes Effective Navigation? .....	27
Reduce Confusion .....	28
Maximize Your Hierarchy .....	28
Don't be too "Out There" .....	28
Keep it Simple .....	28
Test Usability .....	29
Planning Navigation .....	29
Make a Process Flow Chart for Your Navigation .....	29

---

Common Navigation Methods .....	31
Horizontal Navigation .....	31
Vertical Navigation .....	32
Gadget-based Navigation .....	33
Horizontal Navigation .....	34
Simple Horizontal Navigation .....	34
Silky Fading Opacity Effect .....	43
Vertical Navigation .....	49
Setting the Scene .....	50
Adding the ActionScript .....	51
Gadget-based Navigation .....	55
Setting the Scene .....	57
Adding the ActionScript .....	59
Modifications .....	63
Conclusion .....	65
Subliminal Navigation .....	65
Setting the Scene .....	66
Adding the ActionScript .....	67
Modifications .....	71
Advanced Navigation .....	72
Tabbed Interface Extraordinaire .....	72
Conclusion .....	85
<b>3. Animation Effects .....</b>	<b>87</b>
Animation Principles .....	88
Animation Overload .....	89
To Tween or Not to Tween? .....	90
Creating Function Libraries .....	92
Creating a Simple Function Library .....	93
Creating Master Libraries .....	94
Random Motion .....	95
Setting the Scene .....	95
Adding the ActionScript .....	97
Testing the Movie .....	99
Modifications .....	100
Simple Scripted Masking .....	102
Adding Text Labels .....	106
Modifications .....	108
Raindrops Keep Falling on My Head .....	108
Setting the Scene .....	109
Adding the Control Code .....	110
Add Some Randomness .....	112

---

And the Heavens Opened...	112
Creating a Parallax Cloud Structure	113
Creating a Lightning Flash	116
Creating User-driven Motion	118
Setting the Scene	119
Adding the ActionScript	119
Adding the Button Trigger Code	120
Subtle Flame Animation	122
Conclusion	125
<b>4. Text Effects</b>	<b>127</b>
When to Use Text Effects	128
Types of Text Effects	128
When Not to Use Text Effects	129
Lighting Effects	130
Dazzling Chrome Effect	130
Neon Text Effect	134
Glow in the Dark Effect	140
Motion Effects	143
Jiggling Text Effect	143
Zooming In Effect	149
Zooming Out Effect	154
Circulating Text Effect	159
Bouncing Text Effect	164
Opacity Effects	168
Random Fading Text Effect	168
Simple Opacity Transition	173
Slides	178
Advanced Text Effects	184
Falling Text	184
3D Barrel Roll	195
Timeline Effects	201
Conclusion	202
<b>5. Sound Effects</b>	<b>203</b>
When Should You Use Sound in Your Projects?	203
Selecting Sound Clips	204
Importing and Exporting Sound Clips	205
Dynamic Volume Control	206
Setting the Scene	206
Adding the ActionScript	207
Dynamic Panning Control	209

Setting the Scene .....	209
Adding the ActionScript .....	210
Mini Sound Player .....	211
Setting the Scene .....	211
Adding the ActionScript .....	212
Modifications .....	215
Random Track Sequencer Effect .....	218
Setting the Scene .....	219
Adding the ActionScript .....	219
Random Track Overlay Effect .....	221
Setting the Scene .....	222
Adding the ActionScript .....	224
Modifications .....	229
Setting the Scene .....	229
Adding the ActionScript .....	230
XML, MP3, and ID3 Tag Reader .....	236
Setting the Scene .....	237
Creating the XML Playlist .....	238
Adding the ActionScript .....	239
Modifications .....	243
Conclusion .....	245
<b>6. Video Effects .....</b>	<b>247</b>
When to Use Video in Your Flash Movies .....	248
Video Inclusion Options .....	249
Capturing Your Movie .....	251
Importing Your Movie .....	251
Compression Settings .....	253
Advanced Video Import Settings .....	255
Accessing External FLV Files .....	257
Setting the Scene .....	258
Adding the ActionScript .....	260
Modifications .....	261
Creating a Video Wall .....	273
Setting the Scene .....	274
Adding the ActionScript .....	275
Modifications .....	276
Creating a Video Scrubber Device .....	279
Setting the Scene .....	280
Adding the ActionScript .....	281
Subtle Use of Video .....	288
Setting the Scene .....	291

---

Conclusion .....	292
<b>7. Flash Forms .....</b>	<b>293</b>
Creating Usable Flash Forms .....	296
Form Function .....	296
Required Information .....	297
Data Validation .....	297
Form Basics: A Simple Login Form .....	298
Setting the Scene .....	299
Adding the ActionScript .....	304
Modification: Adding a Focus Glow .....	308
Handling Form Data .....	311
Setting the Scene .....	312
Adding the ActionScript .....	314
Snazzy Forms: Creating a Simple Navigation System from Form Components .....	323
Setting the Scene .....	324
Adding the ActionScript .....	326
Intricate Forms and Form Validation .....	327
Setting the Scene .....	328
Adding the ActionScript .....	334
Creating a Scripted Questionnaire .....	341
Setting the Scene .....	342
Adding the ActionScript .....	347
Conclusion .....	353
<b>8. External Data .....</b>	<b>355</b>
Setting Variables Outside of Flash: A Breadcrumb Header .....	355
Setting the Scene .....	356
Adding the Script .....	357
Inter-Movie Communications .....	362
Setting the Scene .....	363
Adding the ActionScript .....	365
Creating a Simple Blog Reader Application .....	369
Setting the Scene .....	370
The Server-Side Script .....	372
Adding the ActionScript .....	373
Storing Preferences with a Local Shared Object .....	378
Setting the Scene .....	379
Adding the ActionScript .....	380
Modifications .....	385
Creating a Note Application Using External Data .....	385

Setting the Scene .....	387
Adding the ActionScript .....	388
Modifications .....	391
Conclusion .....	392
<b>9. Debugging .....</b>	<b>393</b>
Common Causes of Errors .....	393
Syntax Errors .....	393
Logic Errors .....	395
Runtime Errors .....	396
Debugging Basics: Using trace .....	396
Setting the Scene .....	397
Adding the ActionScript .....	397
Using the Error Object .....	399
Using the Flash Debugger .....	401
Harnessing the Power of the Debugger .....	402
The Anatomy of the Debugger .....	403
Tiptoe Through the Code .....	405
Tracking Data in Complex Conditions and Functions .....	407
Using the Debugger to Follow Logical Process Flow .....	409
Debugging Applications with Watches .....	412
Conclusion .....	413
<b>10. Miscellaneous Effects .....</b>	<b>415</b>
CSS in Flash .....	415
Setting the Scene .....	416
Creating the CSS .....	417
Adding the ActionScript .....	419
Charting in Flash .....	422
Setting the Scene .....	423
Adding the ActionScript .....	424
Getting Indexed by Search Engines .....	426
meta Tags .....	426
Macromedia Flash Search Engine Software Development Kit (SDK) .....	428
Conclusion .....	428
Resource Sites for Flash .....	431
Index .....	433

---

# 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 through 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](mailto: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!

# 1

## Flash Essentials

---

Life without Flash would be uninteresting and mundane. Flash sites are to static HTML sites what a family-size, deep-crust pizza with all the toppings is to a piece of toast. Many of today's big-impact sites are either full-blown Rich Internet Applications (RIAs) or a prudent blend of HTML and Flash. This careful melding of technologies, coupled with seamless integration, means the difference between an online experience that's striking, and one that's utterly forgettable.

Sites that use Flash as the sole medium for conveying their message, and sites that use several mini-Flash applications to punctuate static HTML, have a common underlying theme: they all harness the power of Flash to create lightweight vector- (or coordinate-) based interfaces, applications, and animations. Flash has become a powerful tool for the communication of ideas, its capabilities having inspired a large and passionate following of dedicated users.

## Why Use Flash?

The new user can approach Flash from many different angles. Designers may well be impressed by Flash's capabilities in the realms of interface design, aesthetics, and functionality. If you have a strong coding background, mathematical experimentation and the opportunity to learn new coding techniques may pique your interest. Regardless of which direction you approach it from, the technology offers something to every budding Flash developer.

---

Flash's inherent ability to create super-compact, vector-based animations, coupled with a powerful scripting language (ActionScript), allows users to develop complex effects, transitions, and interfaces with relative ease—something that would take many more hours, or be completely impossible, using traditional HTML and DHTML coding methods.

Flash has enjoyed several incarnations since the early days before Macromedia acquired the now ubiquitous plug-in and authoring tool, and development continues today with the recent release of Flash MX 2004 Professional. The passing years have seen the use of Flash technology shift from black art to mainstream Web development practice, as each new version of the software provides users with additional capabilities, from animation, sound, and interactivity, to application server access and video inclusion.

If a complete Flash newbie were to purchase the latest copy of Flash and install it, he or she would likely be overwhelmed by the plethora of panels, drop-down menus, and user options available. Yet, after some experimentation, and a little time spent reading the manual and online tutorials, things would start to make sense. If you're in this boat—if this is the first book you've purchased since your recent acquisition of Flash—congratulations! You're in good company. If, on the other hand, you know your way around the application and its interface, and know what you want to do but are not exactly sure how to do it, then you, too, have come to the right place. Throughout this book, I'll attempt to present methods and procedures that address the questions that are most commonly posed in online forums and newsgroups. Sure, there are thousands of examples I could show you, but the solutions I've included here are the pick of the crop. Once you understand how the examples in this book fit together, you'll be able to apply them to many different instances and situations, and use them as building blocks for other Flash applications you may create.

Flash's power springs from its dynamism and versatility. There are so many things you can do with it, from creating simple animations for inclusion in your Web applications, to building robust SQL and XML data-feeding super-applications. It can be difficult to identify a single point at which to start or stop. You could decide to create a low-bandwidth animated headline for your Website; you might choose to build the whole of your Website as a Flash application or series of Flash applications. The choice is entirely up to you. This book provides real-world examples and leading-edge techniques involving different aspects of Flash. After each example, modifications will be presented that will allow you to extend that example to suit your particular needs.

I'll assume you're using Flash MX or later, and that you have some understanding of the software. This book won't walk you through the interface and tell you what it can do; instead, I expect you have a grip on the basics, and that you're itching to start using the program to its full potential. Some of the examples in this book are specific to Flash MX 2004, demonstrating powerful techniques for getting the most out of that version in the shortest time, but the majority apply to Flash MX and above.

It's time to buckle up and turn on the ignition. Let's see what Flash has to offer!

## What's New In Flash MX 2004?

Flash MX 2004 is a step above previous versions. Not only does it offer a plethora of enhancements and new features, but the software comes in two "flavours": Flash MX 2004 and Flash MX Professional 2004. Your needs will determine which version is right for you. Flash MX 2004 is the ideal instrument for developing multimedia content, or adding video, audio, graphics, or data to your projects. Flash MX 2004 Professional, on the other hand, contains additional features such as forms-based programming (much like Microsoft Visual Studio), data connectors for binding XML and Web Services to your projects, and project management tools.

If you intend to create projects that contain anything more than a modicum of ActionScript, you require connection to external data sources, or want to harness the power of advanced components, then I would strongly advise you to buy Flash MX Professional 2004. The day will come—probably sooner than you expect—when you'll need the extra horsepower it delivers. Don't get caught short!

## Flash MX 2004 Features at a Glance

Let's take a look at the major improvements and new features offered by the 2004 releases. Those features marked (PRO) are available in the professional edition only.

### ActionScript 2.0

A major object-oriented improvement to the ActionScript model, ActionScript 2.0 allows for strong typing and inheritance in accordance with the ECMA specifications. Don't worry if these terms are new to you—we'll discuss them in detail later.

### **Performance**

The new version of the Flash Player has been substantially overhauled to increase playback speeds and responsiveness.

### **Error Reporting**

The error reports generated when a Flash movie is previewed internally have been much improved in this release of the software. The reports provide more descriptive information, which makes it easier to fix show-stopping errors. See Chapter 9 for more information.

### **Security Model**

The Flash Player security model has been improved significantly, and now provides even tighter security for your applications.

### **Importing**

Flash developers can now import several new formats, including Adobe PDF and Adobe Illustrator 10. Flash MX 2004 also includes a Video Import Wizard to make importing video formats easier. See Chapter 6 for details on this functionality.

### **Spell Checking**

A fantastic new feature, spell check allows you to search text within your movies for typographical errors.

### **Help Panel**

The Help Panel includes context-based help as well as a full ActionScript reference that updates at the click of a button with the latest content from Macromedia.

### **New Templates**

A list of new templates designed to aid the rapid development of Flash applications is available with the 2004 release of Flash. The templates include:

- Advertising
- Form Applications (PRO)
- Mobile Devices
- Photo Slideshows
- Presentations

- Quiz
- Slide Presentations (PRO)
- Video (PRO)

To access the templates, select File > New, and click the Templates tab.

### **Timeline Effects**

Timeline Effects allow you quickly to create editable common effects such as exploding text, blurred animation, and opacity transitions. Due to the extended API of Flash in the 2004 editions, many third-party software vendors now create these effects for use in Flash. An excellent range of third-party Timeline Effects is available from Macromedia's online store[1], and includes Distort FX, Pixel FX, and Text FX, developed by Red Giant Software[2]. These allow you to create some mind-bending visuals from within Flash.

### **Behaviors**

Behaviors allow you to quickly add basic ActionScript commands to your movies. They can form building blocks for your code, or you can simply use them to add interactivity without getting your hands dirty with ActionScript.

To access the Behaviors Panel, select Window > Development Panels > Behaviors.

### **Advanced Components (PRO)**

Flash MX 2004 Professional includes a new series of components that facilitate the development of complex applications.

### **Flash Form Presentation (PRO)**

Flash Form Presentation reflects a new development system much like that of Microsoft Visual Studio. It allows for the rapid creation of, for example, contact and registration forms. See Chapter 7 for more.

### **Flash Slide Presentation (PRO)**

Creating slide-based applications has never been easier, thanks to the inclusion of built-in navigation to help users move through the slides.

---

[1] <http://www.macromedia.com/software/flash/extensions/>

[2] <http://www.redgiantsoftware.com/>

### **Data Binding and Data Connection Objects (PRO)**

Flash MX 2004 Professional introduces built-in data connectors to external data sources such as Web services and XML. With the introduction of Data Binding, components such as `ComboBoxes` can be populated with external data relatively easily.

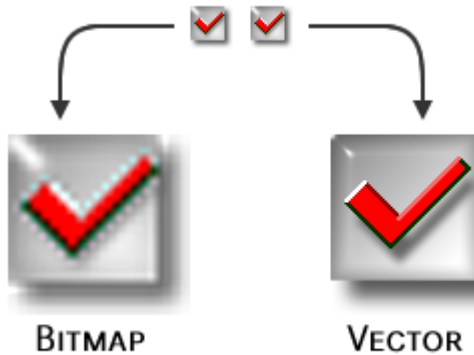
### **Source Control (PRO)**

You can now easily leverage source control in your Flash projects by integrating Flash with Microsoft Visual SourceSafe. This ensures you don't overwrite work by other team members on critical projects via the 'Check-In/Check-Out' methodology.

## **Comparing Vectors and Bitmaps**

During the early days of Web design, bandwidth was always an issue—the greater the page content, and the larger the number of images, the longer the wait for the page to render in the browser. Long download times proved to be a big turn-off for many surfers. However, as the Web evolves and high-speed connections become more commonplace in homes and workplaces, less emphasis is placed on creating super-tight pages with ultra-skinny images. Of course, it's still best practice to write concise code, and make sure your images are as small as possible, in order to keep download times to a minimum. With Flash's vector-based graphics system, you can consistently generate Web-optimized images and create low bandwidth applications. The beauty of creating images within Flash is that when you zoom into the movie, you don't experience the loss of clarity or the pixellation that occurs with traditional bitmaps. This produces a crisper look and feel, as shown in Figure 1.1.

**Figure 1.1. Compare bitmap and vector image file formats.**



You can import traditional bitmaps into Flash, but these will obviously increase the size of the final movie. For example, if you compared a vector-based interface created within Flash to an interface built from multiple bitmaps and HTML, the Flash-developed interface file size would be dramatically lower, and the download speed significantly faster, than the bitmap-and-HTML version of the page.

## Interactivity Unlimited

Flash uses an object-oriented programming language called ActionScript, which shares many themes and functions with JavaScript. As JavaScript is familiar to many Web designers, understanding the ActionScript syntax can be relatively easy. In fact, this is the case even for those without a programming background. Subsequent chapters provide a full explanation of how the application fits together, as well as the reasoning behind ActionScript code.

Designing interactive effects in Flash stumps many developers. They might have an idea of what they want to do, but they don't know how to do it, what can be scripted, or which effects require normal timeline interaction and tweening. The major advantage of creating scripted effects is that merely altering a few variables or the way a function is referenced can have a dramatic impact upon the final movie. Creating effects with traditional methods means altering the timeline, keyframes, and tweening, and can be quite daunting—especially when you're working with a large number of frames and layers. The techniques outlined in this book will involve scripted effects unless otherwise noted.

When a new user casts their eye over several lines of ActionScript, the typical response is a glazing over of the eyes. The sight of virtually any programming language can scare beginners away, but it's important to remember that ActionScript isn't difficult to understand; its learning curve is comparatively gentle. Events are triggered in three ways: buttons within the interface are clicked, certain conditions are met, or the playhead of the timeline reaches a certain frame.

While ActionScript can get very complicated, there are many ways to create quick, pleasing effects with minimal effort. Every line of ActionScript you create will help you understand further the intricacies of the language and how it can be modified to suit your needs. Every line you write will give you more confidence to undertake increasingly complex and rewarding projects.

With this in mind, let's move on to explore ActionScript's common terminology, the programming interface, and how it all fits together to create the effects you desire.

## ActionScript Uncovered

ActionScript is an Object-Oriented Programming (OOP) language that interacts with a movie's objects and controls the movie's playhead. ActionScript is ECMA-262-compliant, which means it conforms to the international standard for JavaScript.

While we will examine many aspects of ActionScript, it's beyond the scope of this publication to cover every built-in function and object. The books listed below cover most of the extended functionality of JavaScript and ActionScript, and are invaluable references that should be a part of any Flash developer's library.

❑ *ActionScript: The Definitive Guide* (2nd Edition, O'Reilly)

❑ *Professional JavaScript* (2nd Edition, SAMS)

Throughout this book, I'll explain why we're using particular pieces of code; where appropriate, I'll also show how similar functions can be carried out by timeline manipulation, and why it's best practice to script them in most circumstances.

Here's a simple example that illustrates the benefits of ActionScripting and indicates how easy it is to manipulate parts of the ActionScript to alter the final outcome.

To animate an object's movement from point A to point B, you could create a movie clip of the object you want to move, create two key frames on the timeline, and tween between them to produce the movement. While this is a legitimate method for creating animation, you can instead script this process in a few lines of simple code.

Assume we have a movie clip named `myMC`. Adding the following ActionScript code to the main timeline moves the movie clip to the right if the horizontal position (`_x`) is less than 500:

```
myMC.onEnterFrame = function ()
{
    if (this._x < 500)
    {
        this._x++;
    }
};
```

Pretty simple, isn't it?

The good thing about scripting animation and other effects is that the scripts can be stored within your own script library (or a shared library), ready for inclusion in your Flash applications whenever you need them. Changing the scripts to suit your needs, rather than altering nested timelines, has obvious benefits in terms of your own productivity, and in helping you develop a reference library. To change the range of motion in the above example, we could simply alter the numeric value from 500 to some other value.

Let's take a quick look at some of the objectives ActionScript can accomplish, bearing in mind that these are just a few of its many uses. In later chapters, we'll cover many more, including some that may surprise you.

- Simple or complex animation effects
- Transitional effects
- Interactive user interfaces
- Menu systems that cannot be produced in any other medium
- Experimental mathematical effects
- Simple or complex interactive games

- ❑ Connection to databases or external files via a middle tier (ASP, ColdFusion, PHP, or Macromedia Data Connection Components)

## ActionScript Dot Notation

ActionScript uses a special method to reference objects that appear on the stage and are embedded within movie clips and buttons. If you're familiar with JavaScript, you'll be at home with the ActionScript Dot Notation for referencing objects within your Flash movies. According to this notation, `document.layer1.form1.checkbox1.value` refers to the value of a checkbox within a form inside a layer contained in the current document.

If you're not familiar with this method of referencing objects, don't worry—it's as straightforward as falling off a bike. You'll soon pick it up!

