Visual

Read Less-Learn More™

Flash™ ActionScript

Your visual blueprint for creating Flash-enhanced Web sites

Web development tools on CD-ROM!

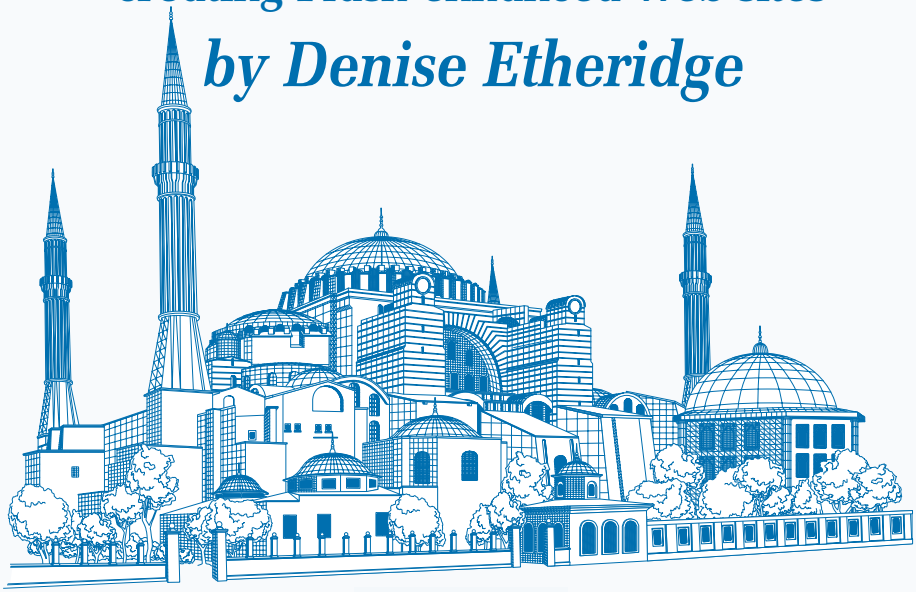
- Macromedia Flash, FreeHand, and Flash Player trial versions
- Plus sample graphics, a searchable e-version of the book, and more

Visit us at www.hungryminds.com

Flash™ ActionScript

Your visual blueprint for creating Flash-enhanced Web sites

by Denise Etheridge



Visual™

From

maranGraphics®

&

Hungry Minds™

Best-Selling Books • Digital Downloads • e-Books • Answer Networks • e-Newsletters • Branded Web Sites • e-Learning

New York, NY • Cleveland, OH • Indianapolis, IN

Flash™ ActionScript: Your visual blueprint for creating Flash-enhanced Web sites

Published by
Hungry Minds, Inc.
909 Third Avenue
New York, NY 10022

Copyright © 2002 Hungry Minds, Inc.

Certain designs/text/illustrations Copyright © 1992-2002 maranGraphics, Inc., used with maranGraphics' permission. All rights reserved. No part of this book, including interior design, cover design, and icons, may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording, or otherwise) without the prior written permission of the publisher.

maranGraphics, Inc.
5755 Coopers Avenue
Mississauga, Ontario, Canada
L4Z 1R9

Library of Congress Control Number: 2001099316

ISBN: 0-7645-3657-5

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

1V/SQ/QS/QS/IN

Distributed in the United States by Hungry Minds, Inc.

Distributed by CDG Books Canada Inc. for Canada; by Transworld Publishers Limited in the United Kingdom; by IDG Norge Books for Norway; by IDG Sweden Books for Sweden; by IDG Books Australia Publishing Corporation Pty. Ltd. for Australia and New Zealand; by TransQuest Publishers Pte Ltd. for Singapore, Malaysia, Thailand, Indonesia, and Hong Kong; by Gotop Information Inc. for Taiwan; by ICG Muse, Inc. for Japan; by Intersoft for South Africa; by Eyrolles for France; by International Thomson Publishing for Germany, Austria and Switzerland; by Distribuidora Cuspide for Argentina; by LR International for Brazil; by Galileo Libros for Chile; by Ediciones ZETA S.C.R. Ltda. for Peru; by WS Computer Publishing Corporation, Inc., for the Philippines; by Contemporanea de Ediciones for Venezuela; by Express Computer Distributors for the Caribbean and West Indies; by Micronesia Media Distributor, Inc. for Micronesia; by Chips Computadoras S.A. de C.V. for Mexico; by Editorial Norma de Panama S.A. for Panama; by American Bookshops for Finland.