There are three special keywords in ActionScript that can be used to address objects on the stage:

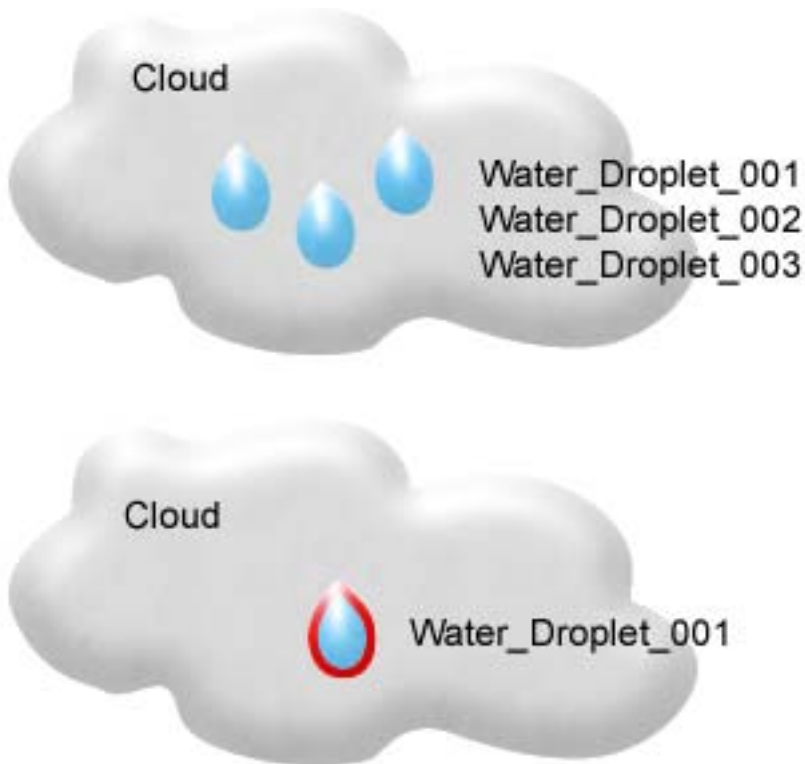
- this** refers to the actual object to which the script is attached (i.e. itself)
- \_parent** refers to the movie clip that *contains* the object to which the script is attached
- \_root** refers to the main Flash movie, which contains the entire hierarchy of objects

In this discussion, let's concentrate on `_root`. Say you create a Flash movie that contains a movie clip called `Cloud`, which in turn contains another movie clip called `Water_Droplet_001`. To find the `alpha` property value (opacity) of the `Water_Droplet_001` clip, you could use the following ActionScript expression:

```
_root.Cloud.Water_Droplet_001._alpha
```

How does this code work? Have a look at Figure 1.2 below. Notice that the water droplets are contained within the cloud (i.e. they're children of the parent cloud). We reference a single water droplet—for the sake of simplicity, the first water droplet (`water_droplet_001`)—within that cloud. The reason we use the term `_root` before the object is to tell Flash to start looking from the very top of the movie hierarchy within the stage.

**Figure 1.2. Understand the impact of the ActionScript dot notation.**



`_root.Cloud.Water_Droplet_001._alpha`

Grabbing this value of the `_alpha` property of the `Water_Droplet_001` movie clip, and storing it in a variable named `Water_Droplet_001_Alpha`, is as simple as this:

```
Water_Droplet_001_Alpha = _root.Cloud.Water_Droplet_001._alpha;
```

Objects placed on the canvas can have many properties controllable both from within, and outside, the objects themselves—all of which adds to the fun of experimentation and learning.

# Actions Panel Unraveled

Actions are pieces of ActionScript code that perform specified functions. Functions that are commonly used to create basic actions within Flash include:

<b>getURL</b>	Sends the browser to an HTML page you specify
<b>stop</b>	Stops the playhead of the timeline
<b>play</b>	Starts the playhead of the timeline
<b>gotoAndPlay</b> <b>gotoAndStop</b>	Jumps the playhead of the timeline to either a labeled frame or frame number

These core ActionScript functions are among the most common you'll use to navigate to different parts of a movie. You may already have encountered them while dabbling in Flash 5 or later versions of the software. This really is just the tip of the iceberg when it comes to ActionScript. As you'll see later, it offers numerous ways to increase interaction and add special effects to your work.

ActionScript is added to the stage via the **Actions Panel**. If you've ever created a scripted animation, or any effects that use these basic methods, you will have seen the Actions Panel. In this book, it will be the single most important weapon in your arsenal, so let's go through it now to make sure you understand everything this tool places at your disposal. You can access the panel through Window > Development Panels > Actions, or by hitting **F9**.

**Figure 1.3. The Actions Panel in Flash MX 2004 is your single most important weapon.**

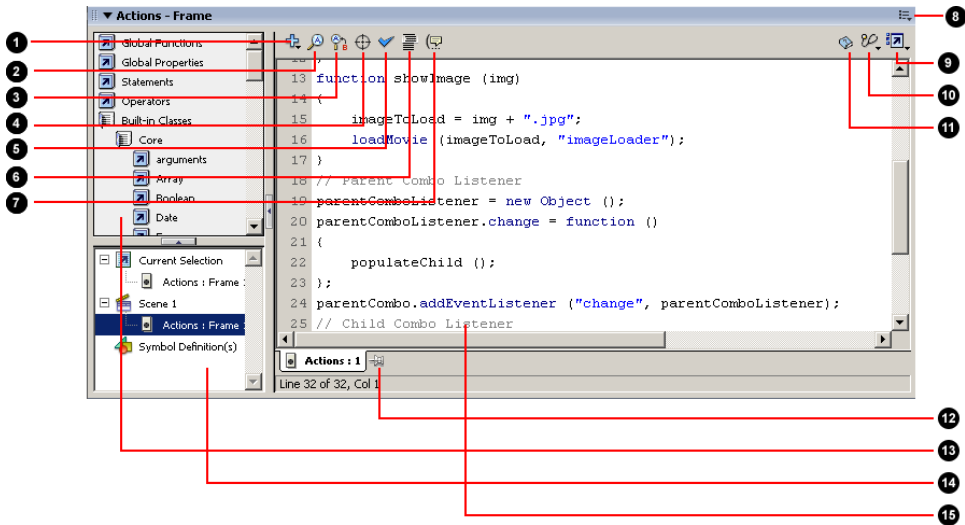


Figure 1.3 shows the Actions Panel with the following components indicated:

- ❶ Add a new item to the script window
- ❷ Find code
- ❸ Find and replace (very useful)
- ❹ Insert target path browser
- ❺ Check syntax
- ❻ Auto-format code (you can change formatting preferences from the Options menu)
- ❼ Show code hint (when at a relevant point)
- ❽ Options menu (useful in configuring code hinting, code colors etc.)
- ❾ View options (line numbers, word wrap, Esc shortcut keys)
- ❿ Debug options (see Chapter 9)
- ⓫ ActionScript reference
- ⓬ Pin active script within Actions Panel
- ⓭ Actions toolbox

- 14 Script navigator
- 15 Script pane

## Actions Panel Framework

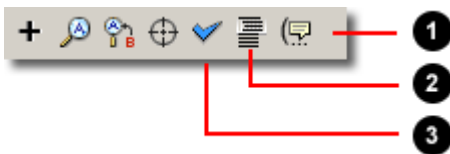
The Actions Panel doesn't offer Expert and Normal modes in Flash MX 2004 or Flash MX 2004 Professional. In earlier versions, the Actions Panel could be run in one of these two modes, toggled to from its Options menu. In Normal mode, actions could be inserted only via the menu system; Expert Mode allowed the developer to enter actions by typing directly into the Actions Panel itself. In the 2004 editions, you can add ActionScript to the Script pane only through direct input, the Actions toolbox, or the + button above the Script pane.

If you're still "finding your feet" when it comes to the Actions Panel, you aren't left completely on your own. Flash's code hinting functionality will attempt to complete your code automatically, or offer you a choice of syntax based on what you've already typed. For example, if you type "Math" within the Actions Panel, you'll be presented with a list of methods that are available for the `Math` class, ordered alphabetically from `abs` to `tan`. You can then either select the method you require from the drop-down menu or continue typing the script yourself.

## Optimizing the Actions Panel

One of the important elements of working efficiently within Flash, and decreasing the development time of scripted effects, is an efficient and comfortable working environment. Now, I'm not about to tell you to vacuum your room and polish your desk (although I have to admit that before I sit down to a scripting session, I do sort my papers out and try to clear up my workspace). But remember: a tidy environment reflects a tidy mind, and tidy code is easier to understand when you come back to it at a later date.

**Figure 1.4. Get to know the important options within the Actions Panel.**



There are three major options available within the Actions Panel that are great time-savers, and will aid you in your quest for coding perfection. The buttons for these are found in Figure 1.4:

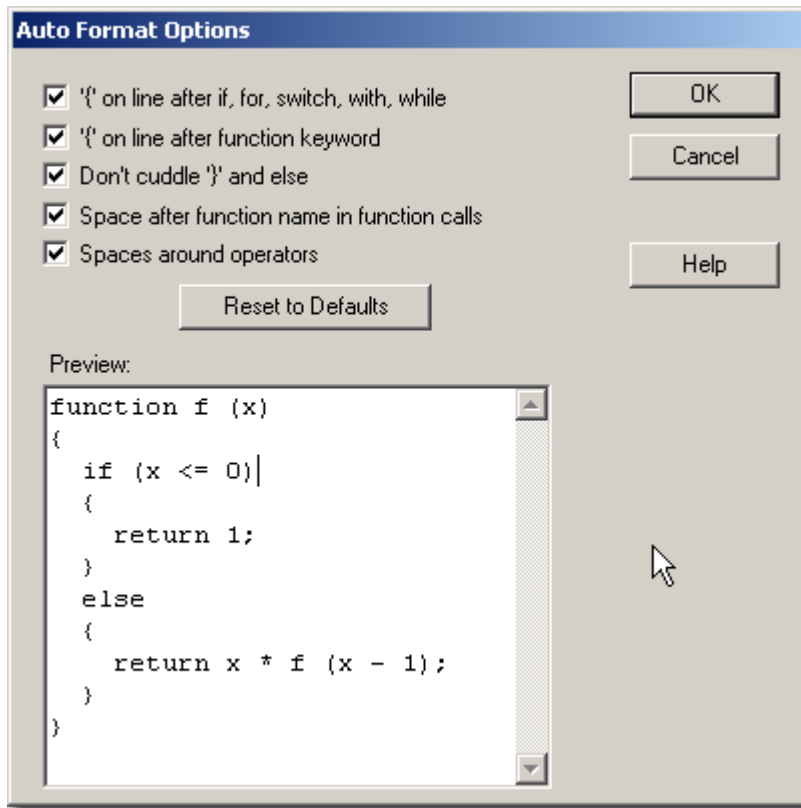
- ❶ **Show Code Hint:** When you type directly into the Actions Panel and Flash suggests syntax to complete your code, you'll note that if you don't accept this suggestion, after a short time it will disappear from the screen. At other times, you may be happily coding away, only to have the ActionScript gremlins run in and steal thoughts from your brain—suddenly, you forget what you were typing! In either event, click this button to have the program display a code hint relative to your current position in the script.
- ❷ **Auto Format:** After heavy code editing, additions, and deletions, the comments and formatting within your code can get a little out of kilter. A simple click of the Auto Format button will format your code beautifully, based on either preset rules or custom rules you set up, as described in the next section. This is one of the most frequently clicked buttons in my copy of Flash—it's a welcome addition to Flash MX and later versions.
- ❸ **Check Syntax:** When you're examining the impact of a piece of script on the final movie, the last thing you want to see is an output window full of errors. To check your code for errors without either exporting the movie or viewing it within the internal Flash Player, click on the Check Syntax button (**Ctrl-T**, or **Command-T** on Mac). This will report any errors immediately, which can save a lot of time in the initial development phases.

## Configuring Auto Format

Coming back to a piece of code you developed six months earlier can be a pleasant experience if your code is effectively formatted and commented; if not, it can be a veritable nightmare.

The Auto Format button helps keep your code in the best possible condition. While it doesn't comment code for you, it does make code readable—definitely a bonus when you're revisiting libraries of compiled code. To alter the Auto Format options to your liking, select Auto Format Options... from the Options drop-down menu in the corner of the Actions Panel. To keep your code in pristine condition, I suggest you check all the checkboxes, which will produce a similar result to that shown in Figure 1.5.

**Figure 1.5. Adjust the Auto Format options to generate cleaner code.**



## Code Commenting

There comes a time during every project, be it large or small, when you return to a line or chunk of code and think ‘Hmm... what does that bit do again?’ There’s nothing worse than having no idea what your code does, and needing to work through it all again to make alterations or reuse it in another project. Commenting will make your code more readable, easier to navigate, and, if you work as part of a team, more easily transferred or shared via common libraries.

*note*

Comments in ActionScript are not executed when a movie is running; in fact, Flash completely ignores them. They're for *your* benefit, used to describe your code or temporarily disable sections of it.

As an example, here's a snippet I include as the first line of every function I write. I save this as a text file in the ActionScript library on my hard drive so that I don't need to retype it each time I use it. It contains a wealth of information that makes it easy for me (or any other developer) to see what the code does, what variables it will accept and/or output, and relevant revision data:

```
//-----
//Function Name: Name of Your Function
//Function Date: 15/09/2003
//Input:         Input Variables (e.g Color Variable (String))
//Output:        Output Variables (e.g Hex Color Reference)
//Revision Date: 28/09/2003: Added Non Numeric Error Checking
//-----
```

A double slash (//) tells Flash to ignore the remainder of a line of code. In the example above, each line of the comment begins with a double slash. You can also use this trick to temporarily disable a line of code, like so:

```
// var TestVar = _root.parentMovieClip.childMovieClip.value;
```

You can create multiline comments by starting them with /\* and ending them with \*/. Here's what the function description would look like using that style of comment:

```
/*-----
Function Name: Name of Your Function
Function Date: 15/09/2003
Input:         Input Variables (e.g Color Variable (String))
Output:        Output Variables (e.g Hex Color Reference)
Revision Date: 28/09/2003: Added Non Numeric Error Checking
-----*/
```

This style of comment can be used to “comment out” entire blocks of code, like this:

```
/*
_root.Option_Button.onPress = function ()
{
    getURL ("http://www.google.com");
};
*/
```

Taking a little extra time to comment code will save you and others time in the long run. Once you get into the habit of commenting, you'll wonder how you ever got by without it.

## ActionScript Coding Tips

There are a few key points to remember about creating functions and actions within the Actions Panel. These are by no means the be-all and end-all of best coding practices within Flash; they're merely standards that will see you spend more time developing, and less time searching for and squashing bugs.

### ActionScript is Case-Sensitive

Just like the JavaScript on which it's based, ActionScript will not work as expected if it's written in the wrong case. But there are some exceptions. For example, consider the following function:

```
function TestTrace () {  
    trace ("hello");  
}
```

This function is completely error-free, and will output 'hello' to the Output Window when it's called. Now, even though the function name is capitalized (`TestTrace`), you can call the function without the capitalization (`testtrace`), and Flash won't complain:

```
myButon.onRollOver = function ()  
{  
    testtrace ();  
}
```

*note*

In Flash MX, the ActionScript compilation routine is quite forgiving—it's not case-sensitive. ActionScript will be treated as case-sensitive if you include `#strict pragma` as the first line of a script.

Just because ActionScript is forgiving with your own function and variable names, don't expect it to do you the same favors when it comes to built-in names. These mistakes will stop your code from executing and are often the most annoying and difficult to find as you introduce more code into your project. For this reason, the best policy is to always take care to use the correct capitalization.

## Externalize ActionScript

Externalizing large blocks of ActionScript is a good way to separate the elements of your project and help keep your code tidy and concise. Imagine, for example, that we have a piece of navigation code that's 50 lines long. We could externalize it by cutting and pasting the code chunk into Notepad (Windows) or BBEdit (Macintosh), and saving it as `navigation.as`. We would then replace the code in the movie with this line:

```
#include "navigation.as"
```

This `.as` file would then be included in the SWF file at compilation time and need not be distributed with it.



To quickly export your ActionScript into an external file, select **Export Script...** from the Options menu within the Actions Panel.

## Script Pinning

Because you can associate ActionScript actions with almost any element of a Flash movie, you may be working on one script fragment and need to refer to another somewhere else. Clicking the other element will display that other fragment; if you need to return to the fragment you were working on, you could easily lose your place.

To help developers avoid confusion while jumping between multiple script fragments, Macromedia has provided script pinning. Simply click the **Pin Script** button in the Actions Panel, and Flash will keep the currently displayed script fragment open in a tab along the bottom of the Actions Pane. You can then navigate to another element to view its script without fear of losing your place in the original script—simply click the tab to switch back to the pinned script whenever you need to. You can even pin multiple script fragments when the situation calls for it.



Flash MX 2004 introduces an extra pane in the Actions Panel—the script navigator pane. Located just below the Actions toolbox, this new pane allows easy navigation through all the code in a movie.

## Lift and Lock

Actions can be added to the timeline within keyframes, or assigned to objects on the stage. Regardless which type of action you're creating, however, it's best practice to create a new layer, give it a meaningful name such as 'ActionScript', and lock the layer before placing your ActionScript in it. This way, you can easily see the objects containing ActionScript and navigate to them without having to open the Movie Explorer.

## Naming Conventions

When you write code, create functions, or name layers and folders, there is a set of best practices that you should try to implement. Some of these are common sense; others are not so obvious. None of these conventions is essential, but by following these simple guidelines and other suggestions in this chapter, you should be able to organize your Flash projects successfully.

Do	Do Not
<input type="checkbox"/> create filenames that relate to the purpose or goals of the Flash project	<input type="checkbox"/> use acronyms that make little or no sense
<input type="checkbox"/> create names for functions and variables that indicate their roles in the project	<input type="checkbox"/> shorten filenames
<input type="checkbox"/> use underscores to separate words; avoid hyphens	<input type="checkbox"/> create function, file, or variable names that require a Masters degree in Particle Science to decrypt
<input type="checkbox"/> be consistent in your application of the naming system you decide to use	<input type="checkbox"/> use spaces or merge words together

## The Timeline, Simplified

In our reality, time can only go forward (though some may argue otherwise!); the same is not true in Flash. The playhead within the timeline can move forward and backward—it can even jump frames. The Flash timeline is integral to creating effects, and all Flash movies use the timeline, even if only for the first frame. The introduction of Timeline Effects in Flash MX 2004 has made it very easy to create rapid animations and transitional effects within your projects. With basic Action-

Script, you can start the playhead, play an animation, and send the playhead back to the beginning of the timeline to play again. And, with only slight modification, you can add an auto-incrementing counter to either conditionally jump to another point on the timeline or return to the start of the timeline. It's simple to create basic effects and build them up into something extraordinary.

Timeline Effects also make it easier to develop complex effects that are transferable to other projects or collaborators. Indeed, this concept of scalability should always be considered at the start of a project. Before you begin, ask yourself, "Will I easily be able to use this effect again in the same movie, or another project, without extensive recoding?" If the answer is no, you need to identify ways to make it scalable.

The easiest path to scalability is to invest a little time in scripting common tasks. If you can animate an object using four lines of ActionScript that represent a reusable function or snippet, go for it! This approach makes for easier maintenance than a script containing multiple duplicated, nested movie clips in which the variables may clash.

Ask yourself these questions before creating a new effect or starting any project:

1. Will multiple effects occur on the stage at the same time?
2. Will I use the same effect more than once in the movie?
3. Can I script the effect I wish to create?
4. Can I make the effect easily portable between projects?
5. Can I easily add new functionality to the effect?
6. If the clients or I don't like the effect, can I easily tweak it?

You can create complex effects using the timeline—in fact, before the release of Flash 5 and later versions, this was the only way to do so. Even today, most of the animation effects we see rely on timeline tweening between frames to create animation, movement, and opacity fading. The combination of Timeline Effects and ActionScript can be a powerful one, but this approach also has a technical advantage. Effects developed this way are more 'tweakable;' changing a few settings here and there impacts the final product considerably in very little time.

The importance of the timeline should not be underestimated. Every single object, from layers to movie clips, is present within the timeline when it's added to the

stage. Take some time to become familiar with the best practice principles for working with the timeline.

## Label Me Up

After a weekend or holiday, how long does it take you to come to terms with the Flash projects you left behind? Organization is the key to minimizing this “re-familiarization” time. There’s no excuse for sloppy code or disorganized Flash projects. Invest a little extra time at the start of a project, add elements with care, and you’ll reap the rewards when you return to it after time away or hand it over to your colleagues.

I advocate the creation of a layer named ‘Labels’ to hold commentary, conditions, and pointers as to what’s happening at each step on the timeline. This process is as simple as selecting a frame from the timeline, and adding a Frame Label within the Property Inspector. Maintaining these rolling labels allows you to see what’s happening within your movies—you can even add small notes, for example, to remind yourself that code requires alteration, or graphics need to be swapped out.



Flash MX 2004 allows you to efficiently alter the frame label type. To change a name, label, or anchor’s representation within the timeline, simply click the drop-down box that appears beneath the frame label within the Property Inspector.

Adding a Labels layer keeps this information separate and helps you organize your thoughts regarding timeline planning.

## Organize Your Layers

In Flash version 5 and earlier, users had the ability to organize work into distinct layers within the timeline. This facility was extremely helpful for separating out design components within the stage. A little planning and organization of these layers can benefit you both during the initial development process and when re-visiting a project at a later date.

In Flash MX and Flash MX 2004, you can organize these layers into discrete folders. Let me show you how this seemingly small enhancement can improve your workflow.

Let’s say you’ve been working for a while on a project that has many movie clips, buttons, and bitmaps on the stage. You’re using many layers, though they’re not

organized into any particular format, and you're scrolling up and down the timeline window to access them as you go. To make matters worse, the project is growing more complicated, which means more layers and even more scrolling. Fear not—this is all about to change! Figure 1.6 shows how you can create and organize folders into discrete compartments, placing relevant layers within each. When you finish editing a folder, just click the arrow next to it to roll it up and give yourself more space within the timeline. As the projects you undertake become more involved and larger, the improved efficiency you can gain from this approach will become self-evident.

In adding components to a project, I usually start with a standard framework. I have a folder containing two layers—one for ActionScript, the other for labels. I have another folder for Animations, which is then subdivided into relevant sections, and a folder for static imagery. Obviously, you can change this structure to suit your development needs and personal preferences.

**Figure 1.6. Organize folders and layers with ease in Flash MX and Flash MX 2004.**



## Timeline Effects (New in Flash MX 2004)

New within Flash MX 2004 is a much-anticipated addition to the arsenal of experienced programmers and non-programmers alike: Timeline Effects. These are available from the Insert menu after you've selected an object from the stage. You can access them at: Insert > Timeline Effects > Effect.

There are several effect categories, each containing many possible choices:

- Blur
- Expand
- Explode
- Fade In
- Fade Out
- Fly In
- Fly Out
- Grow
- Shrink
- Spin Left
- Spin Right
- Wipe In
- Wipe Out

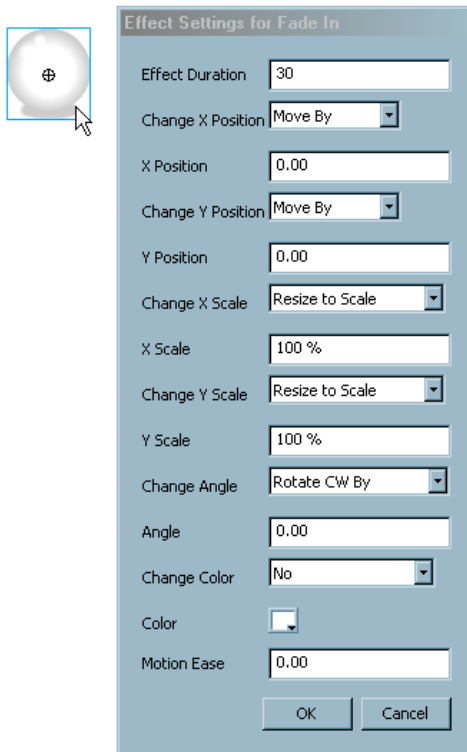
We'll cover these in depth, and see how they can be used to create engaging effects, in Chapter 4.

Traditionally, to fade the opacity of an object from completely transparent to completely opaque, I would access the timeline and tween between two instances of a movie clip, altering the alpha values accordingly. Though functional, this technique can be quite time-consuming and fiddly, especially if you're looking to build an effect with multiple movie clips fading in, in a stepped manner, to produce a blur. The addition of Timeline Effects provides developers a large degree of control over setup parameters. With so many of these at your disposal, it's quick and easy to create compositions for yourself or for presentation to clients.

Figure 1.7 shows that many properties are available within the Fade In command. Straight out of the box, the effect can be applied and previewed within a matter of seconds, shaving considerable time off the old techniques, especially if you're using multiple iterations of the same effect. Combining Timeline Effects often

produces unexpected yet pleasing results. The key to creative success is, as always, to experiment until you're happy with the outcome.

**Figure 1.7. Edit the Fade In Timeline Effects parameters within Flash MX 2004.**



## Wrapping Up the Basics

We've covered the basics of ActionScript and the Timeline, and discussed best practices for both. Now it's time to apply these tactics to create stunning effects that are appealing, easily transferable among projects, and fun to explore and expand. In the next few chapters, we'll cover a range of subjects, with examples, suggestions, and explanations of techniques that will prove invaluable in your day-to-day Flash projects.



# 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 ActionScript.

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 your 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

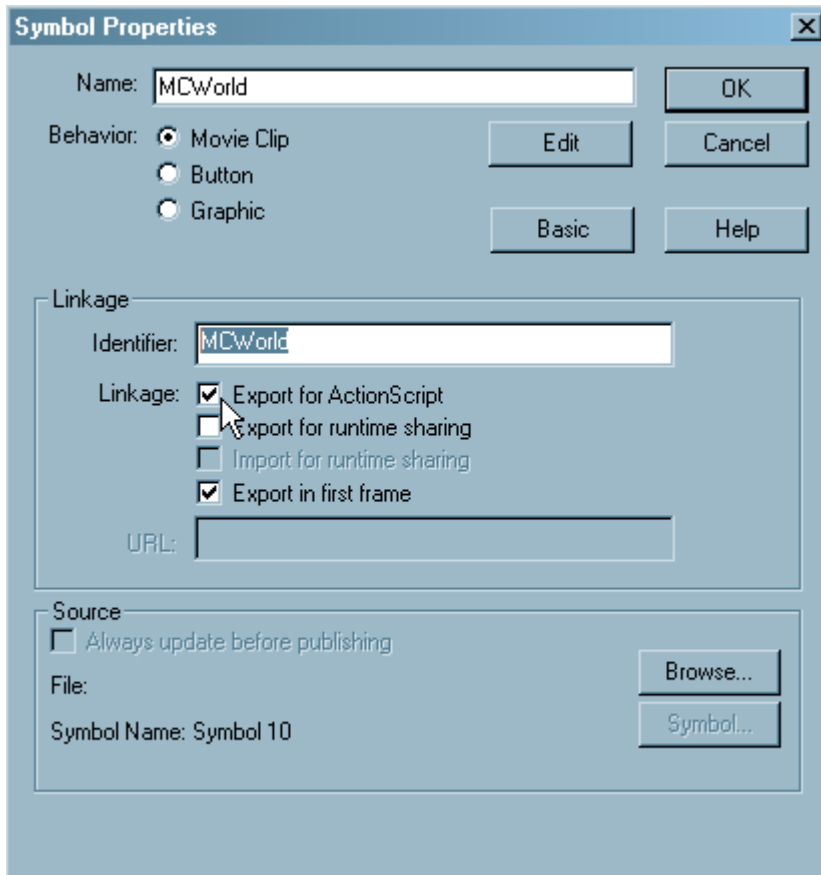
First, you'll need to create a new Flash movie to showcase your effect.

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.**



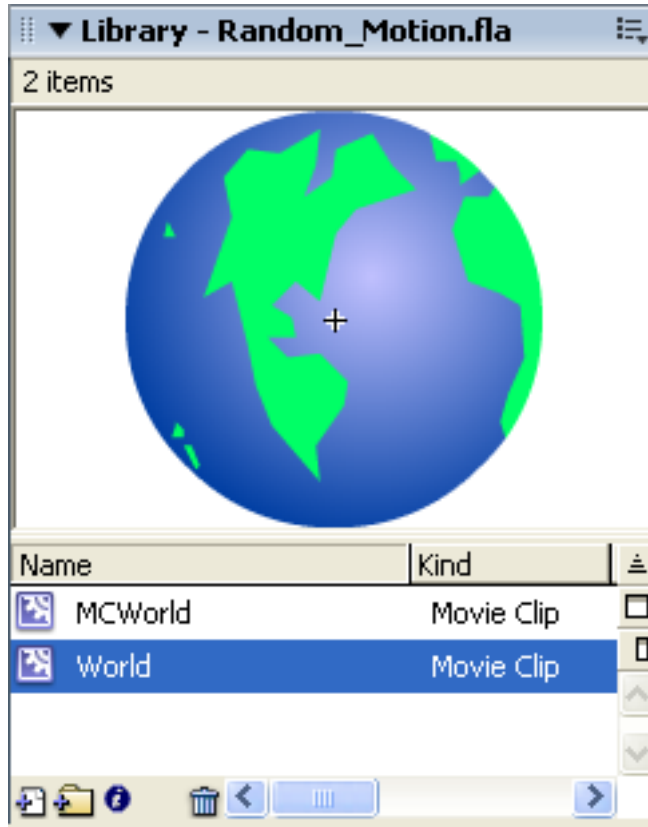
*note*

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;  
}
```

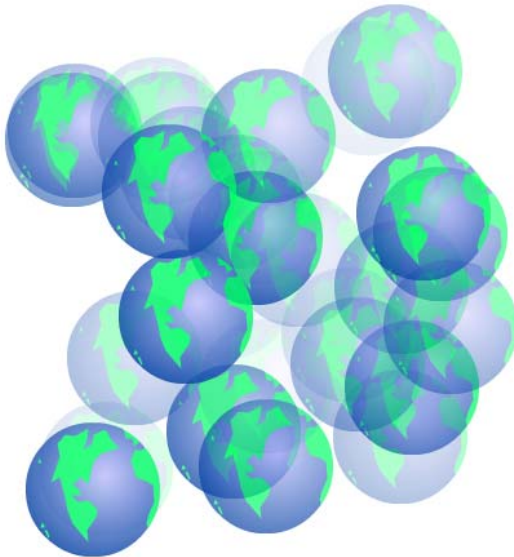
We could instead have used an `onEnterFrame` event handler to do this, but `setInterval` allows us to define the speed of the motion independent of the movie's frame rate, which can be useful in many circumstances.

## 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:

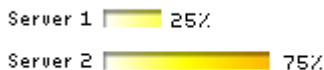
```
var nervousness = 2;  
var xShift = random(nervousness * 2 + 1) - nervousness;  
var yShift = random(nervousness * 2 + 1) - nervousness;
```

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.

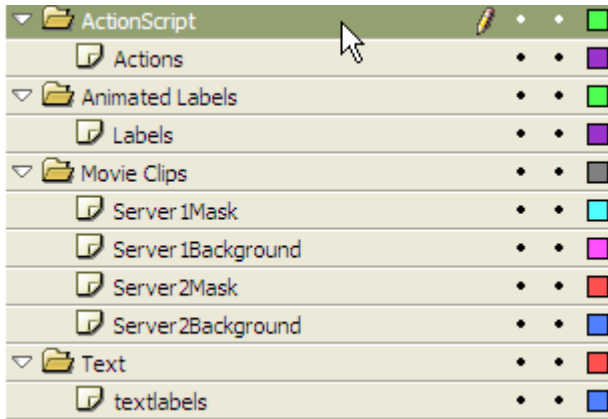
**Figure 3.4. This simple scripted masking effect animates a mask between two points.**



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.

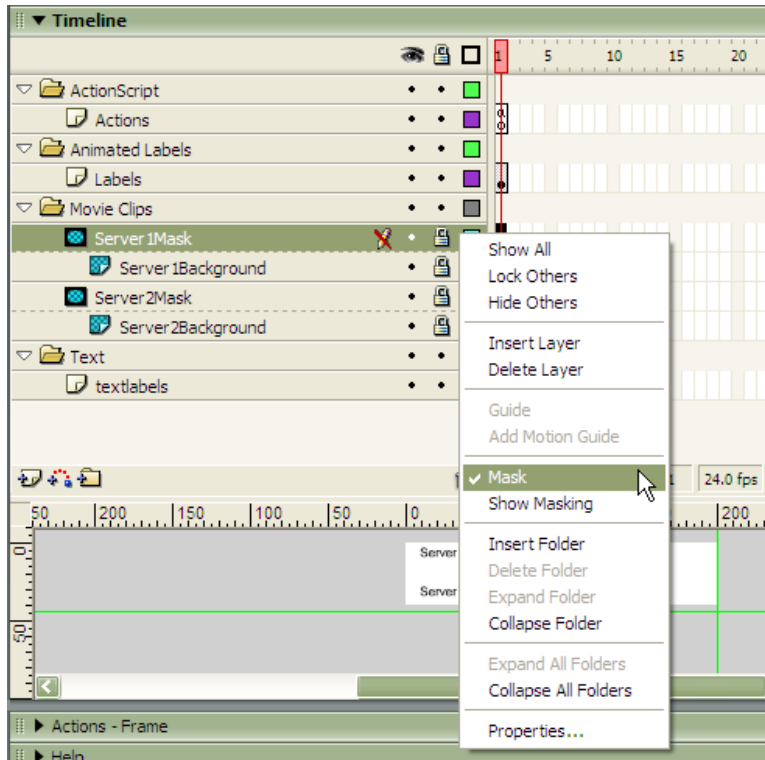
Now that we've created the backgrounds, we can build the masks we'll control via ActionScript:

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:

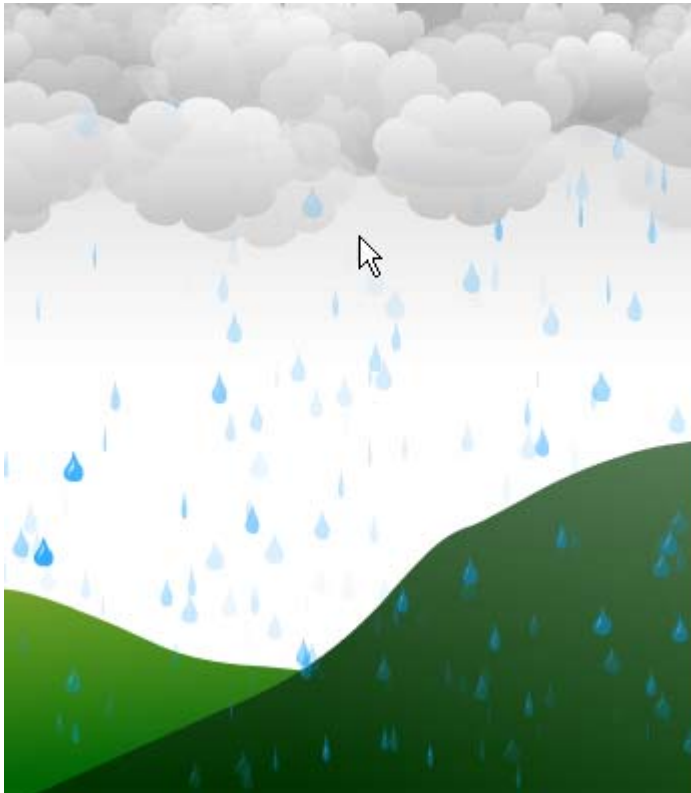
- Our `for` loop counts to 100, instead of 50, so we'll create twice as many duplicates.
- We create duplicates of the `raindropSlow` clip this time.
- We use a new counter variable for this loop (`j`), and add it to the count at the end of our previous loop (`i`) when setting the stacking order of the 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.

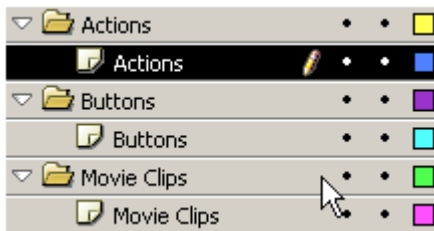


If you don't feel like creating this effect from scratch, locate `User_Drive_Motion.fla` in the code archive.

## 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.



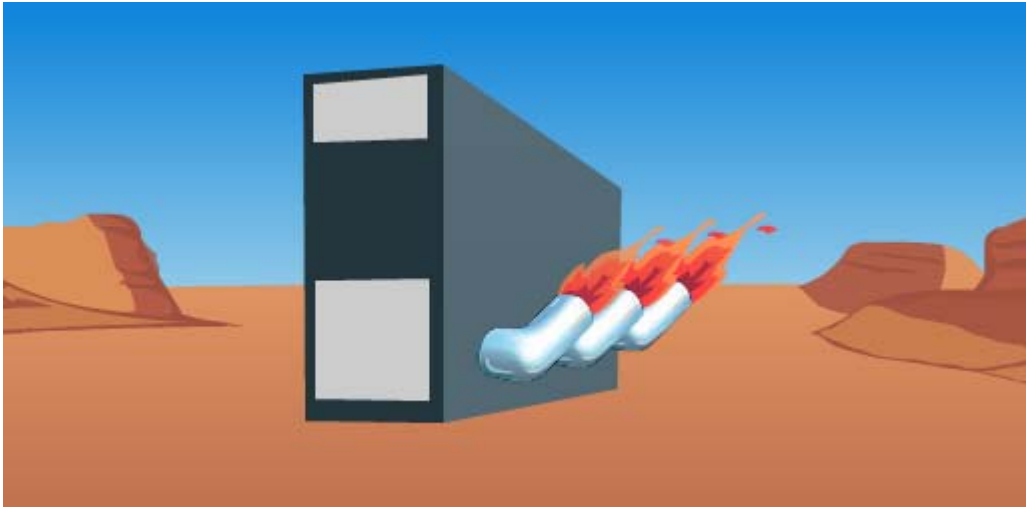
### Looks like a menu to me!

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.fl1a` 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!



# 5

## Sound Effects

---

Sadly, sound within Flash projects is often included as an afterthought—if it’s included at all. Hurried, last-minute sound additions typically amount to no more than a clichéd loop that serves little purpose. If you ever find yourself in the position of importing a sound clip into a project without a clear idea of why you’re doing it, stop and repeat the Flash designers’ mantra: *bad use of sound can destroy the entire user experience.*

In this age of digital media, sound can and should be a part of the online environment. A little sound can go a long way toward enhancing and solidifying the user experience. Even a simple “click” sound can provide valuable auditory feedback, provided it’s used in the right manner. But beware overloading your projects with sound “just for the sake of it.” The use of sound, then, is a question of balance.

## When Should You Use Sound in Your Projects?

This is one of those golden questions, and the answer depends on the type of project you’re developing. Some productions don’t require any sound when viewed in isolation; yet, when incorporated into a larger project, they call for some degree

---

of aural enhancement. If a project is heavily related to sound, then it should definitely involve an appropriate variety of audio clips.

Always consider your work from the perspective of users: will the addition of sound increase their enjoyment of the project? Including short or long musical loops, sound bytes, or a narrative can greatly increase the appeal of an interface, but may also detract from other content and become extremely annoying after just a short while. Consider the following examples of possible Flash projects and think about sound might benefit them. Your choice could include sound effects, musical loops, narratives, or, of course, nothing at all.

- Looping animation
- Single level menu system
- Multilevel menu system
- Multi-paged Flash site
- Flash form
- Streaming video content

Quickly run through these examples in your head. It's not difficult to see what will and won't work for different projects. Over time, you'll become increasingly adept at making decisions about when and how to add sound to your projects.

## Selecting Sound Clips

One of the major benefits of digital sound is its flexibility. Audio processing software such as Steinberg Wavelab<sup>[1]</sup> makes it easy to edit, modify, loop, and add effects to sounds. You can take a ten-second section from one audio loop, paste it into another, or paste it across multiple tracks, with no more difficulty than editing a Word document. The advanced sound processing tools that are now available, combined with a microphone and your own imagination, allow you to create a huge array of audio clips. If you're pushed for time or need extra inspiration, there are many online resources from which you can download free clips or purchase collections of clips for use in your projects.

---

[1] [http://www.steinberg.net/en/products/audio\\_editing/wavelab/](http://www.steinberg.net/en/products/audio_editing/wavelab/)

The application of sound to your projects should not be taken lightly. You need to pay attention to a number of factors if the effects are to be successful—most notably, timing, volume, and composition. Remember, you’re seeking to enhance your visual messages rather than work against them, so your choice of sound is important. A manic techno loop attached to the submit button of a Flash form may well provoke visitors to close the browser. On the other hand, if you give users subtle feedback that the form has been submitted, they’ll likely gain confidence in the interface and continue to interact.

As you can see, it’s all about context. The appropriate integration of sound into a project can support the efficient communication of the message.

## Importing and Exporting Sound Clips

Flash MX 2004 offers a built-in sound editor, which is fine for simple editing tasks. However, this tool is too basic for most needs: it doesn’t allow you to add effects to your sound selections, for example. I definitely advocate the use of a more powerful sound processor, especially when it comes to trimming, duplicating, and looping clips.

You can import a sound clip from the File menu just as you would any other media type. Imported sounds can be applied directly to buttons and button states (Up, Down, Over, etc.), incorporated via streaming external files, included through ActionScripted control in the Action Panel, or added via other means.

The addition of a “click” sound to a button’s Down state to simulate a real world mouse click is an example of a basic sound effect. It doesn’t *require* ActionScript knowledge, but it certainly can be created via script. In the following example, the necessary code is applied to a button called `myButton`, which references a sound clip that has a linkage identifier of `mySound` in the Library Panel:

```
myButton.onPress = function ()
{
    newSound = new Sound ();
    newSound.attachSound ("mySound");
    newSound.start ();
}
```

For more advanced effects, including streaming and abstract playback mechanisms, we need to look under the hood of the ActionScript `Sound` class to see how we can control sound from within the interface and code conditions. It’s an exciting

arena in which you can create interesting and dramatic effects quickly, with just a little imagination.

As this book is concentrated on the creation of effects via ActionScript, we won't cover the integration of sound effects into buttons and frames on the timeline. For examples of this, refer to the Flash MX 2004 documentation.

Let's start with an interesting volume control that's linked to an animation.

## Dynamic Volume Control

In the following demonstration, we scale the vertical height of a movie clip dynamically, tying the volume of a sound associated with the clip to its height. It's an important example, as it forms the basis for many of the effects you'll create in future.

To edit a completed version of the effect, locate `volcontrol.fla` in the code archive.

## Setting the Scene

1. Create a new Flash document that's 500 pixels wide and 400 pixels high. Set the frame rate to 24 fps (Modify > Document...).
2. Rename the default layer Actions, and create two new folders in the Library Panel: Movie Clips and Sound.
3. Create a new movie clip symbol in the Movie Clips folder with some arbitrary content. The example in the code archive includes a static text area in which the text "sound effect" appears.
4. Drag an instance of the movie clip to the stage and name it `clipper`.
5. Select File > Import to Library... and either select `reverb.mp3` from the code archive, or import another sound file of your choice. Place it in the Sound folder.

The file used here is a short, three-second clip of medium quality chosen primarily because it's a small file.

6. Select the sound clip from the Library Panel, right-click and select Linkage.... Check the Export for ActionScript checkbox and enter `reverb` as the identifier.

Having added the element we're going to animate and linked the sound clip for ActionScript export, we can now add the control code.

## Adding the ActionScript

7. Select the first frame of the Actions layer and add the following code to the Actions Panel:

```
File: volcontrol.fla Actions : 1 (excerpt)  
MovieClip.prototype.WaveFade = function (minValue, maxValue,  
    period)  
{  
    this.period = period;  
    this.minValue = minValue;  
    this.maxValue = maxValue;  
    this.count = 0;  
    this.onEnterFrame = function ()  
    {  
        var value = (1 + Math.cos (this.count++ * 2 * Math.PI /  
            this.period)) / 2;  
        this._yscale = this.minValue + value *  
            Math.abs (this.maxValue - this.minValue);  
    };  
};  
clipper.WaveFade (20, 100, 48);
```

Here, we extend the movie clip class with a new method called `WaveFade`, which scales the clip up and down vertically in a smooth, regular pattern. The method is relatively straightforward, with the exception that careful use of the cosine function is required to produce the values that control the scaling.

The method takes three parameters: `minValue`, the minimum scale for the clip (as a percentage); `maxValue`, the maximum scale for the clip; and `period`, the number of frames in one oscillation (from the maximum scale to the minimum and back again).

The `value` variable is assigned a value using the cosine function. We add one and then divide the result by two to get values oscillating between zero and one. (Cosine oscillates between one and minus one.) We then multiply that by the difference between the maximum and minimum scaling values to determine the vertical scaling of the clip for each frame.

8. Save and preview your work.

Notice that the `_yscale` of the movie clip oscillates smoothly from its full height down to 20% and then back up again. Now that we have the animation in place, we can add the sound clip:

9. Add the lines shown in bold to the existing code in the Actions Panel:

```
File: volcontrol.fla Actions : 1  
MovieClip.prototype.WaveFade = function (minValue, maxValue,  
    period)  
{  
    this.period = period;  
    this.minValue = minValue;  
    this.maxValue = maxValue;  
    this.count = 0;  
    this.reverb = new Sound ();  
    this.reverb.attachSound ("reverb");  
    this.reverb.start (0, 999);  
    this.onEnterFrame = function ()  
    {  
        var value = (1 + Math.cos (this.count++ * 2 * Math.PI /  
            this.period)) / 2;  
        this._yscale = this.minValue + value *  
            Math.abs (this.maxValue - this.minValue);  
        this.reverb.setVolume (this.minValue + value *  
            Math.abs (this.maxValue - this.minValue));  
    };  
};  
clipper.WaveFade (20, 100, 48);
```

We start by creating a new `Sound` object and attaching the sound from the Library Panel. We start the clip at the beginning (0), and set it to loop 999 times.

Meanwhile, within the `onEnterFrame` event handler, we set the percentage volume of the sound clip using the same calculation we employed for its `_yscale`:

```
this.reverb.setVolume (this.minValue + value *  
    Math.abs (this.maxValue - this.minValue));
```

This is the same code we used to control the clip's `_yscale`; in theory, we could replace it with the following:

```
this.reverb.setVolume(this._yscale);
```

However, we avoid using this code as it's less readable. When you revisit your code at a later date, you might easily become confused if you saw sound properties linked to movie clip properties.

10. Save and preview your work.

Notice that, as the scale of the clip decreases, so does the volume of the sound. As the scale increases, so does the volume. The addition of scripted sound has enhanced this simple effect, which strengthens the message and creates a more powerful experience.

## Dynamic Panning Control

In this example, we animate an object across the screen and alter the panning of sound in accordance with the object's location. Panning involves the movement of a sound in stereo, which can add dynamism to even a single continuous tone. This builds on the previous example to increase your skills with the `Sound` class using a new method: `setPan`.

By default, the panning of a sound is set to 0, but we can use `setPan` to alter that between the limits of -100 (all sound to the left channel) and +100 (all sound to the right channel).

Let's mirror the movement of a simple animation from left to right with an effect that moves the sound from the left channel to the right.

To skip ahead and modify this effect, locate `pancontrol1.fla` in the code archive.

## Setting the Scene

1. Create a new Flash document that's 500 pixels wide and 400 pixels high. Set the frame rate to 24 fps (Modify > Document...).
2. Rename the default layer Actions, and create two new folders within the Library Panel: Movie Clips and Sound.
3. Create a new movie clip symbol in the Movie Clips folder with arbitrary content. The example in the code archive uses a static text area in which the text "Sound Effect" appears.
4. Drag an instance of the movie clip to the stage and name it `clipper`.

5. Select File > Import to Library... and choose `reverb.mp3` from the code archive, or import a sound file of your choice.
6. Select the sound clip from the Library Panel, then right-click and select Linkage.... Check the Export for ActionScript checkbox, and enter `reverb` as the identifier.

## Adding the ActionScript

We're done setting the scene. It's time to add the code to bring it all together.

7. Select the first frame of the Actions layer and add the following code within the Actions Panel. As this is so similar to the previous example, I've highlighted the differences in bold:

```
File: pancontrol.fla Actions : 1  
MovieClip.prototype.WavePan = function (minValue, maxValue,  
    period)  
{  
    this.period = period;  
    this.minValue = minValue;  
    this.maxValue = maxValue;  
    this.count = 0;  
    this.reverb = new Sound ();  
    this.reverb.attachSound ("reverb");  
    this.reverb.start (0, 999);  
    this.onEnterFrame = function ()  
    {  
        var value = (1 + Math.cos (this.count++ * 2 * Math.PI /  
            this.period)) / 2;  
        this._x = this.minValue + value *  
            Math.abs (this.maxValue - this.minValue);  
        this.reverb.setPan (-100 + value * 200);  
    };  
};  
clipper.WavePan (50, Stage.width - 100, 48);
```

We give the method a new name (`WavePan`), we've, set the horizontal position (instead of the vertical scale) of the movie clip to oscillate between the maximum and minimum values, and have the sound pan between the left and right speakers.

In our call to `WavePan`, we pass minimum and maximum values to ensure that the movie clip covers most of the stage:

```
clipper.WavePan (50, Stage.width - 100, 48);
```

All that remains is to test-drive the effect!

8. Save your Flash document and preview your work.

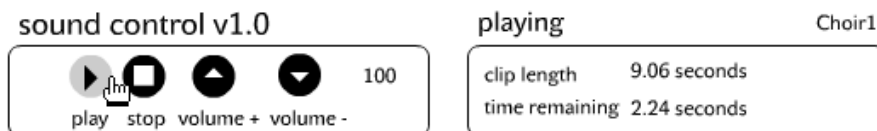
Move your speakers apart or put your headphones on, so you can enjoy this effect to the full.

This simple effect can serve multiple purposes within your projects. It can really bring your creations to life, adding pizzazz to animations, navigation systems, and more!

## Mini Sound Player

It's time to stretch our legs! We've covered the fundamentals of sound control. Now, let's move into the development of a fully functional sound playback device. In this example, we'll create a random sound player that provides visual feedback about various properties of the random sound clips it plays, as shown in Figure 5.1.

**Figure 5.1. Create a mini sound player.**



To jump forward and edit this effect, locate `miniplayer.fla` in the code archive.

## Setting the Scene

1. Create a new Flash document that's 450 pixels wide and 100 pixels high. Set the frame rate to 24 fps (Modify > Document...).
2. Create three new layers, naming the top layer Actions, the middle Elements, and the bottom Background.

3. Select the Background layer and create a background element that will surround the controls, as shown in Figure 5.1. Lock the Background layer.
4. Create four buttons (as symbols in the Library) that visually reflect the following functions: Play, Stop, Increase Volume, and Decrease Volume. Refer to `miniplayer.fla` in the code archive or Figure 5.1 for examples.
5. Drag instances of the buttons into the first frame of the Elements layer, naming them `PlayClip`, `StopClip`, `VolumeUp`, and `VolumeDown`. Arrange the buttons so they sit within the bounds of the background you created in step 3.
6. Add a static text box beneath each of the buttons to identify the tasks they perform. In the example file, I labeled them `play`, `stop`, `volume +` and `volume -`, respectively.
7. Locate the following files in the code archive and import them into your library (File > Import to Library...): `choir1.mp3`, `choir2.mp3`, `choir3.mp3`, `choir4.mp3`, and `choir5.mp3`.
8. Select each sound clip within the Library Panel, then right-click and select Linkage.... Check the Export for ActionScript checkbox and accept the default identifier value (which should be `choir1`, `choir2`, etc.).

As I've explained before, this allows us to access the sound clips dynamically without having to drag them onto the stage.

We've successfully created the objects and imported the sounds we need to achieve basic functionality. All that remains is to add the ActionScript that controls the application.

## Adding the ActionScript

First, we'll add the code that controls the playback of the sounds we imported.

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

```
File: miniplayer.fla Actions : 1 (excerpt)  
function randomBetween (a, b)  
{  
    return Math.min (a, b) + random (Math.abs (a - b) + 1);  
}
```

```

PlayClip.onPress = function ()
{
    stopAllSounds ();
    _root.orchestra = new Sound ();
    _root.orchestra.attachSound ("choir" + randomBetween (1, 5));
    _root.orchestra.start ();
};

```

As the music player plays clips at random, our familiar `randomBetween` function comes in handy.

As the heart of this example, the `onPress` event handler of the `PlayClip` button definitely merits further study. The first thing it does is stop any sound that may already be playing, so that multiple sound clips aren't played at the same time if the `PlayClip` button is clicked repeatedly. The built-in `stopAllSounds` function handles this:

```
stopAllSounds ();
```

Next, we create a new `Sound` object and store it in a variable in the main movie (`_root`) called `orchestra`. We then use `attachSound` to load one of the five sounds from the library and start it playing:

```

_root.orchestra = new Sound ();
_root.orchestra.attachSound ("choir" + randomBetween (1, 5));
_root.orchestra.start ();

```

10. Save and preview your work.

Press the Play button and a randomly selected clip will begin to play; press it again and another random clip will play. It's all functioning as expected.

Now, we'll add the Stop button's control code. This is much the same as the code we used with the Play button to remove the movie clips and stop all sound on the stage:

11. Add the following code beneath what you already have:

```

File: miniplayer.fla Actions : 1 (excerpt)
StopClip.onPress = function ()
{
    stopAllSounds ();
    _root.orchestra = null;
};

```

We use `stopAllSounds` to stop any playing sounds, and throw away the `Sound` object we'd stored in the `orchestra` variable. This frees up the memory the object was using.

12. Save and preview your work.

When you press the Play button, the sound plays; when you hit the Stop button, it stops. Let's add the volume control buttons and finish this example.

13. Insert the following code beneath what you already have:

```
File: miniplayer.fla Actions : 1 (excerpt)  
VolumeUp.onPress = function ()  
{  
  if (_root.orchestra.getVolume () < 100)  
  {  
    _root.orchestra.setVolume (_root.orchestra.getVolume () +  
      10);  
  }  
};  
VolumeDown.onPress = function ()  
{  
  if (_root.orchestra.getVolume () > 0)  
  {  
    _root.orchestra.setVolume (_root.orchestra.getVolume () -  
      10);  
  }  
};
```

When the volume of a sound clip is less than 100, we increase it in increments of ten each time the `VolumeUp` button is pressed. Conversely, when its volume is greater than zero, it's decreased in steps of ten each time the `VolumeDown` button is pressed.

14. Save and preview your work.

You can select a random clip by pressing the Play button, stop the dynamically loaded clip with the Stop button, and increase or decrease the volume of the clip using the volume controls. You may want to experiment with sound clips of your own, as those distributed in the code archive are quite short.

## Modifications

We've successfully created an interface for the playback of random sound clips! We can extend it easily to provide visual feedback about the duration of the loaded track, the play time remaining, the name of the playing clip, and the volume level.

To edit these additions, locate `miniplayer-feedback fla` in the code archive.

15. Working from the previous example, add a new layer below the Actions layer, and name it Labels. This will hold our text labels, which provide feedback about what's going on.
16. Select the first frame of the Labels layer and create a new dynamic text field to the right of the volume buttons, as depicted in Figure 5.1. Name the instance `volume`.
17. With the text box selected, click Character... in the Property Inspector and select Basic Latin (All Characters in Flash MX). Click OK.
18. Referencing Figure 5.1 for placement, create the titles `clip length` and `time remaining` using static text areas.
19. Insert three dynamic text fields beside the relevant titles, naming the fields `clipname`, `cliplength` and `timeleft`. Repeat step 3 for each of the text boxes. Make sure each box is large enough to hold the required information.
20. Select the first frame of the Actions layer and replace the existing code with the following. The changes are extensive, but once again I've highlighted them in bold:

```
File: miniplayer-feedback fla Actions : 1
function roundNumber (toRound, numDecimals)
{
  return Math.round (toRound * Math.pow (10, numDecimals)) /
  Math.pow (10, numDecimals);
}
function randomBetween (a, b)
{
  return Math.min (a, b) + random (Math.abs (a - b) + 1);
}
root.createEmptyMovieClip ("tracker", 1);
PlayClip.onPress = function ()
```

```
{
  stopAllSounds ();
  _root.orchestra = new Sound ();
  var clipnumber = randomBetween (1, 5);
  _root.orchestra.attachSound ("choir" + clipnumber);
  _root.orchestra.start ();

  _root.tracker.onEnterFrame = function ()
  {
    _root.timeleft.text = roundNumber (
      (_root.orchestra.duration - _root.orchestra.position)
      / 1000, 2) + " seconds";
  };
  _root.cliplength.text = roundNumber (
    _root.orchestra.duration / 1000, 2) + " seconds";
  _root.clipname.text = "Choir" + clipnumber;
  _root.volume.text = _root.orchestra.getVolume ();
};
StopClip.onPress = function ()
{
  stopAllSounds ();
  _root.orchestra = null;
  _root.cliplength.text = "";
  _root.clipname.text = "";
  _root.timeleft.text = "";
};
VolumeUp.onPress = function ()
{
  if (_root.orchestra.getVolume () < 100)
  {
    _root.orchestra.setVolume (_root.orchestra.getVolume () +
      10);
    _root.volume.text = _root.orchestra.getVolume ();
  }
};
VolumeDown.onPress = function ()
{
  if (_root.orchestra.getVolume () > 0)
  {
    _root.orchestra.setVolume (_root.orchestra.getVolume () -
      10);
    _root.volume.text = _root.orchestra.getVolume ();
  }
};
};
```

Let's dissect the changes we've made, so we understand how it all works!

First, we introduce a new math function, `roundNumber`, which rounds figures to a specified number of decimal places. This will be used to show both the play time remaining for each clip and the clip's total duration.

```
function roundNumber (toRound, numDecimals)
{
  return Math.round (toRound * Math.pow (10, numDecimals)) /
    Math.pow (10, numDecimals);
}
```

We'll be updating the track time remaining with each frame of the movie, so we'll need an `onEnterFrame` event handler. Unfortunately, we don't have any obvious movie clip to hook that handler to. We could attach it to one of the buttons in our interface, but it's tidier to create an empty movie clip called `tracker` to host it:

```
_root.createEmptyMovieClip ("tracker", 1);
```

We then add the following code to the `PlayClip` button's `onPress` handler that sets up the event handler itself:

```
_root.tracker.onEnterFrame = function ()
{
  _root.timeleft.text = roundNumber ((_root.orchestra.duration -
    _root.orchestra.position) / 1000, 2) + " seconds";
};
```

This continuously updates the `timeleft` dynamic text field we created earlier. We constantly monitor the position of the sound clip using the `duration` and `position` properties of the `Sound` object, calculating the remaining time in milliseconds. We then convert this to a value in seconds, rounding to two decimal places with the help of our `roundNumber` function.

We also update the other dynamic text fields that hold the length of the clip being played, its name, and its volume.

```
_root.cliplength.text = roundNumber (_root.orchestra.duration /
  1000, 2) + " seconds";
_root.clipname.text = "Choir" + clipnumber;
_root.volume.text = _root.orchestra.getVolume ();
```

Within the `StopClip` button's `onPress` event handler, we include the following code, which clears the dynamic text fields of the information associated with the clip played previously:

```
_root.cliplength.text = "";  
_root.clipname.text = "";  
_root.timeleft.text = "";
```

Finally, the `VolumeUp` and `VolumeDown` `onPress` event handlers are updated. This ensures that any changes to the clip's volume are reflected in the `volume` dynamic text field.

```
_root.volume.text = _root.orchestra.getVolume ();
```

That's it for the code changes.

21. Save your Flash document and preview it.

There! We have a fully-fledged random clip player that provides information on the duration of the current clip, its name, and the play time remaining.

This example can easily be extended through the addition of more clips to the library. Export them for ActionScript reference, and alter the code that selects the clip to play within the `PlayClip` button's `onPress` event handler.



### Use an array!

A useful variation would be to store the linkage names of available clips in an array, so that the number of clips available would not have to be hard-coded.

## Random Track Sequencer Effect

This effect is different from others in the book in that it has no interface! There's nothing to see, but plenty to hear. We're aiming to overlay random tracks to create a unique sound—something that's a little different from what a user might normally hear. The composition contains four ten-second loops comprising a random selection from the following categories:

- Background: a selection of five background bed tracks
- Drums: a selection of five drum tracks

- Guitar: a selection of five guitar loops
- Overlay: a selection of different overlays

As the movie loads, a random selection is made from each category. The tracks are then combined to create one of 625 possible soundtracks.

That's enough of the introduction—let's get cracking! To edit this effect, locate `random.fla` in the code archive.

## Setting the Scene

1. Create a new Flash document that's 200 pixels wide and 200 pixels high. Modify the frame rate to 24 fps (Modify > Document...).
2. Rename Layer1 as Actions.
3. Select File > Import > Import to Library... and select the following MP3 files from the code archive: `Background1.mp3`, `Background2.mp3`, `Background3.mp3`, `Background4.mp3`, `Background5.mp3`, `Drum1.mp3`, `Drum2.mp3`, `Drum3.mp3`, `Drum4.mp3`, `Drum5.mp3`, `Guitar1.mp3`, `Guitar2.mp3`, `Guitar3.mp3`, `Guitar4.mp3`, `Guitar5.mp3`, `Overlay1.mp3`, `Overlay2.mp3`, `Overlay3.mp3`, `Overlay4.mp3`, and `Overlay5.mp3`.
4. Select these sound clips within the Library Panel, then right-click and select Linkage.... Check the Export for ActionScript checkbox, accept the provided identifier (remove the `.mp3` extension if you're using Flash MX 2004), and click OK.

That's all the setting up we need to do for this example. As we're not adding an interface or allowing any user interaction, no other elements need to be included, apart from the ActionScript to produce the sound composition.

## Adding the ActionScript

5. Select the first frame of the Actions layer and add the following code to the Actions Panel:

```
File: random.fla Actions : 1
function randomBetween (a, b)
{
    return Math.min (a, b) + random (Math.abs (a - b) + 1);
}
```

```
}
MovieClip.prototype.PlayQueue = function (clip)
{
    var orchestra = new Sound (this);
    orchestra.attachSound (clip);
    orchestra.start ();
};
function PlayClips ()
{
    var background = "background" + randomBetween (1, 5);
    var drums = "drum" + randomBetween (1, 5);
    var guitar = "guitar" + randomBetween (1, 5);
    var overlay = "overlay" + randomBetween (1, 5);
    var clips = new Array (background, drums, guitar, overlay);
    for (var i = 0; i < clips.length; i++)
    {
        var mc = createEmptyMovieClip ("SoundHolder" + i, i);
        mc.PlayQueue (clips[i]);
    }
}
PlayClips ();
```

That's a fairly concise piece of code. Preview the movie and skip the code walk-through, or read on for the nitty-gritty details—the choice is yours!

Let's look at the `PlayClips` function to see how it all fits together. For each of the tracks, we create a variable (e.g. `drums`) that we fill with one of the linkage identifiers for the sound clips in the library, using the `randomBetween` function to generate a random number between one and five. We then combine these variables into an array of sound clips:

```
var clips = new Array (background, drums, guitar, overlay);
```

From here, we use a `for` loop to create an empty movie clip for each element of the array. These clips play the selected sound clips at random. This is accomplished by calling the `PlayQueue` movie clip method (which we'll examine in a moment) and passing the sound clip linkage identifier as a parameter.

```
for (var i = 0; i < clips.length; i++)
{
    var mc = createEmptyMovieClip ("SoundHolder" + i, i);
    mc.PlayQueue (clips[i]);
}
```

We call the `PlayQueue` method for each of the new movie clips created by the `PlayClips` function. This embeds a sound clip into each of these movie clips, then plays it.