For U.S. corporate orders, please call maranGraphics at 800-469-6616 or fax 905-890-9434.

For general information on Hungry Minds' products and services, please contact our Customer Care Department within the U.S. at 800-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

For sales inquiries and reseller information, including discounts, premium and bulk quantity sales, and foreign-language translations, please contact our Customer Care Department at 800-434-3422, fax 317-572-4002, or write to Hungry Minds, Inc., Attn: Customer Care Department, 10475 Crosspoint Boulevard, Indianapolis, IN 46256.

For information on licensing foreign or domestic rights, please contact our Sub-Rights Customer Care Department at 212-884-5000.

For information on using Hungry Minds' products and services in the classroom or for ordering examination copies, please contact our Educational Sales Department at 800-434-2086 or fax 317-572-4005.

For press review copies, author interviews, or other publicity information, please contact our Public Relations department at 317-572-3168 or fax 317-572-4168.

For authorization to photocopy items for corporate, personal, or educational use, please contact Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, or fax 978-750-4470.

Screen shots displayed in this book are based on pre-released software and are subject to change.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND AUTHOR HAVE USED THEIR BEST EFFORTS IN PREPARING THIS BOOK. THE PUBLISHER AND AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS BOOK AND SPECIFICALLY DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THERE ARE NO WARRANTIES WHICH EXTEND BEYOND THE DESCRIPTIONS CONTAINED IN THIS PARAGRAPH. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES REPRESENTATIVES OR WRITTEN SALES MATERIALS. THE ACCURACY AND COMPLETENESS OF THE INFORMATION PROVIDED HEREIN AND THE OPINIONS STATED HEREIN ARE NOT GUARANTEED OR WARRANTED TO PRODUCE ANY PARTICULAR RESULTS, AND THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY INDIVIDUAL. NEITHER THE PUBLISHER NOR AUTHOR SHALL BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGES, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR OTHER DAMAGES.

Trademark Acknowledgments

Hungry Minds, the Hungry Minds logo, Visual, the Visual logo, Read Less - Learn More and related trade dress are registered trademarks or trademarks of Hungry Minds, Inc., in the United States and/or other countries and may not be used without written permission. The maranGraphics logo is a registered trademark or trademark of maranGraphics, Inc. Flash, Director Shockwave Studio, Freehand and Dreamweaver UltraDev Copyright © 1994-2000. Macromedia, Inc. 600 Townsend Street, San Francisco, CA 94103 USA. All Rights Reserved. Contains Macromedia Flash software by Macromedia, Inc. Copyright © 1995-1999 Macromedia, Inc. All rights reserved. Macromedia, Flash, Director Shockwave, Freehand, Dreamweaver and UltraDev are trademarks or registered trademarks of Macromedia, Inc. in the United States and/or other countries. All other trademarks are the property of their respective owners. Hungry Minds, Inc. and maranGraphics, Inc. are not associated with any product or vendor mentioned in this book.

FOR PURPOSES OF ILLUSTRATING THE CONCEPTS AND TECHNIQUES DESCRIBED IN THIS BOOK, THE AUTHOR HAS CREATED VARIOUS NAMES, COMPANY NAMES, MAILING, E-MAIL AND INTERNET ADDRESSES, PHONE AND FAX NUMBERS AND SIMILAR INFORMATION, ALL OF WHICH ARE FICTITIOUS. ANY RESEMBLANCE OF THESE FICTITIOUS NAMES, ADDRESSES, PHONE AND FAX NUMBERS AND SIMILAR INFORMATION TO ANY ACTUAL PERSON, COMPANY AND/OR ORGANIZATION IS UNINTENTIONAL AND PURELY COINCIDENTAL.

Permissions

maranGraphics

Certain text and illustrations by maranGraphics, Inc., used with maranGraphics' permission.