```
MovieClip.prototype.PlayQueue = function (clip)
{
    var orchestra = new Sound (this);
    orchestra.attachSound (clip);
    orchestra.start ();
};
```

Because the sound clips are created in quick succession at runtime, the effect is one of a single composite track with a background, plus drum, guitar, and overlay sections.

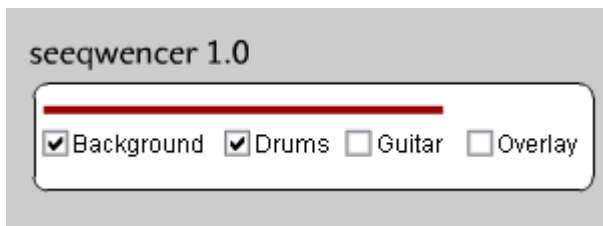
6. Save and preview your document.

You may notice that every time you play the movie, you get a different composition. Some work very well, while others sound fairly clunky! It's all about experimentation and seeing what's possible.

## Random Track Overlay Effect

This effect is based on the previous “headless” example, but builds on it quite extensively, utilizing the `CheckBox` component, custom event handlers, and providing users with the ability to switch tracks on or off. The user interface is shown in Figure 5.2.

**Figure 5.2. This random track overlay effect builds on the previous example.**





## CheckBox incompatibilities

While developing this effect, which uses the **CheckBox** component, I was perplexed to find that the code I had in place for Flash MX didn't work with Flash MX 2004. I investigated this through the Reference Panel and found that there is no `getValue` method for the Flash MX 2004 version of the **CheckBox** component. Additionally, the **CheckBox** no longer allows for a change event handler.

An engineer friend at Macromedia told me that these features hadn't just been deprecated—they had been removed completely!

The code and steps that follow are for Flash MX, while the changes for the 2004 version are provided as asides. The example provided in the code archive was developed using the Flash MX version of the component.

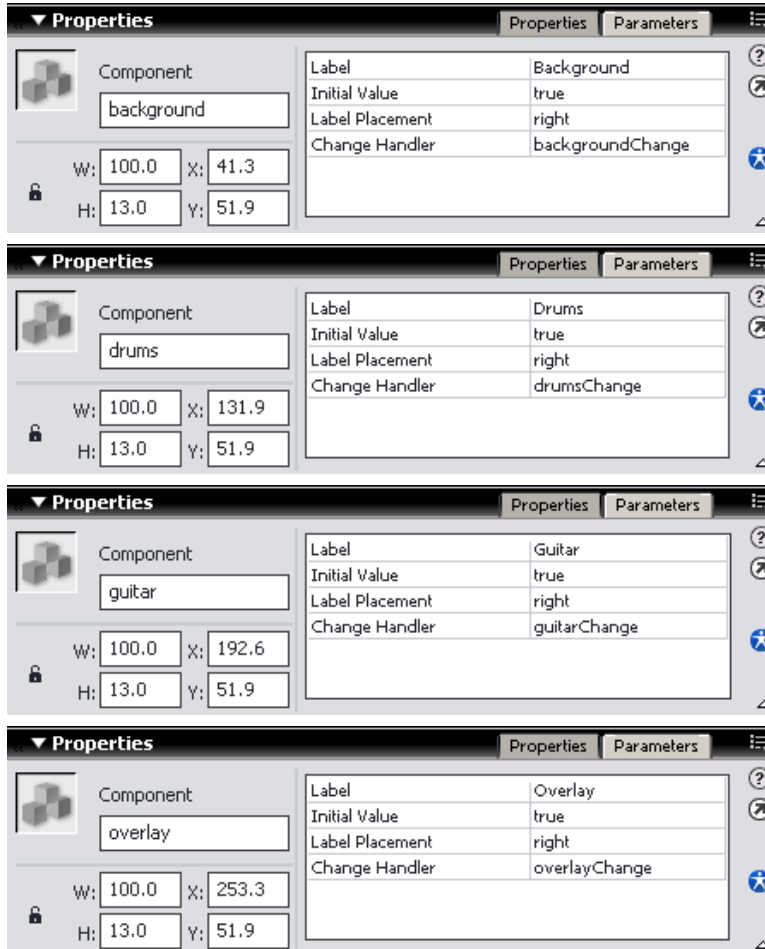
To modify this effect, locate `random-gui.fla` in the code archive.

## Setting the Scene

1. Create a new Flash document that's 550 pixels wide and 100 pixels high. Modify the frame rate to 24 fps (Modify > Document...).
2. Rename Layer1 as **Actions** and create beneath it three layers named CheckBoxes, Elements, and Background.
3. Select File > Import to Library... and select the following MP3 files in the code archive: `Background1.mp3`, `Background2.mp3`, `Background3.mp3`, `Background4.mp3`, `Background5.mp3`, `Drum1.mp3`, `Drum2.mp3`, `Drum3.mp3`, `Drum4.mp3`, `Drum5.mp3`, `Guitar1.mp3`, `Guitar2.mp3`, `Guitar3.mp3`, `Guitar4.mp3`, `Guitar5.mp3`, `Overlay1.mp3`, `Overlay2.mp3`, `Overlay3.mp3`, `Overlay4.mp3`, and `Overlay5.mp3`.
4. Select each of the sound clips within the Library Panel, then right-click and select Linkage... Check the Export for ActionScript checkbox, accept the provided unique identifier (remove the `.mp3` extension if you're using Flash MX 2004), and click OK.
5. From the Components Panel (Window > Development Panels > Components), select Flash UI Components (or just UI Components in Flash MX 2004) and drag four instances of the **CheckBox** component onto the CheckBoxes layer of the stage. Arrange them from left to right using the Align Panel to set their spacing if necessary (Window > Design Panels > Align).

6. Select each CheckBox component on the stage in turn, from left to right, and apply the instance name and parameters shown in Figure 5.3.

**Figure 5.3. The properties for the four CheckBox components are displayed in Flash.**



*note*

### No Change Handler in Flash MX 2004

The Change Handler parameter for each CheckBox is especially important here, as it will be used to mute a track when the corresponding checkbox is unchecked. This parameter doesn't exist in Flash MX 2004,

so we'll add some ActionScript later to handle these events in that version.

7. Select the first frame of the Elements layer and insert into it a 270x4 pixel rectangle. Convert this to a movie clip symbol named **Progress**, and name the instance **progress**. This will act as a quick progress bar, indicating the progress of the composition as it plays.
8. Select the first frame of the Background layer and create a frame around the controls. If you're stuck for inspiration, look at Figure 5.2. Once you're happy with the frame, lock this layer.

Now, let's add the code to make the effect work.

## Adding the ActionScript

9. With the first frame of the Actions layer selected, add the following code to the Actions Panel:

```
File: random-gui.fla Actions : 1 (excerpt)
function randomBetween (a, b)
{
    return Math.min (a, b) + random (Math.abs (a - b) + 1);
}
function roundNumber (toRound, numDecimals)
{
    return Math.round (toRound * Math.pow (10, numDecimals)) /
        Math.pow (10, numDecimals);
}
MovieClip.prototype.PlayQueue = function (clip)
{
    this.orchestra = new Sound (this);
    this.orchestra.attachSound (clip);
    this.orchestra.start ();
};
function PlayClips ()
{
    var background = "background" + randomBetween (1, 5);
    var drums = "drum" + randomBetween (1, 5);
    var guitar = "guitar" + randomBetween (1, 5);
    var overlay = "overlay" + randomBetween (1, 5);
    var clips = new Array (background, drums, guitar, overlay);
    var mc;
```

```
for (var i = 0; i < clips.length; i++)
{
    mc = createEmptyMovieClip ("SoundHolder" + i, i);
    mc.PlayQueue (clips[i]);
}
mc.onEnterFrame = function ()
{
    _root.progress._xscale = this.orchestra.position /
        this.orchestra.duration * 100;
};
mc.orchestra.onSoundComplete = function ()
{
    stopAllSounds ();
    LoopClips ();
};
}
function LoopClips ()
{
    _root.SoundHolder0.orchestra.start ();
    _root.SoundHolder1.orchestra.start ();
    _root.SoundHolder2.orchestra.start ();
    _root.SoundHolder3.orchestra.start ();
}
PlayClips ();
```

We've used the `randomBetween` and `roundNumber` functions previously in this chapter, so there's no need to cover them again.

The `PlayQueue` method is identical to that used in the previous example, as is the `PlayClips` function, except for a couple of event handlers we added to the last dynamically generated movie clip.

```
mc.onEnterFrame = function ()
{
    _root.progress._xscale = this.orchestra.position /
        this.orchestra.duration * 100;
};
mc.orchestra.onSoundComplete = function ()
{
    stopAllSounds ();
    LoopClips ();
};
```

After the sound object is initialized, the sound clip is attached, and the sound clip has started to play, we dynamically track the position of the last sound object

that's created using an `onEnterFrame` event handler attached to the containing movie clip. This event handler controls the width of the progress bar by altering its `_xscale` property. When the sound finishes, the `onSoundComplete` event handler will call the `stopAllSounds` function. Immediately after that, we call `LoopClips` to restart the four tracks we've produced, creating the effect of an infinite loop.

```
function LoopClips ()
{
    _root.SoundHolder0.orchestra.start ();
    _root.SoundHolder1.orchestra.start ();
    _root.SoundHolder2.orchestra.start ();
    _root.SoundHolder3.orchestra.start ();
}
```

So far, so good, right? When the movie is running, we want to dynamically switch the different tracks on or off by clicking the checkboxes on the stage. To this end, each `CheckBox` has its own event handler that controls the volume of the track. Let's add these now.

10. Add the following code below the existing code in the Actions Panel:

```
File: random-gui.fla Actions : 1 (excerpt)
function backgroundChange (background)
{
    if (background.getValue ())
    {
        _root.SoundHolder0.Orchestra.setVolume (100);
    }
    else
    {
        _root.SoundHolder0.Orchestra.setVolume (0);
    }
}
function drumsChange (drums)
{
    if (drums.getValue ())
    {
        _root.SoundHolder1.Orchestra.setVolume (100);
    }
    else
    {
        _root.SoundHolder1.Orchestra.setVolume (0);
    }
}
```

```
function guitarChange (guitar)
{
    if (guitar.getValue ())
    {
        _root.SoundHolder2.Orchestra.setVolume (100);
    }
    else
    {
        _root.SoundHolder2.Orchestra.setVolume (0);
    }
}
function overlayChange (overlay)
{
    if (overlay.getValue ())
    {
        _root.SoundHolder3.Orchestra.setVolume (100);
    }
    else
    {
        _root.SoundHolder3.Orchestra.setVolume (0);
    }
}
```

The names of these functions correspond to the Change Handler parameter values we assigned to each of the `CheckBox` components when we created them. All these event handlers work the same way.<sup>1</sup> They check whether the `CheckBox` is checked using the `getValue` method, then set the volume of the corresponding sound clip to match. The initial state of each `CheckBox` is `true` (checked). So, when the checkbox is clicked, its status switches to `false`, and the volume is set to 0 for that particular `Sound` object. Clicking it again changes the status to `true` and the volume to 100. Using the change handlers in this way, we can drop individual tracks out and bring them back in at will.

11. Save your Flash document and preview it.

You're probably listening to a funky composition right now. Click the checkboxes to drop different tracks out and bring them back in. If you don't like the composition, or it's a little too wild for your tastes, preview the movie again to see what the random sequencer comes up with!

---

<sup>1</sup>If you're feeling ambitious, there is room for a little optimization by combining the common functionality of the four handlers into a single function.

### Changes for Flash MX 2004

If you're using Flash MX 2004, you'll find that the checkboxes don't work as advertised. That's because the **CheckBox** component, besides looking better, lacks a **Change Handler** parameter. As a result, our event handler methods have not been hooked up to the checkboxes. This is a good thing, because these methods use another feature that's missing from the Flash MX 2004 **CheckBox** component: the **getValue** method. Let's now make the changes necessary to get the example working in Flash MX 2004.

First, change each of the event handler functions so they check the **selected** property of the appropriate **CheckBox**, rather than the **getValue** method, which no longer exists. For example, adjust this line of the **backgroundChange** function:

```
if (background.getValue ())
```

Replace it with this line:

```
if (background.selected)
```

Next, set up the event handler functions so they're called when one of the checkboxes is clicked. Add the following code below all the other code in the **Actions Panel**:

```
var cboxListener = new Object ();
cboxListener.click = function()
{
    backgroundChange (background);
    drumsChange (drums);
    guitarChange (guitar);
    overlayChange (overlay);
};

background.addEventListener ("click", cboxListener);
drums.addEventListener ("click", cboxListener);
guitar.addEventListener ("click", cboxListener);
overlay.addEventListener ("click", cboxListener);
```

This creates a new **Object** with a single method called **click**, which is set up as an **event listener** for each of the four **CheckBox** objects on the stage. When any one of the checkboxes is clicked, the **click** method is called. In turn, it calls each of our four event handlers to update the volume accordingly.



### Watch the file size!

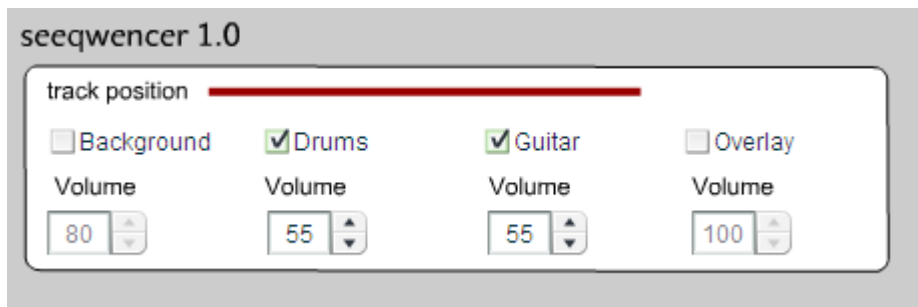
This example contains 20 ten-second clips, weighing in at nearly 3MB in total! This project would be better suited to CD or kiosk deployment than to distribution over the Internet. If the clips were shorter, or if there were

fewer of them, it might be possible to distribute the application over the Internet—just be careful with the file size. Don't say I didn't warn you!

## Modifications

We can make this effect even more interactive by, for example, directly controlling the properties of the `Sound` objects. In the following modification, we'll control the volume of the various sound clips with the `NumericStepper` component, which ships with Flash MX 2004. The interface is shown in Figure 5.4.

**Figure 5.4. Add volume control to the dynamic sound clips.**



As we'll be adding the Flash MX 2004 `NumericStepper` component, we'll take advantage of the new `CheckBox` component that comes with Flash MX 2004, rather than the Flash MX version we used in the previous example.

If you're working with Flash MX, you might as well skip ahead to the next example.

To edit this effect, locate `random-volume.fla` in the code archive.

## Setting the Scene

Building on the previous example, we'll add several pieces of code to accommodate the volume control mechanism, and we'll replace the existing Flash MX `CheckBox` components with their Flash MX 2004 counterparts. We'll also add the new `NumericStepper` component to the interface to provide users with volume control.

12. If you're working from the previous example, locate the Flash MX `CheckBox` components within the `CheckBoxes` layer, and delete them.

13. Select the first frame of the CheckBoxes layer and drag four instances of the `CheckBox` component from the UI Components section of the Components Panel, spacing them evenly across the stage. Name these instances, from left to right: `backgroundCBox`, `drumsCBox`, `guitarCBox`, and `overlayCBox`.
14. For each of the `CheckBox` instances you just created, change the selected parameter within the Property Inspector to `true`, so that the checkboxes will be selected by default.
15. Change the label parameter for each of the `CheckBox` instances. From left to right, they should read: `Background`, `Drums`, `Guitar`, and `Overlay`.
16. Drag four instances of the `NumericStepper` component from the UI Components section of the Components Panel into the first frame of the CheckBoxes layer. Align each `NumericStepper` beneath each of the checkboxes. Name the instances `vol1Control`, `vol2Control`, `vol3Control`, and `vol4Control`, and set their widths to 50 pixels.
17. For each of the `NumericStepper` component instances, set the parameters within the Property Inspector as follows:

<b>maximum</b>	100
<b>minimum</b>	0
<b>stepSize</b>	5
<b>value</b>	100

18. Above each of the `NumericStepper` components, add a static text box containing the text “Volume” to indicate the purpose of the `NumericStepper` component. The outcome should resemble Figure 5.4.
19. Alter the width and height of the movie and the rounded rectangle frame to accommodate the extra controls we’ve added.

Now we can add the extra ActionScript needed to accomplish this effect.

## Adding the ActionScript

20. Select the first frame of the Actions layer. We’ll leave the math and playback functions/methods (`randomBetween`, `roundNumber`, `PlayQueue`, `PlayClips`,

LoopClips) alone, but replace the remaining code (the checkbox event handlers) with the following:

```
File: random-volume.flas Actions : 1 (excerpt)
var backgroundOff = false;
var drumsOff = false;
var guitarOff = false;
var overlayOff = false;
function backgroundChange ()
{
    if (backgroundCBox.selected)
    {
        backgroundOff = false;
        vol1Control.enabled = true;
    }
    else
    {
        backgroundOff = true;
        vol1Control.enabled = false;
        _root.SoundHolder0.orchestra.setVolume (0);
    }
}
function drumsChange ()
{
    if (drumsCBox.selected)
    {
        drumsOff = false;
        vol2Control.enabled = true;
    }
    else
    {
        drumsOff = true;
        vol2Control.enabled = false;
        _root.SoundHolder1.orchestra.setVolume (0);
    }
}
function guitarChange ()
{
    if (guitarCBox.selected)
    {
        guitarOff = false;
        vol3Control.enabled = true;
    }
    else
    {
        guitarOff = true;
    }
}
```

```
        vol3Control.enabled = false;
        _root.SoundHolder2.orchestra.setVolume (0);
    }
}
function overlayChange ()
{
    if (overlayCBox.selected)
    {
        overlayOff = false;
        vol4Control.enabled = true;
    }
    else
    {
        overlayOff = true;
        vol4Control.enabled = false;
        _root.SoundHolder3.orchestra.setVolume (0);
    }
}
function setCustomVolume ()
{
    if (!backgroundOff)
    {
        SoundHolder0.orchestra.setVolume (vol1Control.value);
    }
    if (!drumsOff)
    {
        SoundHolder1.orchestra.setVolume (vol2Control.value);
    }
    if (!guitarOff)
    {
        SoundHolder2.orchestra.setVolume (vol3Control.value);
    }
    if (!overlayOff)
    {
        SoundHolder3.orchestra.setVolume (vol4Control.value);
    }
}
var volumeListener = new Object ();
volumeListener.change = function ()
{
    setCustomVolume ();
};
var cboxListener = new Object ();
cboxListener.click = function ()
{
    backgroundChange ();
```

```
drumsChange ();
guitarChange ();
overlayChange ();
setCustomVolume ();
};
vol1Control.addEventListener ("change", volumeListener);
vol2Control.addEventListener ("change", volumeListener);
vol3Control.addEventListener ("change", volumeListener);
vol4Control.addEventListener ("change", volumeListener);
backgroundCBox.addEventListener ("click", cboxListener);
drumsCBox.addEventListener ("click", cboxListener);
guitarCBox.addEventListener ("click", cboxListener);
overlayCBox.addEventListener ("click", cboxListener);
```

Once again, we have functions to handle the `CheckBox` changes (`backgroundChange`, `drumsChange`, `guitarChange`, and `overlayChange`). They all look fairly similar.

```
function backgroundChange ()
{
    if (backgroundCBox.selected)
    {
        backgroundOff = false;
        vol1Control.enabled = true;
    }
    else
    {
        backgroundOff = true;
        vol1Control.enabled = false;
        _root.SoundHolder0.orchestra.setVolume (0);
    }
}
```

These functions detect the current status of the relevant `CheckBox` by analyzing its `selected` property. If the `CheckBox` is selected, we set a variable to `false` (in this case, the variable is `backgroundOff`) to indicate that the corresponding clip is not switched off. These four variables are set to `false` to begin with, since all four sound clips are switched on at the start of the movie. We also enable the associated volume control `NumericStepper` (`vol1Control`).

If the `CheckBox` is *not* selected, we set the variable to `true` and disable the corresponding `NumericStepper` component, indicating that the sound clip is switched off. We also set the volume of the related sound clip to zero. This switches the

clip off, but allows it to keep playing silently, maintaining its position in case we decide to switch it back on later.

The `setCustomVolume` function sets the volume of each clip to the value specified by its associated `NumericStepper` control, unless the flag variable indicates that it's currently switched off.

```
function setCustomVolume ()
{
  if (!backgroundOff)
  {
    SoundHolder0.orchestra.setVolume (vol1Control.value);
  }
  if (!drumsOff)
  {
    SoundHolder1.orchestra.setVolume (vol2Control.value);
  }
  if (!guitarOff)
  {
    SoundHolder2.orchestra.setVolume (vol3Control.value);
  }
  if (!overlayOff)
  {
    SoundHolder3.orchestra.setVolume (vol4Control.value);
  }
}
```

All that's left is to ensure these functions are called at the appropriate times. A new listener Object called `volumeListener` offers a method called `change` that calls the `setCustomVolume` function.

```
var volumeListener = new Object ();
volumeListener.change = function ()
{
  setCustomVolume ();
};
```

By calling the `addEventListener` method from each of the `NumericStepper` components, we have this object handle changes to the volume of our clips:

```
vol1Control.addEventListener ("change", volumeListener);
vol2Control.addEventListener ("change", volumeListener);
vol3Control.addEventListener ("change", volumeListener);
vol4Control.addEventListener ("change", volumeListener);
```

If any of the `NumericStepper` component values change, the `setCustomVolume` function is called to handle it.

We've also set up an event listener in a similar manner for the `CheckBox` components:

```
cboxListener = new Object ();
cboxListener.click = function ()
{
    backgroundChange ();
    drumsChange ();
    guitarChange ();
    overlayChange ();
    setCustomVolume ();
};
...
backgroundCBox.addEventListener ("click", cboxListener);
drumsCBox.addEventListener ("click", cboxListener);
guitarCBox.addEventListener ("click", cboxListener);
overlayCBox.addEventListener ("click", cboxListener);
```

When any one of the checkboxes is clicked, the event handler calls the four functions we defined earlier (switching on or off the correct sound clip) before calling `setCustomVolume` to correctly set the volume of any clip that has just been switched on.

21. Save your document and export the SWF file to an appropriate location.



## Don't forget ActionScript 2.0

Components distributed with Flash MX 2004 work internally using features of ActionScript 2.0. If you try to publish a movie that contains such components with Publish settings configured for ActionScript 1.0, the components will not work (they'll look like white rectangles with black borders).

New movies created in Flash MX 2004 default to ActionScript 2.0, so you're okay there. But, if you start from the finished version of the previous example provided in the code archive, you'll need to change the ActionScript version setting for the file. Simply click the background of the movie, then click the Settings... button in the Property Inspector. You can adjust the ActionScript version setting on the Flash tab.

Use the checkboxes to switch on and off some of the tracks; notice how they disable and enable the volume controls we've added. Experiment with the volume controls to set a nice balance between the tracks, and see what masterpieces you can come up with!

You can build on this example to modify any number of sound object properties. Spend a little time with it and see what you can achieve.

## XML, MP3, and ID3 Tag Reader

This effect uses the same choir sound clips we used previously in this chapter, but with a slightly different approach. We'll import a playlist from an XML file and load the MP3 files dynamically into Flash. We'll also present song information (including Album, Artist, and Song Title) that's stored within the MP3 in what are called **ID3 tags**, as shown in Figure 5.5.

**Figure 5.5. Create a random MP3 player using XML and ID3 data.**



ID3 tag information is referenced differently in Flash MX than in Flash MX 2004, owing to the differences between versions of Flash Player. Macromedia Flash Player 7 supports the latest ID3 v2.2 and v2.4 tags. As the data is stored at the beginning of the MP3 file, this information is referenced immediately upon

opening the MP3 stream. On the other hand, Macromedia Flash Player 6 supports only ID3 v1.0 and v1.1 tags, which are stored at the end of the file. On that platform, we must wait until the stream is fully loaded before we can retrieve the ID3 tag information.

We'll initially create an MP3 player for version 6 of the Flash Player, supporting ID3 v1.x tags. Then, we'll modify the effect to take advantage of ID3 v2.x tag support within version 7 of the Flash Player.

To edit this effect, locate `mp3-mx fla` in the code archive.

It uses several MP3 files: `choir1.mp3`, `choir2.mp3`, `choir3.mp3`, `choir4.mp3`, and `choir5.mp3`. We'll also be needing the XML file that acts as the playlist; it's called `playlist.xml` and can also be found in the code archive.

## Setting the Scene

For this project, we need a clickable movie clip that loads a random MP3 file and a dynamic text field that displays the ID3 data from the MP3 file.

1. Create a new Flash document that's 500 pixels wide and 450 pixels high. Accept the default frame rate and click OK.
2. Rename the default layer Actions and add beneath it two new layers named Interface and Background.
3. Select the first frame of the Interface layer and create a movie clip called `PlayRandomButton` containing a button that's approximately 70 pixels high and 70 pixels wide. Name the instance `randomPlayer` and move the clip to the left of the stage centre, as shown in Figure 5.5.

We'll use this clip to randomly load an MP3 file and pass the ID3 tag information from the MP3 file into a dynamic text field.

4. Create a new, 320x160 pixel dynamic text field to the right of the button. Name the field `trackID3Info`.
5. In the Property Inspector for the new field, click the Character... button and choose Basic Latin (or All Characters in Flash MX).

This dynamic text field will act as a container that receives the ID3 tag information from the dynamically loaded MP3 files.

Now, for a little window-dressing:


6. Add to the first frame of the Background layer a rectangle that frames the text area you just created.

You can add as many extras as you wish to increase the aesthetics of the interface. Refer to Figure 5.5 for inspiration!

7. Save your work.
8. Locate the files `choir1.mp3`, `choir2.mp3`, `choir3.mp3`, `choir4.mp3` and `choir5.mp3` from the code archive and move them into the same folder in which you saved the document.

## Creating the XML Playlist

We now need to create the XML playlist that references the MP3 files we're planning to load.

 For more information on working with XML, see *Introduction to XML*, by Harry Fuecks[2].

9. Type the following code into a text editor and save it as `playlist.xml` in the same folder.

```
File: playlist.xml Actions : 1
<?xml version="1.0" encoding="iso-8859-1"?>
<playlist>
  <mp3file track="choir1.mp3" />
  <mp3file track="choir2.mp3" />
  <mp3file track="choir3.mp3" />
  <mp3file track="choir4.mp3" />
  <mp3file track="choir5.mp3" />
</playlist>
```

Unlike HTML, which uses a predefined set of tags, XML allows you to create custom data structures. As long as all opening tags have a matching closing tag (or, as in the case of the `mp3file` tag in this example, they close themselves),

---

[2] <http://www.sitepoint.com/article/930>

Flash will import the data structure as an ActionScript XML object, allowing you to manipulate and access the information contained within the file.

This XML file is structurally simple. The `playlist` tag is the top-level node, and the `mp3file` tags are its child nodes. These child nodes also contain an attribute called `track`, which holds the name of the MP3 file we wish to pull into Flash.

Our ActionScript will import this XML file and move through the data structure, populating an array with the names of the MP3 files it contains.

## Adding the ActionScript

10. Select the first frame of the Actions layer and add the following code to the Actions Panel:

```
File: mp3-mx.fla Actions : 1 (excerpt)
var tracklist = new Array ();
var mp3List = new XML ();
mp3List.ignoreWhite = true;
mp3List.onLoad = createPlayList;
mp3List.load ("playlist.xml");
function createPlayList (success)
{
    if (!success)
    {
        return;
    }
    var topLevel = null;
    for (i = 0; i <= this.childNodes.length; i++)
    {
        if (this.childNodes[i].nodeValue == null &&
            this.childNodes[i].nodeName == "playlist")
        {
            topLevel = this.childNodes[i];
            break;
        }
    }
    if (topLevel != null)
    {
        for (i = 0; i <= topLevel.childNodes.length; i++)
        {
            if (topLevel.childNodes[i].nodeName == "mp3file")
            {
                var track =
```

```
        topLevel.childNodes[i].attributes["track"];
        _root.tracklist.push (track);
    }
}
}
function randomBetween (a, b)
{
    return Math.min (a, b) + random (Math.abs (a - b) + 1);
}
function playTrack ()
{
    var track = _root.track;
    if (track.getBytesLoaded () == track.getBytesTotal () &&
        track.duration > 0)
    {
        clearInterval (_root.checkLoaded);
        trackID3Info.text = "";
        trackID3Info.text += "Title: " + track.id3.songname +
            newline;
        trackID3Info.text += "Artist: " + track.id3.artist +
            newline;
        trackID3Info.text += "Album: " + track.id3.album +
            newline;
        trackID3Info.text += "Year: " + track.id3.year + newline;
        trackID3Info.text += "Comments: " + track.id3.comment +
            newline;
    }
}
randomplayer.onPress = function ()
{
    stopAllSounds ();
    var trackNo = randomBetween (0, _root.tracklist.length - 1);
    _root.track = new Sound ();
    _root.track.loadSound (_root.tracklist[trackNo], true);
    _root.checkLoaded = setInterval (playTrack, 500);
};
```

Let's break this code into manageable chunks and see what's going on.

First, we create an empty array to contain our list of MP3 tracks.

```
var tracklist = new Array ();
```

Next, we create an XML object called `mp3List`, and set its `ignoreWhite` property to `true` so that line feeds and whitespace aren't parsed as separate nodes.

```
var mp3List = new XML ();  
mp3List.ignoreWhite = true;
```

We then set the event handler that will execute when the XML file has loaded (`createPlayList`) and load the data from `playlist.xml`.

```
mp3List.onLoad = createPlayList;  
mp3List.load ("playlist.xml");
```

Now we define the `createPlayList` function, which gets called to handle the XML data once it has loaded.

```
function createPlayList (success)  
{  
    if (!success)  
    {  
        return;  
    }  
    var topLevel = null;  
    for (i = 0; i <= this.childNodes.length; i++)  
    {  
        if (this.childNodes[i].nodeValue == null &&  
            this.childNodes[i].nodeName == "playlist")  
        {  
            topLevel = this.childNodes[i];  
            break;  
        }  
    }  
}
```

The first thing we do is check whether the XML file loaded successfully, as indicated by the `success` parameter. If it didn't, we bail out before attempting to process it.

We then enter a `for` loop based on the number of top-level nodes within the XML object (`this`). We check these nodes until we find the one that has a `null` `nodeValue`, meaning it's a tag (as opposed to a piece of text) and that its name is `playlist`. This node is stored in a variable named `topLevel`. We then break out of the `for` loop.

Next, we enter another `for` loop, in order to examine its children:

```
if (topLevel != null)  
{
```

```
for (i = 0; i <= topLevel.childNodes.length; i++)
{
    if (topLevel.childNodes[i].nodeName == "mp3file")
    {
        var track = topLevel.childNodes[i].attributes["track"];
    }
}
```

For each of the child nodes, we check to see if it's an `mp3file` tag, and then populate a variable with the `track` attribute of the tag, which is the name of an MP3 file.

Finally, we populate the `tracklist` array using its `push` method to add the filename to the start of the array.

```
_root.tracklist.push (track);
```

That takes care of building the playlist. Now, when the `randomplayer` movie clip is clicked, the following code is brought into play:

```
randomplayer.onPress = function ()
{
    stopAllSounds ();
    var trackNo = randomBetween (0, _root.tracklist.length - 1);
    _root.track = new Sound ();
    _root.track.loadSound (_root.tracklist[trackNo], true);
    _root.checkLoaded = setInterval (playTrack, 500);
};
```

First, we stop all currently running sounds using the `stopAllSounds` function. We then populate a variable called `trackNo` with a random number in order to choose one of the tracks in our `tracklist` array.

We then create a new `Sound` object and load it up with a random MP3 file using the `trackNo` variable we have just populated.

Finally, we use the `setInterval` function to call the `playTrack` function every half-second. We store the interval identifier in a variable called `checkLoaded` in the root of the movie so we can cancel it when appropriate.

Now, all that's left to discuss is the `playTrack` function:

```
function playTrack ()
{
    var track = _root.track;
    if (track.getBytesLoaded () == track.getBytesTotal () &&
        track.duration > 0)
```

```
{
  clearInterval (_root.checkLoaded);
  trackID3Info.text = "";
  trackID3Info.text += "Title: " + track.id3.songname + newline;
  trackID3Info.text += "Artist: " + track.id3.artist + newline;
  trackID3Info.text += "Album: " + track.id3.album + newline;
  trackID3Info.text += "Year: " + track.id3.year + newline;
  trackID3Info.text += "Comments: " + track.id3.comment +
    newline;
}
```

Each time this function is called, we check whether the MP3 file has loaded successfully. Once it has, we cancel the interval that is set to call the function every half-second with the `clearInterval` function. We can then populate the `trackID3Info` dynamic text field with the ID3 tag information from the MP3 file.



### **ID3 v1.x Caveat**

As Flash Player 6 and Flash MX support only ID3 v1.0 and v1.1 tags, we have to make sure that the MP3 file is completely loaded before we attempt to pull the ID3 tag information from it. This is essential, as the ID3 tag data is found at the end of the MP3 file.

In Flash MX 2004, which can produce content for Flash Player 7 and supports the ID3 v2.2 and v2.4 tag information located at the beginning of the MP3 file, this information is much more easily accessed. We'll see what this means to Flash developers when we modify this example.

11. Save your document and preview your work.

When the movie loads, the XML file is imported and the `tracklist` array is populated with all the track information from the `playlist.xml` file. Clicking on the `randomplayer` movie clip loads an MP3 file and displays its ID3 tag data in the dynamic text field.

## **Modifications**

As we discussed before, it's much easier to reference ID3 tag data in Flash Player 7. This Player supports v2.2 and v2.4 ID3 tags, which are stored at the beginning of the MP3 file rather than at the end.

*note*

### Flash MX 2004 only!

As you can only produce Flash Player 7 SWF files using Flash MX 2004, this modification is suitable for Flash MX 2004 only.

To edit this effect, locate `mp3-mx2004.fla` in the code archive.

We need only make a couple of simple modifications to the movie to allow Flash MX 2004 to read ID3 v2.2 and 2.4 tag information.

12. Select the first frame of the Actions layer, then locate and delete the `playTrack` function from the Actions Panel. It's no longer needed!
13. Replace the `onPress` event handler code for the `randomPlayer` movie clip with the following code:

```
File: mp3-mx2004.fla Actions: 1 (excerpt)
randomPlayer.onPress = function ()
{
    stopAllSounds ();
    var trackNo = randomBetween (0, trackInfo.length - 1);
    var track = new Sound ();
    track.onID3 = function ()
    {
        trackID3Info.text = "";
        trackID3Info.text += "Title: " + track.id3.TIT2 + newline;
        trackID3Info.text += "Artist: " + track.id3.TPE1 + newline;
        trackID3Info.text += "Album: " + track.id3.TALB + newline;
        trackID3Info.text += "Year: " + track.id3.TYER + newline;
        trackID3Info.text += "Comments: " + track.id3.COMM +
            newline;
    };
    track.loadSound (tracklist[trackNo], true);
};
```

This code is essentially the same as before, but it uses an event handler to process the ID3 tag information rather than calling a function every half-second to check for it.

After we've created the new `Sound` object (`track`), we reference a new event handler, called `onID3`, which is available in Flash Player 7. This is invoked when new ID3 data is detected for a loaded MP3 file, which means that we don't need to confirm that the MP3 file has completely loaded before we check for the ID3

tag data. Once the `onID3` event handler is invoked, we can proceed to populate the `trackID3Info` dynamic text field with the ID3 2.0 tags. It's as simple as that!



### **ID3 Reference**

For a complete listing of supported ID3 tags within Flash MX 2004, search for `Sound.ID3` in the Help Panel.

14. Save and preview your work.

Now, when you click the `randomPlayer` movie clip, the MP3 file starts to play and the ID3 tag data is displayed in the dynamic text field almost instantly. You no longer need to wait for the MP3 file to load completely as you did in the previous example.

## **Conclusion**

Who said that all you can do with sound clips is play them? Certainly not me! I hope that we've covered enough material within this chapter to fuel your desire to create some really interesting sound-based interfaces, experimental effects, and other items.

It's now time to see what we can accomplish using video effects in Flash!



---

# Index

## A

### Actions Panel

- adding script using, 12
- Export Script option, 19
- math functions, 191
- optimization, 14

### ActionScript, 3, 8–20

- (*see also* ActionScript examples)
- accessing property values, 10
- advantages over tweening for animation, 87, 92
- application areas, 9
- coding tips, 18
- conditional statements, 56
- core functions, 12
- dot notation, 10
- error types and causes, 393
- externalizing, 19
- interactivity through, 7
- keywords, 10
- script pinning, 19
- setting color in, 83
- single-frame movies, 90
- triggering events, 8
- trigonometric functions, 162, 190
- version settings, 235
- XML objects, 239

### ActionScript 1.0 Publish settings, 235

### ActionScript 2.0

- Flash MX 2004 new feature, 3
- full reference, Flash MX 2004 Help Panel, 4

### ActionScript examples

- barrel.fl, 196–197
- barrel-noise.fl, 200
- blog.fl, 370, 373
- blog-prefs.fl, 379–380
- bounce.fl, 164–165

bounce2.fl, 168

breadcrumb.fl, 356, 360

breadcrumb-caption.fl, 363–365

bulb.fl, 141–142

charting.fl, 424

chromesweep.fl, 130, 133

chromesweep-duplicate.fl, 133

css.fl, 419

Duplication.fl, 108, 110

Duplication\_Modification.fl, 112

Duplication\_Modification\_Clouds.fl, 114

Duplication\_Modification\_Clouds\_And\_Flash.fl, 116

fall.fl, 185

fall-array.fl, 188

fall-stop.fl, 192

fall-sway.fl, 189

Flaming\_Server.fl, 122

flv.fl, 258, 260

flv-control.fl, 265–266

flv-progress.fl, 262–263

flv-random.fl, 269–270

gadget.fl, 57, 59

gadget-alpha.fl, 63

horizontal.fl, 39

intricate.fl, 328, 334, 341

jiggle.fl, 144–145

jiggle-control.fl, 146–147

jiggle-scale.fl, 147

jiggle-scale-rotate.fl, 148

jump.fl, 323, 326

login.fl, 298, 304

login-handle.fl, 312, 315

login-styles.fl, 308–309

mask.fl, 178–179

mask-horizontal.fl, 182

menu.fl, 363, 365–366

---

- miniplayer.fl, 211–212, 214
- miniplayer-feedback.fl, 215
- mp3–mx.fl, 237, 239
- mp3–mx2004.fl, 244
- neon.fl, 135, 138
- opacity.fl, 175
- opacity-multiple.fl, 177
- pancontrol.fl, 209–210
- questions.fl, 342, 347
- random.fl, 219
- Random\_Motion.fl, 95, 98
- Random\_Motion\_Alpha.fl, 100
- randomfade.fl, 169–170
- randomfade-kern.fl, 172
- random-gui.fl, 222, 224, 226, 231
- random-volume.fl, 229
- rotate.fl, 160–161
- rotate-random.fl, 163
- scrubber.fl, 280
- silky.fl, 43, 46
- Simple\_Animation\_Masking.fl, 102, 105, 107
- Simple\_Motion.fl and Simple\_Motion.as, 93
- stickynotes.fl, 386, 388
- subliminal.fl, 66, 68
- subliminal-smooth.fl, 71
- SubtleVideo.fl, 290
- tabs.fl, 72, 80
- tabs-color.fl, 83
- trace.fl, 397–398
- Transition\_Opacity.fl, 174
- try-catch.fl, 399, 406
- User\_Drive\_Motion.fl, 118–119
- validation.fl, 408
- vertical.fl, 50–51
- video-scrubber.fl, 280, 285
- videowall.fl, 273, 275
- videowall-random.fl, 277–278
- volcontrol.fl, 206–208
- zoom.fl, 150
- zoom-out.fl, 154–155
- zoom-out-race.fl, 157
- zoom-step.fl, 152
- zoom-step2.fl, 153
- zoom-timed.fl, 153
- addEventListener method, random track overlay modification, 234
- addItem method, F3DPieChart component, 425
- ADE (application development environment), Flash MX 2004 Professional, 293
- adjustFor function, random fading text effect, 173
- Adobe Systems Inc. (*see* After Effects; Illustrator)
- Advanced settings option, Video Import Wizard, 255
- After Effects (Adobe), 251, 255
- Alert components
  - form-based navigation example, 337
  - introduced, 302
  - mimicking in Flash MX 2004, 303
  - registration form example, 332
- \_alpha values
  - (*see also* opacity)
  - accurate positioning and, 292
  - negative values, 171
  - random modification, 100, 112
- alphanumeric data validation, 409
- Amazon.com navigation example, 31
- angles, conversion of degrees to radians, 162, 198
- animated text (*see* text effects)
- animation, 87–125
  - ActionScript advantages for, 87
  - flame effect, 122
  - function libraries, 92
  - guidelines for effective, 89
  - increasing the redraw rate, 101
  - linear motion example, 90
  - masking, 102

- 
- poetry, 153
  - previewing from a distance, 90
  - raindrops example, 108
  - random motion, 95
  - simulating full video, 289
  - smoothness, 42
  - sound effect linked to, 206
  - sound panning to accompany, 209
  - timeline animation example, 90–91
  - tweening and scripting approaches, 9
  - user-driven, 118
  - variable speed, 112
  - animation overload, 89
  - arrays
    - storing error instances, simple login form, 306
    - storing linkage IDs, video wall effect, 279
    - storing message text
      - falling text effect modification, 189
      - questionnaire form example, 349
      - simple opacity transition, 176
      - sticky note application, 386
    - storing MP3 tracks, 240
    - storing sound clips
      - mini sound player modification, 218
      - random track sequencer effect, 220
  - arrays of arrays, 350
  - attachMovie method, 98
    - falling text effect, 186
    - vertical sliding transition effect, 181
    - video wall effect, 276
  - attachSound method, mini sound player example, 213
  - audio processing software, 204
  - Audio track option, 256
  - authentication (*see* validation)
  - Auto Format option
    - Actions Panel button, 15
    - configuring, 15
- ## B
- backgrounds
    - adding background noise, 200
    - creating global backgrounds, 302
    - creating shaded backgrounds, 45
    - dazzling chrome effect, importing, 131
    - different colored areas, 324
    - masks, video wall effect, 274
    - scripted masking example, 103
    - silky fading opacity effect, 43
  - backups using email, 93
  - bandwidth
    - animation simulating full video, 289
    - importing video clips, 259
    - vector graphics and, 6
    - Video Import Wizard, 253
  - bar graphs, 106
  - behaviors, Flash MX 2004, 5
  - bitmap graphics
    - glow effects, 136, 160
    - importing, jiggling text effect, 144
  - blog readers (*see* News Sweeper application)
  - bouncing text effect, 164
    - modifications, 167
  - brackets and unbalanced identifier errors, 394
  - breadcrumb navigation, 356
  - breadcrumb.php, 356, 358
  - breadcrumb-caption.php, 367
  - Break Apart option, jiggling text effect, 144
  - breakpoints
    - adding to code being debugged, 409
    - Debugger Panel and, 405

- Button components
  - inter-movie communication example, 364
  - simple login form example, 302
- buttons
  - assigning event handlers, 41
  - creating, simple navigation example, 36
  - customizing labels, 305
  - down state effects, 44
  - feedback to users pressing, 28
  - Flash MX 2004 Common Libraries, 265
  - mini sound player example, 212, 214
  - rollover effects, 37
  - scripting the opacity of, 63
  - silky fading opacity effect, 44
  - sound effect for clicking, 205
  - subliminal navigation example, 67
  - trigger buttons
    - tabbed navigation example, 75, 81
    - user-driven motion example, 119–120
- C**
- case sensitivity, ActionScript 2.0, 18, 394
- Change Handler parameter
  - absence from Flash MX 2004 CheckBoxes, 223
  - absent from Flash MX 2004 CheckBoxes, 228
  - functions corresponding to, 227
- characters, rotating (*see* circulating text effect)
- characters, separating words into
  - jiggling text effect, 144
  - random fading text effect, 169
- charAt method, String object, 171
- Charting in Flash, 422–426
- chasing clips effect, 157
- Check Syntax button, Actions Panel, 15
- CheckBox components
  - Flash MX and Flash MX 2004 incompatibilities, 222
  - Change Handler parameter, 223, 228
  - random track overlay effect, 221–222, 229
  - registration form example, 330
- checkLength function
  - data validation debugging example, 410–411
  - checkLength function, data validation debugging example, 410–411
- checkLogin function
  - login form example, 317, 320
  - checkLogin function, login form example, 317, 320
- choir\*.mp3 files, mini sound player, 212
- chrome effect, 130
- circular menu example, 56–63
  - modifications, 63
- circular navigation systems, 56
- circulating text effect, 159
  - modifications, 163
  - reverse spin, 162
- client-side validation, 297
  - simple login form example, 305
- cloud effects, raindrops animation example, 113
- code hinting, 14
- code readability
  - Auto Format option, 15
  - commenting, 16
  - dynamic volume control example, 209

---

code samples (*see* ActionScript examples)

coefficient of restitution, 166

color

- coding values in ActionScript and CSS, 359
- Flash color codes, 360
- setting for a movie clip, 83

Colorizer movie clip, tabbed navigation example, 84

ComboBox components

- form-based navigation example, 323, 325
- News Sweeper application, 370
- random movie loader, 268
- registration form example, 330

comma-separated lists

- populating an array, 349
- populating text fields, 390

commenting

- code readability and, 16
- multiline comments, 17

Common Libraries Panel, 265

Component Definition dialog, 77

- adding parameters, 83

Component Inspector

- displaying, 300
- simple login form example, 299

components, 77

- adding more instances, 82
- deleting, for subsequent calling, 303
- Flash MX 2004 Professional, 5
- video components, 248
- parameter locking for reuse, 79
- parameters explained, 79
- slider component availability, 280
- use as classes, 305

Components Panel (*see also individual components*)

- Alert component, 302
- Button component, 302
- CheckBox component, 222, 230
- ComboBox component, 269, 325
- DateChooser component, 397
- F3DPieChart component, 423
- Label component, 301
- NumericStepper component, 230
- ProgressBar component, 263
- RadioButton component, 347
- TextArea component, 330
- TextInput component, 299
- UI Components section
  - components tabulated, 294
- Window component, 332

compression profile settings, Video Import Wizard, 253

conditional statements, ActionScript, 56

content, and search engine listing, 427

contentPath property

- logging out, 320
- login form example, 318

context-based help, 4

control buttons, external FLV file access example, 265

ControlPlayback method, video wall effect, 279

cookies and Local Shared Objects, 340

copying (*see* duplication)

cosine functions (*see* Math.cos function)

CPU loading (*see* processor load)

Create Clip button, Video Import Wizard, 252

Create Motion Tween command, timeline animation, 91

createEmptyMovieClip method

- jiggling text effect, 145
- video wall effect, 275

createPlayList function, MP3 player example, 241

cropping video clips, 256

- CSS (Cascading Style Sheets)
  - breadcrumb.php and, 359
  - editors, 417
  - Flash support for, 415–422
    - compound selectors excluded, 418
    - supported properties, 421
- Current Timeline track option, 256
- cursor position (*see* mouse pointer position)
- cycling through text, 173
- D**
- Data Binding
  - Flash MX 2004 Professional, 6
- data sources, external (*see* external data sources)
- data types
  - date formats, 397
  - type mismatch errors, 394
- database linking, News Sweeper application, 369
- date formats, 397
- DateChooser components, trace function example, 397
- dazzling chrome effect, 130–133
  - modification, 133
- Debug Movie option, 409
- Debugger Panel, 401
  - complex code example, 407
  - control options, 403
  - setting breakpoints, 407
  - tabs, 402
  - using breakpoints, 405
  - using watches, 412
- debugging, 393–413
  - Error object use, 399
  - Flash MX 2004 error reporting, 4
  - trace function, 396
  - using the Debugger Panel, 401
- debugging tools, 393
- degrees, conversion to radians, 162, 198
- depth, illusion of
  - falling text effect modification, 191
  - raindrops animation example, 113
  - using opacity, 199
- description meta tag, 426–427
- dimensions
  - (*see also* scaling objects)
  - random modification, 112
- distance, suggesting (*see* depth)
- dot notation, ActionScript, 10
  - circular menu example, 62
- drawing tools, raindrops animation, 109
- drop-down lists, populating, 375
- DropIn method
  - bouncing text effect, 167–168
- DropIn method, bouncing text effect, 167–168
- dummy text for content areas, 325
- duplicateMovieClip method, raindrops animation, 111
- duplicating objects
  - animation by, 108
  - components as an alternative, 77
  - text boxes, 344
  - using the Library Panel, 36
- dynamic data (*see* external data sources)
- dynamic panning control example, 209
- dynamic responsiveness, ActionScripted animation, 92
- dynamic sticky notes (*see* sticky note application)
- dynamic text fields
  - breadcrumb navigation example, 357
  - circular menu example, 59
  - CSS styling example, 417
  - inter-movie communication, 363
  - mini sound player modification, 217
  - MP3 player example, 237
  - registration form example, 341

---

sticky note application, 388  
dynamic text, falling text effect, 187  
dynamic volume control example, 206

## E

elastic effects, subliminal navigation  
example, 67  
electricity, sound suggesting, 134  
email archiving of backups, 93  
Embed Font Outlines For option, 417  
embedded video, 249  
example, 258  
empty movie clips  
jiggling text effect, 145  
video wall effect, 275  
zooming text effect, 151  
Encoding dialog, Video Import Wizard,  
253  
crop option, 256  
error messages, simple login form ex-  
ample, 306  
Error object, 399  
error reporting, Flash MX 2004, 4  
errors in ActionScript  
handling with try-catch blocks, 399  
types and causes, 393  
event handlers  
corresponding to Change Handler  
parameter, 227  
onEnterFrame  
changing button opacity, 64  
circular menu example, 61  
fading effects, 47  
setInterval function and, 99  
stopped movie clips, 287  
video loading progress bar, 263  
onID3, 244  
onLoad  
checking for errors, 401  
sticky note application, 389  
video scrubber device, 286  
onPress  
inter-movie communication, 367  
mini sound player example, 213  
questionnaire form example, 351  
simple login form example, 305  
sticky note application, 391  
onRelease, 391  
onRollOver and onRollOut  
glowing bulb effect, 143  
silly fading opacity effect, 47  
simple navigation example, 41  
vertical navigation example, 53  
onSetFocus and onKillFocus, 310  
event listeners  
pie chart example, 425  
random track overlay effect, 228,  
235  
event triggers, ActionScript, 8  
exploding menus, 33  
exponential function, ActionScript, 191  
Export for ActionScript option, 95–96  
dynamic volume control example,  
206  
falling text effect, 185  
Export Script option, Actions Panel, 19  
exporting video clips, Flash MX 2004  
Professional, 248  
external data sources, 355–392  
checking for loading errors, 400  
Flash MX 2004 Professional, 6  
inter-movie communication, 362  
simple blog reader application, 369  
sticky note application, 385  
storing preferences with a Local  
Shared Object, 378  
external FLV file access, 257  
adding control buttons, 265  
modifications, 261  
random movie loader, 268

**F**

- F3DPieChart component, 422–423
- Fade In command, 24
- FadeIn method, random fading text effect, 171
- fadeMe method, breadcrumb navigation example, 361
- fading
  - (*see also* alpha values; opacity effects)
  - bouncing text effect, 166
  - glowing bulb effect, 141
  - onEnterFrame event handler, 47
  - silky fading opacity effect, 43
  - Timeline Effects and, 24
- falling text effect, 184
  - modifications, 187, 191
- feedback
  - auditory, 203, 205
  - importance of, in navigation systems, 28, 49
  - visual
    - adding a focus glow, 311
    - through a GUI, 211
    - through dynamic text fields, 215
    - with a progress bar, 261
- file sizes
  - embedded video, 249
  - importing video clips, 259
  - random track overlay effect, 228
  - vector graphics and, 7
- Final Cut Pro, 251
- Fireworks MX 2004, 30, 76
  - (*see also* phireworx.com)
  - bouncing text graphics, 165
  - circulating text graphics, 160
  - custom icons, 303
  - dazzling chrome effect use, 131
  - glowing bulb effect graphics, 141
  - neon text effect graphics, 135
  - simple login form use, 302
- FixDate function, trace function example, 398
- FLA files (*see* ActionScript examples)
- flame effect animation, 122
- Flash code examples (*see* ActionScript examples)
- Flash Debugger (*see* Debugger Panel)
- Flash Exchange, for Timeline Effects, 202
- Flash Form Presentation, 5
- Flash forms (*see* forms)
- Flash JavaScript API, 202
- Flash MX
  - ID3 tag referencing, 236, 243
  - organizing layers into folders, 22
  - version of random track overlay effect, 222
- Flash MX 2004
  - Actions Panel, 12–15
    - components, 13
  - CheckBox incompatibilities with Flash MX, 222
  - CSS support, 415
  - debugging tools, 393
  - ID3 tag referencing incompatibilities with Flash MX, 236, 243
  - ID3 tag support, listing, 245
  - mimicking Alert component functionality, 303
  - new features, 3
    - CheckBox component, 229
    - Error object, 399
    - fast frame label editing, 22
    - improved security model, 4
    - NumericStepper component, 229
    - RadioButton rewritten, 341
    - script navigator pane, 19
    - spell checking, 4
    - Timeline Effects, 23, 201
    - video enhancements, 247
    - Video Import Wizard, 247

- 
- sound editor limitations, 205
  - use for examples in this book, 3
  - version of random track overlay effect, 228
  - Flash MX 2004 Professional
    - forms development methodology, 298
    - new features, 3, 5
      - video enhancements, 247
    - recent release of, 2
    - registration form example, 328
    - Screen Outline Pane, 294
    - simple login form example, 298
    - video exporting, 260
  - Flash Player
    - Debugger Panel and, 401
    - detecting unbalanced identifiers, 395
    - performance enhancement in Flash MX 2004, 4
    - previewing in, instead of Flash, 199
    - version 7
      - MP3 player modifications, 243
    - versions 6 and 7
      - ID3 support, 237
      - MP3 player example, 236, 243
  - Flash Slide Presentation, 5
  - Flash video files (*see* FLV files)
  - flickering
    - neon text effect, 134
    - opacity, 100
  - flowcharting navigation systems, 29
    - phireworks.com example, 31
  - fluid motion, simple navigation example, 42
  - flush method, Local Shared Objects, 340
  - FLV files
    - accessing external files, 257
    - exporting video clips as, 259
    - Flash MX 2004 support, 247
    - video inclusion with, 250
  - focus glows, 308
  - folder and layer structure
    - circular menu example, 57
    - silky fading opacity effect, 44
  - folders
    - author's standard framework, 23
    - organizing layers and
      - flame effect animation, 122
      - scripted masking example, 103
      - tabbed interface example, 73
      - user-driven animation, 119
    - organizing layers into, 22
  - font sizes, CSS, 418
  - form fields
    - (*see also* required form fields)
    - validation, debugging, 407
  - formatting text using CSS, 415
  - forms, 293–353
    - design considerations, 296
    - Flash and HTML compared, 293
    - Flash MX 2004 Professional, 5
    - grouping questions, 297, 324
    - handling form data, 311
    - importance of planning, 296
    - registration form example, 327
    - scripted questionnaire, 341
    - simple login form example, 298
    - snazzy forms, 323
  - form-based navigation, 323–327
  - frame extraction, subtle video example, 290
  - frame rates
    - included video, 250
    - setInterval and independence of, 101
  - frame size, captured video footage, 251
  - frameRate global variable, video scrubber device, 286
  - frames
    - random, advancing to, 111
  - fully-qualified class names, components, 305
  - function calls, ActionScript, 94
  - function libraries, 92–95

## functions

- coding tips, 18
- header comments, 17

**G**

- gadget-based navigation, 33, 55
  - circular menu example, 56–63
- GetData function, News Sweeper application, 375
- getMaxValue method, Scrubber component, 284
- getMinValue method, Scrubber component, 284
- getURL function
  - introduced, 12
  - simple navigation example, 37, 42
- getVal method, Scrubber component, 284
- getValue method, CheckBox class
  - Flash MX and Flash MX 2004 incompatibilities, 222
  - random track overlay effect, 227
- glassy buttons, 67
- gliding effects, onEnterFrame event handlers, 54
- \_global object, 41, 53
- global variables
  - circular menu rotation speed, 61
  - questionnaire form example, 349
  - simple navigation example, 40
  - vertical navigation example, 53
  - video scrubber device, 286
- glow effects
  - circulating text graphics, 160
  - login form focus glow, 308
  - neon text effect, 136
- glowing bulb effect, 140
  - modifications, 143
- gotoAndPlay function, 12
  - flame effect animation, 124
  - raindrops animation example, 111

- gotoAndStop function, 12
  - video scrubber device, 288
- GrabData function, News Sweeper application, 376
- gradient fill
  - dazzling chrome effect, 131
  - scripted masking example, 103
  - vertical sliding transition effect, 179
- graphics
  - (*see also* bitmaps; icons)
  - charting in Flash, 422
  - confirming logged in status, 313
  - vector and bitmap compared, 6
- gravity, simulating, 167
- GUI (graphical user interface) (*see* user interfaces)
- guides, switching on, 37

**H**

- Help Panel, Flash MX 2004 improvement, 4
- HelpPopup button, registration form example, 330
- hideControls function, questionnaire form example, 352
- hierarchies, navigational, 28
- highlight animation, 130
- highlightMarker function, questionnaire form example, 350
- hitTest method
  - circular menu example, 62
  - falling text effect modifications, 195
- horizontal navigation systems, 34–49
  - gadget-based example, 55
  - introduced, 31
  - silky fading opacity effect, 43
  - simple horizontal navigation, 34
- horizontal oscillation, 210
- horizontal sliding transition, 182
- hotmail.com backups, 93

---

## HTML

- Flash advantages over, 2
- forms, compared with Flash forms, 293
- indexing by search engines and, 426
- jump menus, 323
- text, CSS formatting, 415
- hyperlinks
  - search engine listing and, 427
  - styling, 419

## I

### icons

- buttons as, 37
- custom icons, 303
- ID3 tags, 236
  - Flash MX version incompatibilities, 236, 243
- ignoreWhite property, XML object, 241
- illumination (*see* lighting effects)
- Illustrator 10 (Adobe), support for importing, 4
- Import menu, track options, 256
- importing component classes, simple login form, 305
- importing files, Flash MX 2004 supported formats, 4
- importing graphics
  - circulating text effect, 160
  - jiggling text effect, 144
  - simple login form example, 303
- importing PNG files, 51
  - video wall effect, 275
- importing sound clips, 205
  - dynamic volume control example, 206
  - mini sound player example, 212
  - random track overlay effect, 222
  - random track sequencer effect, 219
- importing video, 251
  - QuickTime movies, 257

- importing XML playlists, 236, 239
- #include directive
  - externalizing ActionScript, 19
  - importing functions, 93
- infinite loop effect, random track overlay, 226
- Information Area movie clip, simple navigation example, 38
- Insert Keyframe command, timeline animation, 91
- interactive text effects, 128
- interactivity with users
  - advantages of ActionScript, 7
  - falling text effect modifications, 191
  - gadget-based navigation, 55
  - random track overlay effect, 221
- interfaces (*see* user interfaces)
- inter-movie communication, 362
- intricate forms (*see* registration form example)
- isAlphaNumeric function, data validation debugging example, 410–411

## J

### JavaScript

- ActionScript basis in, 7
- ValidEmail function, 340
- Jiggle method, 146
- jiggling text effect, 144
  - modifications, 146
- jittery nervousness, 101
- jump menus (*see* form-based navigation)

## K

- kerning, 172
- keyframe insertion
  - questionnaire form example, 343
  - timeline animation, 91
- keywords meta tag, 426–427
- keywords, ActionScript, 10

**L**

Label components  
  News Sweeper application, 371  
  registration form example, 329  
  simple login form example, 301  
Labels layer, usefulness of, 22  