Hungry Minds™ is a trademark of Hungry Minds, Inc.

U.S. Corporate Sales
Contact maranGraphics at (800) 469-6616 or fax (905) 890-9434.

U.S. Trade Sales
Contact Hungry Minds at (800) 434-3422 or (317) 572-4002.

Flash™ ActionScript

Your visual blueprint for creating Flash-enhanced Web sites

maranGraphics is a family-run business located near Toronto, Canada.

At **maranGraphics**, we believe in producing great computer books — one book at a time.

maranGraphics has been producing high-technology products for over 25 years, which enables us to offer the computer book community a unique communication process.

Our computer books use an integrated communication process, which is very different from the approach used in other computer books. Each spread is, in essence, a flow chart — the text and screen shots are totally incorporated into the layout of the spread. Introductory text and helpful tips complete the learning experience.

maranGraphics' approach encourages the left and right sides of the brain to work together — resulting in faster orientation and greater memory retention.

Above all, we are very proud of the handcrafted nature of our books. Our carefully-chosen writers are experts in their fields, and spend countless hours researching and organizing the content for each topic. Our artists rebuild every screen shot to provide the best clarity possible, making our screen shots the most precise and easiest to read in the industry. We strive for perfection, and believe that the time spent handcrafting each element results in the best computer books money can buy.

Thank you for purchasing this book. We hope you enjoy it!

Sincerely,

Robert Maran

President

maranGraphics

Rob@maran.com

www.maran.com

www.hungryminds.com/visual

CREDITS

Acquisitions, Editorial, and Media Development

Project Editor
Jade L. Williams

Acquisitions Editor
Jen Dorsey

Product Development Supervisor
Lindsay Sandman

Development Editors
Dana Lesh
Kathleen McFadden

Technical Editor
Kyle D. Bowen

Editorial Manager
Rev Mengle

Media Development Manager
Laura Carpenter Van Winkle

Permissions Editor
Laura Moss

Media Development Specialist
Megan Decreane

Production

Book Design
maranGraphics®

Production Coordinator
Maridee Ennis

Layout and Graphics
Melanie DesJardins
LeAndra Johnson
Kristin McMullan
Jill Piscitelli
Betty Schulte

Screen Artists
Ronda David-Burroughs
Mark Harris
Jill A. Proll

Cover Illustration
David E. Gregory

Proofreaders
Laura L. Bowman, Susan Moritz,
Carl Pierce, Marianne Santy

Indexer
Sherry Massey

Special Help
Macromedia, Inc.,
Malinda McCain

ACKNOWLEDGMENTS

Hungry Minds Technology Publishing Group: Bob Ipsen, Vice President & Publisher; Richard Swadley, Vice President & Executive Group Publisher; Mary Bednarek, Vice President & Publisher, Networking; Joseph Wikert, Vice President & Publisher, Web Development Group; Mary C. Corder, Editorial Director, Dummies Technology; Andy Cummings, Editorial Director, Dummies Technology; Barry Pruett, Vice President & Publisher, Visual/Graphic Design

Hungry Minds Manufacturing: Ivor Parker, Vice President, Manufacturing

Hungry Minds Marketing: John Helmus, Assistant Vice President, Director of Marketing

Hungry Minds Production for Branded Press: Debbie Stailey, Production Director

Hungry Minds Sales: Michael Violano, Vice President, International Sales and Sub Rights

ABOUT THE AUTHOR

Denise Etheridge: Denise Etheridge is the president of BayCon Group Incorporated, a consulting firm specializing in software installation, implementation, and training. The BayCon Group Web site features online software tutorials. You can visit http://www.baycongroup.com/flash/00_flash.htm, for Flash tutorials.

AUTHOR'S ACKNOWLEDGMENTS

I would like to say thank you to all of people who provided me with assistance or support during the development of this book. Thank you to Neil Salkind and Jennifer Dorsey for providing me with the opportunity and for encouraging me along the way. Thank you to Jade Williams for her editing and project management. And, thank you to all of the editors and staff at John Wiley who assisted Jade.

I would like to give a special thanks to David Gregory for assisting with the artwork and Malinda McCain for helping me with this and many other projects.

I dedicate this book to Frederick, Catherine, Freddie, and Erskine Etheridge.

TABLE OF CONTENTS

HOW TO USE THIS BOOK XIV

1) GETTING FAMILIAR WITH FLASH

Introduction to Flash and ActionScript 2
Create a Scene 3
Create Layers 4
Create Graphics 6
Create a Symbol 8
Create an Instance 10
Create a Button 12
Create Animation 14
Add Sound 16
Create a Movie Clip 18
Create Static Text Boxes 20
Create Input Text Boxes 22
Create Dynamic Text Boxes 23
Using the Actions Panel 24
Assign ActionScript to a Button 28
Assign ActionScript to a Movie Clip 29
Assign ActionScript to a Frame 30
Publish Movies 32

2) PROGRAMMING WITH ACTIONSCRIPT

Introduction to ActionScript Syntax 36
Test a Movie 38
Enter the Test Environment 40
Add Comments 42
Stop a Movie 44
Play a Movie 46
Jump to a Frame or Scene 48
Set Movie Quality 50
Open a Web Page 52
Communicate with the Flash Player 54
Create Objects Users Can Drag 56
Print a Movie 60
Check Frame Load 62

3) SETTING MOVIE CLIP PROPERTIES

Introduction to Movie Clip Properties 64
Name an Instance 66
Adjust Transparency 68
Make Movie Clips Invisible 70
Rotate Movie Clips 72
Change the Width of Movie Clips 74
Change the Height of Movie Clips 76
Scale the Width of Movie Clips 78
Scale the Height of Movie Clips 80
Move Movie Clips Across the Stage 82
Move Movie Clips Up and Down 84
Set Movie Clip Quality 86

4) WORKING WITH VARIABLES AND STRINGS

Introduction to Variables and Data Types 88
Assign a Value to a Variable 90
Assign an Object or Movie Clip to a Variable 92
Load Variables 94
Extract a Character from a String 96
Extract a Substring from a String 98
Concatenate Strings 100
Obtain the Length of a String 102
Search a String 104
Change the Case of a String 106
Convert Strings, Numbers, and Booleans 108

5) WORKING WITH OPERATORS

Understand Precedence 110
Add Numeric Values 112
Subtract Numeric Values 114
Multiply Numeric Values 116
Divide Numeric Values 118
Find the Modulo 120
Set Values with the Increment Operator 122
Set Values with the Decrement Operator 124
Using Less Than or Greater Than 126

Using Less Than or Equal to or Greater Than or Equal to 127
Using Equality and Inequality Operators 128
Using Compound Assignment Operators 130
Using Logical Operators 132

6) CHANGING THE SCRIPT FLOW

Using if to Test a Condition 134
Using if with else 136
Using else if 138
Create a Conditional Loop 140
Using for Loops 142

7) USING THE MOVIE CLIP AND ARRAY

Introduction to Objects 144
Using the MovieClip Object 146
Attach a Movie Clip 148
Get Bounds 152
Swap Depths 154
Check MovieClip Load 156
Detect Collision 158
Get the x- and y-Coordinates 160
Create an Array 162
Find the Length of an Array 166
Add Elements to an Array 168
Remove Elements from an Array 170
Extract or Reverse an Array 172
Convert an Array to a String 174

8) WORKING WITH OBJECTS

Change Colors 176
Work with Color Transform Values 178
Using the Mouse Objects and Properties 180
Using the Date Object 182
Get Date and Time Values 184
Set the Date 186

Set Date Values 188
Set Time Values 190
Using Mathematical Functions 192
Raise a Power or Find A Square Root 193
Round Numbers 194
Generate Random Numbers 195
Find Numeric Values 196
Using the Key Object 198
Using the Sound Object 202
Set Volume and Panning 206
Using the Selection Object 208

9) DEMYSTIFYING FUNCTIONS

Get Properties 210
Using Get Timer and Get Version 212
Using Eval 213
Convert a String to a Number 214
Evaluate for Mathematical Errors 216
Create a Scrollable Text Box 218
Create a Custom Function 220