layers  
  organizing into folders, 22  
  placing script into locked layers, 20  
letterboxing, 256  
letters (*see* characters)  
Library Panel  
  Component Definition dialog, 77  
  duplicating objects, 36, 77  
lift and lock, 20  
lighting effects, 130–143  
  dazzling chrome effect, 130  
  glowing bulb effect, 140  
  neon text effect, 134  
lightning flash effect, 116  
lightweight video, 289  
linear motion, simple animation example, 90  
Linkage option, Library Panel  
  (*see also* Export for ActionScript option)  
  offstage instance alternative, 388  
  viewing parameters, 95  
linkage properties, neon text sound byte, 136  
links (*see* hyperlinks)  
LoadVars class  
  login form example, 317  
  sticky note application, 389  
Local Shared Objects, 340  
  introduced, 378  
  sticky note application modifications, 392  
  uses, 385

LocalConnection class  
  inter-movie communication, 363, 365–366  
  uses, 369  
Locals tab, Debugger Panel, 403, 412  
loggedin.png graphic, 313  
logging out, login form example, 319  
logic errors, 395  
login.php script, 317, 321  
logins (*see* simple login form example)  
logout.php script, 322  
“lorem ipsum” dummy text, 325

**M**

Macromedia Corporation (*see* Fireworks; Flash)  
Macromedia Developer Resource Kit 5, 422  
Macromedia DevNet, 280  
Macromedia Flash Search Engine SDK, 428  
markerbutton class, questionnaire form example, 346, 351  
masking  
  dazzling chrome effect, 132  
  hiding passwords, 301  
  imported videos, 249  
  scripted, 102  
  vertical sliding transition effect, 179  
master object libraries, 94  
Math.abs and randomBetween functions, 124  
Math.cos function  
  3D barrel roll effect, 198  
  circulating text effect, 162  
  dynamic volume control example, 207  
  oscillating motion  
    falling text effect, 190  
Math.exp function, 191  
Math.floor function, 287

- 
- Math.min and randomBetween functions, 124
  - Math.round function, 217
  - Math.sin function
    - 3D barrel roll effect, 198
    - circulating text effect, 162
  - memory saturation, embedded video, 250
  - menus
    - extension of user-driven input example, 121
    - inter-movie communication example, 364
  - messages conveyed by text effects, 128
  - meta tags, 426
    - phireworx.com example, 428
  - methods listing, code hinting, 14
  - mini sound player example, 211
    - modifications, 215
  - modes, Actions Panel, 14
  - Moluv's Picks Website vertical navigation example, 33
  - MONO\*crafts Website
    - (www.yugop.com/ver2/), 32
  - motion effects, 143–168
    - (see also movement; random motion)
    - bouncing text effect, 164
    - circulating text effect, 159
    - jiggling text effect, 144
    - zooming in effect, 149
    - zooming out effect, 154
  - motion tweening (see tweening)
  - mouse pointer position
    - determining distance from an object, 61
    - navigation system responding to, 56, 61
    - scaling text and, 191
    - varying rotational speed with, 199
  - mouseover effects
    - circular menu example, 62
    - CSS example, 419
    - falling text effect modifications, 195
    - glowing bulb effect, 143
    - simple navigation buttons, 37–38
    - video wall effect, 279
  - movement
    - (see also motion effects)
    - creating the illusion of, 37, 290
      - avoiding overshooting, 55
      - fluid motion, 42
    - using Transform/Transition Timeline Effects, 201
  - moveText function, simple animation
    - masking, 107
  - MoveTo function, user-driven motion example, 120
  - Movie clip track option, 256
    - example, 258
  - movie clips
    - (see also empty movie clips)
    - capturing, 251
    - creating, on PC and Mac, 34
    - intercommunicating, 362
    - offstage instances and linkage identifiers, 388
    - positioning by reducing `_alpha` values, 292
    - single-frame movies, 90
  - movie libraries, 251
  - Movieclip class
    - prototype functions, 145
  - Movieclip.prototype functions, 142
  - MP3 player example, 236
    - creating the XML playlist, 238
    - modifications, 243
  - multiple instances
    - increasing numbers, 101
    - random placement, 98
- N**
- naming conventions, ActionScript, 20

- naming object instances, 47
- navigation systems
  - advanced navigation, 72
  - breadcrumb navigation, 356
  - circular navigation system, 56
  - common navigation methods, 31
  - design principles, 27
  - extension of user-driven input example, 121
  - flat navigation example, 30
  - form-based navigation, 323
  - gadget-based navigation, 55–65
    - introduced, 33
  - horizontal navigation, 31, 34–49
  - planning stage, 29
  - scrolling navigation systems, 32
  - subliminal navigation, 65
  - tabbed interface example, 72
  - vertical navigation, 32, 49–55
- neon text effect, 134
- nested movie clips, references to, 62
- NetConnection class, 260
- NetStream class, 260
  - video loading progress bar, 264
- newline command, pie chart example, 426
- News Sweeper application, 369–378
  - Local Shared Object use, 378
- Nielsen, Jakob, *Designing Web Usability*, 27
- note application (*see sticky note application*)
- notice board application, 392
- NumericStepper components, 229–230

## O

- <object> tags, introducing external data, 355, 359
- objects
  - creating master libraries, 94

- making accessible through naming, 47
- offstage instances, movie clips, 388
- on\* methods (*see event handlers*)
- OOP (Object-Oriented Programming)
  - ActionScript and, 8
- opacity
  - (*see also alpha values; fading*)
  - adjusting using the Alpha option, 46
  - of buttons, in circular menu example, 63
  - flickering, 100
  - random, raindrops animation, 115
  - suggesting distance using, 199
- opacity effects, 168–184
  - random fading text effect, 168
  - silky fading opacity effect, 43
  - simple opacity transition, 173
  - sliding text, 178
    - horizontal transition effect, 182
    - vertical transition effect, 178
- OpacityTransition method, 176–177
- oscillating motion
  - sound effects accompanying, 210
  - using trigonometric functions, 190
- Output Panel
  - syntax error description, 394
  - trace function and, 396
- overshooting target positions, 55

## P

- parallax effects, raindrops animation, 113
- Parameters are Locked in Instances
  - option, 79
- Parameters Panel, Property Inspector, 80
- \_parent keyword, ActionScript, 10
- parentheses and unbalanced identifier errors, 394
- passwords, hiding by masking, 301

---

Paste in Place command, 76  
 pause method, NetStream class, 267  
 PDF (Portable Document Format) files, 4  
 pen tool, flame effect animation, 123  
 phireworx.com  
     meta information, 428  
     navigation example, 30  
 PHP and server-side validation, 321  
 pie chart example, 422–426  
 placeholdering by script pinning, 19  
 play function, 12  
 PlayClips function, random track sequencer, 220  
 playFLV function, external file access example, 267  
 playlist.xml file, 237–238  
 PlayQueue method, MovieClip class, 220, 225  
 playRandomClip function, random movie loader, 272  
 playSound function, neon text effect, 139  
 playTrack function, MP3 player, 242  
 poetry animation, 153  
 PopulateChart function, pie chart example, 425  
 populateRadios function, questionnaire form, 352  
     questionnaire form example, 350  
 pop-up windows, registration form example, 333  
 PopupManager class, form-based navigation example, 337  
 position marking (*see* breadcrumb navigation; progress markers)  
 position storage, Local Shared Objects, 392  
 preference storage, Local Shared Objects, 378  
 Preview Clip button, Video Import Wizard, 252  
 previews  
     debug mode, 402  
     detecting over-animation, 90  
     Flash and Flash player, 199  
 processor load  
     animation simulating full video, 289  
     increasing the redraw rate, 101  
     multiple objects and, 101  
     Timeline Effects, 201  
 progress bars  
     random track overlay effect, 224  
     video loading, 261  
 progress marker  
     questionnaire form example, 345, 351  
 ProgressBar components, 261  
 properties  
     accessing values, in ActionScript, 10  
     silky fading opacity effect, 48  
 Properties tab, Debugger Panel, 403  
 Property Inspector  
     ActionScript version settings, 235  
     adding a frame label, 22  
     Dynamic Text option  
         circular menu example, 59  
         random fading text effect, 169  
     Embed Font Outlines For option, 417  
     Embed font outlines options, 106  
     modification of applied Timeline Effects, 201  
     Parameters Panel, 80  
     precise sizing, 35  
     Render Text as HTML button, 417  
 prototype functions  
     circulating text effect, 161  
     glowing bulb effect, 142  
     jiggling text effect, 145  
     subliminal navigation example, 67  
 pseudo-classes, CSS, 419

- pseudocode, 30
- Publish settings
  - ActionScript versions and, 235
  - search engine listings and, 427
- push method, MP3 player example, 242
- pushData method
  - Scrubber component, 283
  - video scrubber device, 287

## Q

- quality settings (*see* bandwidth)
- query strings, breadcrumb.php, 360
- questionnaire form example, 341
- QuickTime FLV Exporter Plug-In
  - Flash MX 2004 Professional new feature, 248
  - processing software and, 251
  - video exporting, 260
- QuickTime movies, importing, 257
- quotes and unbalanced identifier errors, 394

## R

- radians, conversion of degrees to, 162, 198
- RadioButton components
  - questionnaire form example, 346
- RadioButton components, questionnaire form, 341
- raindrops animation example, 108
  - lightning flash effect, 116
- random fading text effect, 168
- random frames, advancing to, 111
- random function, 98
  - initial object locations, 111
  - neon text effect, 139
  - raindrops animation example
    - lightning flash effect, 117
- random motion
  - example illustrating, 95
  - keeping the graphics on stage, 102

- modifications, 100
- testing, 99
- jiggling text effect modification, 147
- random movie loader, 268
- random placement, circulating text effect, 160
- random track overlay effect, 221
  - modifications, 229
- random track sequencer effect, 218
- randomBetween function
  - circulating text effect, 161, 163
  - flame effect animation, 124
  - jiggling text effect, 145, 147
  - mini sound player example, 213
  - random fading text effect, 170
  - random movie loader, 272
  - random track overlay effect, 225
  - random track sequencer effect, 220
  - video wall effect, 276, 279
- Randomizer function
  - altering the shift value, 101
  - modifying, 100
  - Random\_Motion fla, 98
- realistic effects, shadows and reflections, 291
- Rectangle Tool, vertical sliding transition effect, 179
- rectangles, precise sizing, 35
- Red Giant Software Timeline Effects, 5
- reflections and the illusion of movement, 291
- registration form example, 327–341
- registration points and scaling, 192
- Render Text as HTML button, 417
- renderText function
  - CSS example, 420–421
- renderText function, CSS example, 420–421
- required form fields
  - Alert component use, 302
  - distinguishing, 297

---

resizing video clips, scaling alternative, 274  
 reusability  
     advantage of scripted effects, 9  
     captured video footage, 251  
 reverb.mp3 file, 206  
 rich Internet applications using screens, 312  
 rollover effects (*see* mouseover effects)  
 \_root keyword, ActionScript, 10  
 rotating text (*see* circulating text effect)  
 rotation direction, reversing, 162  
 rotation speed  
     varying with mouse position, 199  
     varying, circular menu example, 61  
 roundNumber function  
     mini sound player modification, 217  
     random movie loader, 272  
     random track overlay effect, 225  
 rulers, switching on, 37  
 running tap video simulation, 289  
 runtime errors, 396

**S**

SaveSelection function, News Sweeper application, 384  
 saveSelection function, questionnaire form example, 352  
 scalability of ActionScript, 21  
     scripted animations, 92  
 ScaleDown method  
     zooming text effect, 153–154  
 ScaleDown method, zooming text effect, 153–154  
 scaling objects  
     according to cursor proximity, 191  
     horizontal scaling of jiggling text, 147  
     linking to sound volume, 206  
     Math.cos function use, 207  
     random variation, 100  
     registration points and, 192  
     vertical scaling, 207  
     video clips before importing, 274  
 scaling variable, 3D barrel roll effect, 198  
 ScheduleTrigger function, zooming text effect, 154  
 Screen Outline Pane, Flash MX 2004 Professional, 294, 298  
 ScreenJumper application (*see* form-based navigation)  
 script navigator pane, 19  
 script pinning, 19  
 scripted questionnaire, 341  
 scripting  
     (*see also* ActionScript; server-side scripting)  
     function libraries, 92  
 scrolling navigation systems, 32  
 Scrubber components, 282  
 SDK (Software Development Kit) for search engine accessibility, 428  
 search engines, achieving listing by, 426–428  
     Macromedia SDK, 428  
 secure.php script, login form, 322  
 secure.swf movie, login form, 314  
 security model, Flash MX 2004, 4  
 seek method, NetStream class, 267  
 send method, LocalConnection class, 366  
 sendAndLoad method, LoadVars class, 317  
     alternatives to, 369  
 server-side scripting  
     breadcrumb.php, 357  
     introducing external data with, 355  
     online examples, 362, 368, 372  
 server-side validation  
     example, 311  
     introduced, 297

- login.php script, 317
- responsibilities of the server, 319
- session settings and Local Shared Objects, 378
- setBackground-color function, breadcrumb navigation example, 362
- setBreadcrumb function, breadcrumb navigation example, 362
- setCustomVolume function, random track overlay modification, 234
- setInterval function
  - MP3 player example, 242
  - random motion example, 98
  - zooming text effect, 154
- setMask method, vertical sliding transition effect, 181
- setMaxValue method, Scrubber component, 283
- setMinValue method, Scrubber component, 282
- setPan method, dynamic panning control example, 209
- setProgress method, NetStream class, 264
- setRGB method, 84
- setVal method, Scrubber component, 282
- setVolume method, dynamic volume control example, 208
- shadow effects
  - creating the illusion of movement, 291
  - pie chart example, 424
  - registration form example, 329
  - subliminal navigation example, 67
- SharedObject class, News Sweeper application, 383
- shiver function, subliminal navigation example, 69, 71
- short-circuit execution, 410
- Show Code Hint button, Actions Panel, 15
- show method, Alert class, 307
- showHelpWindow function, form-based navigation example, 338
- showMonthInfo method, News Sweeper application, 377
- showTeaser function, form-based navigation example, 337
- silky fading opacity effect, 43–49
  - modifications, 48
- simple animation masking example, 102
  - modifications, 108
- simple blog reader (*see* News Sweeper application)
- simple login form example, 298
  - hard-coding credentials, 321
  - modification, 308
  - submitting contents for validation, 311
- simple navigation example, 34
  - adding the ActionScript, 38
  - modifications, 43
- simple opacity transition, 173
  - modifications, 176
- SimpleMovement function, 93
- simplicity, navigational systems, 28
- sine functions (*see* Math.sin function)
- single-frame movies, 90
- size changes, random (*see also* scaling)
  - flame effect animation, 124
  - raindrops animation example, 112
- slide-based applications, Flash MX 2004 Professional, 5
- slider components, video scrubber device, 279
- SlideTransition method, vertical sliding transition effect, 181
- sliding text effect, 178
  - horizontal sliding transition, 182
  - vertical sliding transition, 178

---

smoothness of movement, onEnter-Frame event handlers, 54  
snazzy form example, 323  
snowflake effect (*see* falling text effect)  
softening edges, 309, 329, 424  
Sound class, ActionScript, 205  
sound effects, 203–245  
    adapting the video scrubber device for, 288  
    availability and selection, 204  
    dynamic panning control example, 209  
    dynamic volume control example, 206  
    embedded video synchronization, 250  
    importing and exporting clips, 205  
    mini sound player example, 211  
    MP3 and ID3 version support, 236, 243  
    MP3 player example, 236  
    neon text effect crackling, 134  
    playing silently, 234  
    random track overlay effect, 221  
    random track sequencer effect, 218  
    when to use, 203  
sound file, fizzle.mp3, 136  
source control, Flash MX 2004 Professional, 6  
spell checking, Flash MX 2004, 4  
spelling variable names, 394  
Spin method, circulating text effect, 163  
spring parameter, bouncing text effect, 166  
SQL (Structured Query Language), News Sweeper application, 370  
SQL Server 2000, 370  
stacking order  
    bringing objects to the front, 81  
    raindrops animation example, 115  
    sticky note application, 391  
    glowing bulb effect, 141  
    placement within, 98, 111  
    raindrops animation example, 113  
    tabbed navigation example, 82  
stage boundary  
    keeping moving graphics within, 102  
    offstage instances, 388  
stage size, Flash and Flash player, 199  
static imagery, illusion of movement, 291  
static text effects, 129  
static text fields, scripted masking example, 103  
static text objects, 150  
step size, randomizing cloud movement, 116  
Step... options, Debugger Panel, 404, 407  
    data validation debugging example, 410  
stereophonic panning, 209  
sticky note application, 385–392  
    modifications, 391  
stop function  
    introduced, 12  
    questionnaire form example, 351  
stopAllSounds function  
    mini sound player example, 214  
stopAllSounds function, mini sound player, 213  
storyboarding text effects, 127  
streaming media components  
    Flash MX 2004 Professional new feature, 248  
#strict pragma, 18  
style sheets (*see* CSS)  
StyleSheet class, 419  
subliminal navigation, 65  
    modifications to the example, 71  
subtle use of video, 288  
subtlety in animation, 89

- swapDepths method
  - sticky note application, 391
  - tabbed navigation example, 81–82
- swaying effect, falling text modification, 190
- sweeper layer, dazzling chrome effect, 131
- Symbol Properties window, 95
- syntax errors, ActionScript, 393–395

## T

- tabbed interface example, 72
  - adding multiple tabs, 82
  - control code, 79
  - converting tabs to components, 77
  - creating the interface, 75
  - creating the tabs, 74
  - modifications, 83
- television, inspiration from, 128
- templates, Flash MX 2004, 4
- Test Movie option, 99
- testing and usability, 29
- text
  - aspects affecting search engine listings, 427
  - separating into characters, 144, 169
  - styling using CSS, 415
- text effects
  - 3D barrel roll, 195
  - advanced effects, 184
  - falling text effect, 184
  - importance of planning, 127
  - lighting effects, 130–143
  - motion effects, 143–168
  - static and interactive effects, 128
  - Timeline Effects, 201
  - when not to use, 129
  - when to use, 128

- text fields (*see* dynamic text fields; static text fields)
- text labels
  - adding to graphics, 106
  - animating, 107
- text messages, cycling through, 173
- text objects
  - static, creating, 155
- text objects, static, 150
- TextArea components
  - News Sweeper application, 371
  - registration form example, 330
- textbooks, inspiration from, 88
- TextField.StyleSheet class, 419
- TextInput components
  - adding a focus glow, 308
  - form field validation example, 408
  - registration form example, 329
  - simple login form example, 299
- thank you messages, 331
  - registration form example, 341
- this keyword, ActionScript, 10
- 3D barrel roll effect, 195
  - modifications, 200
- threeDee method, 3D barrel roll effect, 198
- three-dimensionals, screen representation, 198
- throwing errors, 399
- time remaining display, mini sound player, 217
- timeline animation example, 90
- Timeline Effects, 201
  - accessing, 23
  - animation and, 20
  - available options, 201
  - categories, 23
  - Fade In command, 24
  - Flash MX 2004 new feature, 5
  - processing requirements, 201
  - third party, 202

- third-party, 5
  - use in tweening, 88
  - timeline manipulation and scripting, 8
  - Timeline Pane, Screen Outline Pane, 299
  - timeline tweening (*see* tweening)
  - timelines
    - best practices for working with, 22
    - FLV files inaccessible through, 250
    - length adjustment, 259
  - timelines and animation effects, 20
  - title tags, 427
  - TopStyle 3.0 editor (Bradbury Software), 417
  - \_totalframes property, 276
  - trace function, 396
    - breakpoints as alternatives, 405
    - questionnaire form example, 352
  - track options, Video Import Wizard, 256
  - transitions, Transform/Transition Timeline Effects, 201
  - transparency (*see* \_alpha values)
  - triggerPoint parameter, zooming text effect, 157–158
  - trigonometric functions, ActionScript, 162, 190
    - (*see also* Math.\* functions)
  - try-catch blocks, 399
  - tweening
    - ActionScript advantages over, 7
    - for animation, 87, 90
    - raindrops animation example, 110
    - Timeline Effects and, 88
  - two-dimensional view of 3D positions, 198
  - type mismatches, 394
- U**
- UI Components section, components tabulated, 294
  - unbalanced identifiers, 394
  - underscores, naming Flash objects, 20
  - UpdateProgressBar function, external .flv file access example
    - random movie loader, 272
  - UpdateProgressBar function, external FLV file access example, 267
  - updating indirectly, 392
  - urban environment effects, 140
  - urlencode function, PHP, 360
  - usability, 27
    - form design and, 296
    - testing, 29
  - user interaction (*see* interactivity with users)
  - user interfaces
    - components relevant to forms, 294
    - mini sound player example, 211
    - random track overlay effect, 221
    - random track overlay modifications, 229
    - separating static elements, 343
    - value of simplicity, 28
  - user perspectives
    - 3D barrel roll effect, 199
    - evaluating animation, 89
    - evaluating sound effects, 204
    - forms design and, 297
  - user-driven motion example, 118
  - users
    - focussing attention with text effects, 128
    - risk of overloading, 129
- V**
- validateEmail function, form-based navigation, 340
  - validateForm function, form-based navigation, 340
  - validateuser function
    - login form example, 318

validation  
  form fields, debugging, 407  
  forms design and, 297  
  simple login form example, 305

variables  
  (*see also* global variables)  
  debugging using watches, 412  
  misspelled, 394  
  setting outside of Flash, 355  
  storing values in, 11  
  type mismatches, 394

Variables tab, Debugger Panel, 402

vector-based graphics  
  compared to bitmaps, 6  
  creating objects using Fireworks, 76

vertical navigation systems, 32, 49–55

vertical scaling, dynamic volume control  
  example, 207

vertical sliding transition, 178  
  modifications, 182

video effects, 247–292  
  capturing movie clips, 251  
  creating a video wall, 273  
  external FLV file access, 257  
  importing a movie clip, 251  
  options for including, 249  
  planning stage, 248  
  processing software, 251  
  simulating by animating static images, 289  
  subtle use of video, 288  
  techniques for integrating, 249  
  video scrubber device, 279  
  when to use, 248

Video Import Wizard, 251  
  advanced settings, 255  
  compression profile settings, 253  
  cropping, 256  
  Flash MX 2004 new feature, 4, 247  
  track options, 256  
  video wall effect, 274

video scrubber device, 279

video wall effect, 273  
  modifications, 276

virtual camera, 199

visibility, toggling, 339

Visual SourceSafe (Microsoft), 6

Visual Studio (Microsoft), 5

volume controls  
  dynamic volume control example, 206  
  mini sound player example, 214  
  random track overlay effect, 226, 229

volumeListener object, random track overlay modification, 234

## W

Watch tab, Debugger Panel, 403, 412

water animation, subtle video example, 291

Waterfall method, falling text effect, 186

wave\_away.flv file, 259  
  video scrubber device, 281  
  video wall effect, 274

WaveFade method, dynamic volume control example, 207

Wavelab (Steinberg), 204

WavePan method, dynamic panning control example, 210

Web server traffic, masking example, 102

whitespace, ignoreWhite property, 241

Window components  
  form-based navigation example, 337–338  
  registration form example, 332

## X

XML class, News Sweeper application, 369

---

XML documents  
  returned by News Sweeper script,  
    372  
  sticky note application modifica-  
    tions, 392  
XML playlists, MP3 player, 236, 238

## **Z**

zooming in effect, 149  
  modifications, 152  
zooming out effect, 154  
  modifications, 157

## What's Next?

If you've enjoyed these chapters from *The Flash Anthology: Cool Effects & Practical ActionScript*, why not order yourself a copy?

In the rest of the book, you'll learn how to put Flash to full use in a number of practical situations. In particular, you'll learn how to communicate with external data sources including Web databases and server-side languages, how to use the new capabilities of Flash MX 2004 to create rich user interfaces with Flash Form Applications, and how to stream video to produce more immersive Web experiences.

And there are plenty more simple ActionScript effects waiting to be discovered too. Here are just a few:

- ❑ slick navigation, from simple menus to spinning gadgets
- ❑ pop-up tabbed interfaces
- ❑ a sputtering neon sign, complete with sound effects
- ❑ wiggling, spinning, circling, fading, and bouncing text
- ❑ spinning objects in 3D under the user's control
- ❑ And a whole lot more...

And as with all SitePoint books, you'll also gain access to the code archive download, so you can try out all the examples without retyping.

[Order now and get it delivered to your doorstep!](#)