10) WORKING WITH MULTIPLE TIMELINES

View the Hierarchy of Multiple Movies 222
Assign Target Paths 224
Load and Unload Movies and Movie Clips 226
Using Tell Target 230
Using the with Action 232
Duplicate Movie Clips 234

11) CREATING SMART CLIPS

Create Smart Clips 236
Create a Smart Clip Custom Interface 240

12) USING CREATIVE TECHNIQUES

Validate a String 244
Validate a Date 246
Search a Text Box 248
Convert Symbols 252
Create Rotation Effects 254
Create a Coloring Book 256
Emulate Panels 258
Demystify Trigonometric Functions 262
Create Trigonometric Special Effects 266

13) DEBUGGING ACTIONSCRIPT

Using the Debugger 268
Using the Output Window 270
Using List Objects 272
Using List Variables 273
Debug Your Script 274

14) FLASH ACTIONSCRIPT QUICK REFERENCE

Key Code Values 276
URL Encoding Characters 278
List of Keywords 279
Actions Appended with Num 280
Hexadecimal Color Codes 280
Button Handlers 283
Movie Clip Handlers 283
Operators 284
Functions 285
Actions 286
Properties 287

Array Object 288
Boolean Object 288
Color Object 288
Date Object 289
Key Object 290
Math Object 290
Mouse Object 291
MovieClip Object 291
Number Object 292
Object Object 292
Selection Object 292
Sound Object 293
String Object 293

APPENDIX

What's on the CD-ROM 294
Using the E-Version of the Book 296
Hungry Minds, Inc. End-User License Agreement 298

INDEX 300

HOW TO USE THIS BOOK

Flash ActionScript: Your visual blueprint for creating Flash-enhanced Web sites uses straightforward examples to teach you how to create powerful and dynamic Web sites. The coding style and examples found in this book are used for instructional purposes. Once you are comfortable working with Flash ActionScript, you can use the coding styles and methods that suit your needs.

To get the most out of this book, you should read each chapter in order, from beginning to end. Each chapter introduces new ideas and builds on the knowledge learned in previous chapters. Once you become familiar with Flash ActionScript, this book can be used as an informative desktop reference.

Who This Book Is For

If you are looking for a resource that will help you get started learning ActionScript, *Flash ActionScript: Your visual blueprint for creating Flash-enhanced Web sites* is the book for you. This book will walk you through the basics that you need to get started and familiarize yourself with the essentials of working with Flash ActionScript.

No prior experience with ActionScript is required, but familiarity with Flash is assumed.

Experience with programming languages is also an asset, but even if you have no programming experience, you can use this book to learn the essentials you need to work with Flash ActionScript.

What You Need To Use This Book

The tasks in this book were developed using Flash 5 for Windows 98, 2000, and NT. To perform the tasks in this book, you need Flash 5 and Flash Player installed on your computer. You should also have a Web browser such Internet Explorer or Netscape.

The Conventions In This Book

A number of typographic and layout styles are used throughout *Flash ActionScript: Your visual blueprint for creating Flash-enhanced Web sites* to distinguish different types of information.

`Courier Font`

Indicates the use ActionScript.

Bold

Indicates information that you must type.

Italics

Indicates a new term being introduced.

Apply It

An Apply It section usually contains a segment of code that takes the lesson you just learned one step further. Apply It sections offer inside information and pointers that can be used to enhance the functionality of your code.

Extra

An Extra section provides additional information about the task you just accomplished. Extra sections often contain interesting tips and useful tricks to make working with Flash ActionScript easier and more efficient.

The Organization Of This Book

Flash ActionScript: Your visual blueprint for creating Flash-enhanced Web sites contains 14 chapters.

The first chapter, Getting Familiar with Flash, provides a review of Flash introduces you to the Actions panel.

Chapter 2, Programming with ActionScript, introduces you to ActionScript. You learn to play and stop movies, adjust movie quality, jump to a frame or scene, jump to a URL, check whether a frame is loaded, create draggable objects, communicate with Flash Player, and Print Movies. This chapter illustrates ActionScript by stepping you through examples and explaining the syntax of several actions.

Chapter 3, Setting Movie Clip Properties, demonstrates how you can use properties. Properties are the attributes associated with a movie clip. For example, the _visible property determines whether a movie clip is visible or hidden. This chapter explains properties and steps you through sample scripts that use properties.

Chapter 4, Working with Variables and Strings, shows you how to use variables and manipulate string. Variables can be viewed as containers that store information. You can assign information to a variable or you can use a variable

to collect information from the user, record what a user has done, change values as a movie plays, or evaluate whether a condition is true or false. A string is any sequence of characters consisting of any combination of letters, numbers, or punctuation marks. The ActionScript String object has several methods that enable you to manipulate strings. This chapter explains variables and strings and steps you through sample scripts that use variables and strings.

Chapter 5, Working with Operators, explains operators and steps you through sample scripts that use operators. An operator is a character used to specify how to combine, compare, or modify a value in an expression. For example, the plus (+) operator can be used to add two numbers; the Minus (–) operator can be used to subtract two numbers.

Chapter 6, Changing the Script Flow, steps you through actions that enable you to change the flow of script. In ActionScript, each statement is executed in order from top to bottom unless you change the flow of a script.

Chapter 7, Using the Movie Clip and Array Objects, introduces the MovieClip object, which is the most important object in ActionScript. Since each movie clip has its own Timeline, each movie clip can be a complete animation. Arrays enable you to group values together and are useful when you need to store and retrieve lists of data. The ActionScript Array object has several methods that enable you to manipulate arrays.

Chapter 8, Working with Objects, steps you through ActionScript examples that use objects contains several objects that you can use to manipulate symbols and data. For example, the Math object is used to manipulate numbers. You can use the Math object's round method to round a number. The Color object is used to manipulate color. You can use the methods of the Color object to change the color of a movie clip.

Chapter 9, Demystifying Functions, teaches you how to create you own custom functions. Functions are reuseable blocks of ActionScript. You can pass values to a function or you can use a function to obtain values. For example, you use the getVersion function to get the version of Flash Player. This chapter explains functions and steps you through sample scripts that use functions.

Chapter 10, Working with Multiple Timelines, explains how to work with multiple Timelines in a single movie. You can lay out the sequence of a Flash movie on a Timeline. Movie clips are mini Flash movies. They each have their own Timeline and their own properties. You can embed a movie clip within a Flash movie. In fact, multiple instances of a movie clip symbol can be embedded inside of each other. This chapter steps you through sample scripts that work with multiple Timelines.

Chapter 11, Creating Smart Clips, teaches you how to create smart clips. A smart clip is a movie clip with ActionScript that can be reprogrammed without using the Actions panel. This enables you to create ActionScripts that people with no programming ability can modify.

Chapter 12, Using Creative Techniques, pulls all basics of ActionScript together by combining actions and demonstrating techniques that you can use when creating a Flash movie.

Chapter 13, Debugging ActionScript, is on debugging ActionScript. Flash movies that contain ActionScript can become complex. Flash provides you with several tools that can help you troubleshoot your movie.

The final chapter, Flash Action Script Quick Reference, contains useful tables of reference material and a summary of commands.

What Is On The CD-ROM

The CD-ROM in the back of this book contains the sample code from each of the two-page lessons, as well as the code from most of the Apply It sections. This saves you from having to type the code and helps you quickly get started creating Flash ActionScript programs. The CD-ROM also contains several shareware and evaluation versions of programs that can help you work with Flash ActionScript. An e-version of this book is also available on the companion disc.

Chapter 1 provides a very brief overview of how to create text, input text boxes, dynamic text boxes, buttons, and movie clips. Since this book assumes you know Flash, all of the text, input text boxes, dynamic text boxes, buttons and movie clips used in the sample files have been created for you.

The sample files all have an FLA extension. Copy the files to the directory of your choice and click Files ➪ Open on the Flash menu to open them.

INTRODUCTION TO FLASH AND ACTIONSCRIPT

Flash is a graphics and animation software package that enables Web developers to design and deliver low-bandwidth animations and presentations referred to as movies. With Flash, you can create interactive Web pages with both motion and sound. This book is about ActionScript, the scripting language that enables you to add interactivity to your Flash movie. With ActionScript, you can have your Flash movie respond to mouse clicks and key presses, or you can request information from the user and have your Flash movie respond to the information provided.

FLASH INTERFACE

When you open Flash, the screen shown here appears:

LAYER

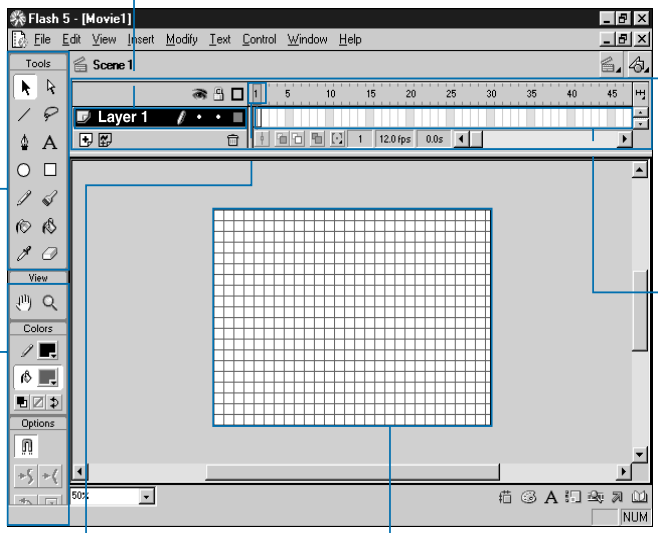
Layers are transparent sheets that you can see through to the layer beneath until you add color. You create the graphics and animations that you want to use in your movie on layers.

TOOLBOX

The toolbox provides you with tools that enable you to select objects on the Stage, draw and paint graphics, and create text boxes. Use the toolbox to create and modify graphics and text.

MODIFIERS

Once you select a tool, modifiers enable you to change stroke colors, fill colors, and the effect of tools. Use modifiers to set tool effects.



TIMELINE

The Timeline enables you to lay out the sequence of your movie. The Timeline includes frames, layers, and a playhead.

FRAMES

You store the individual images that make up a movie in frames. The playhead moving through successive frames displays the contents of each frame.

PLAYHEAD

As the playhead moves across the Timeline, Flash displays the image in each frame on the Stage. You create animation by changing the image in successive frames.

STAGE

You can create or import graphics. Use the Stage to create your movie.

CREATE A SCENE

Using scenes enables you to separate your movie into sections. For example, you can have a scene that displays the title of the movie, a scene that plays the movie, and a final scene that lists the characters. Each scene has its own Timeline and can include its own animation.

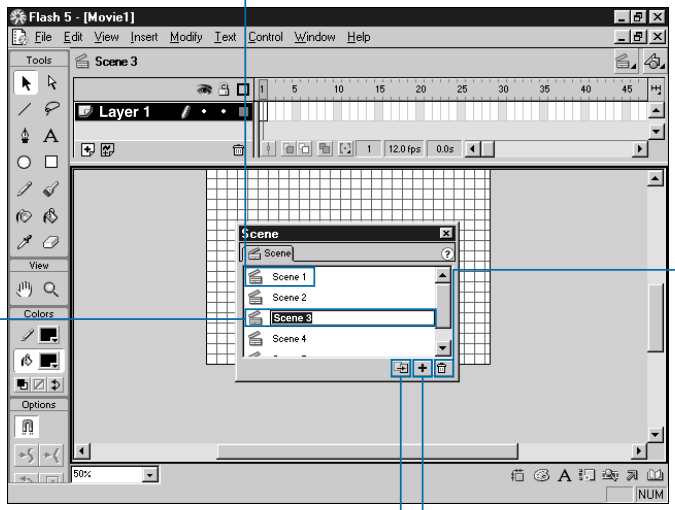
To create a scene, click View ➪ Scene from the menu. A new Stage and Timeline appears in which you can create your scene. By default, Flash names scenes by assigning them sequential numbers, for example, Scene 1 and Scene 2. You can use the Scene panel to add, remove, duplicate, rename, and change the order of scenes. To open the Scene panel, click Window ➪ Panels ➪ Scene from the menu.

SCENE PANEL

To view a scene, click View ➪ Goto from the menu and then choose the scene to which you want to go.

CHANGE THE ORDER OF A SCENE

To change the order of a scene, click a scene name and drag it to a new location.



RENAME A SCENE

To rename a scene, double-click the scene name and type a new name.

DELETE A SCENE

To remove a scene, click the scene name to select it and then click the Delete Scene icon.

DUPLICATE A SCENE

To duplicate a scene, click the Duplicate Scene icon.

ADD A SCENE

To add a scene, click the Add Scene icon.

CREATE LAYERS

When creating a Flash movie, you place individual images and animations on separate layers to prevent them from connecting, erasing, or segmenting each other. You should also use separate layers for sounds, actions, frame labels, and comments. Layers are transparent sheets on which you create the graphics and animations you use in your movie. You can see through each sheet to the layer beneath until you add color.

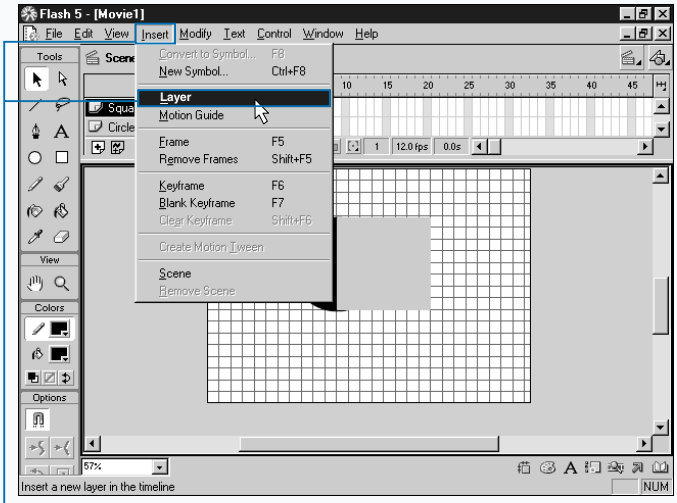
You work on the active layer. You make a layer active by clicking in the layer, or by moving to the Stage and selecting an object on the layer. Flash places a pencil icon next to the layer name of the active layer. Only one layer can be active at a time.

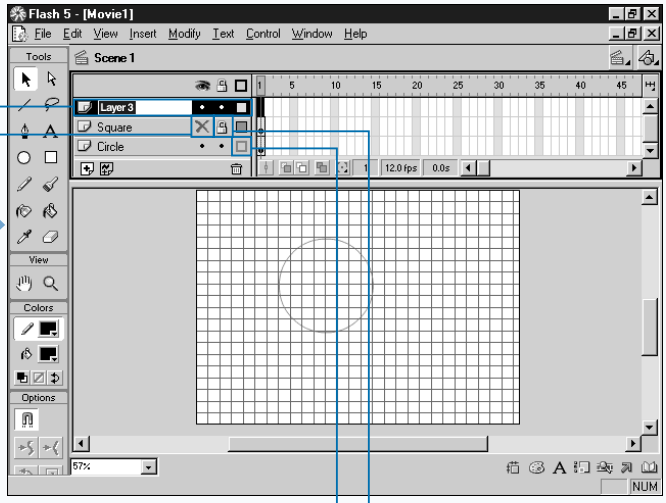
As you create objects, they are stacked according to layer. An object on a lower layer will appear as if it is behind an object on a higher layer. You can change the order of layers and the name of a layer.

You can lock a layer to prevent changes to the layer. Flash uses a padlock icon to indicate a locked layer. You can also hide a layer. Hiding a layer is useful when you want to remove certain objects on the Stage from view to make it easier for you to create graphics. You can delete layers when you no longer need them.

To assist you when you are drawing graphics, you can create guide layers. You can also create motion guide layers to assist you when creating motion tweens. You can use mask layers to create spotlight effects. Flash also provides you with an option that allows you to view a layer as an outline. You can specify the outline color. Adding layers does not increase the file size of your published movie.

CREATE LAYERS





1 Click Insert ➪ Layer to create a layer above the active layer.

■ A new layer appears above the active layer.

2 Double-click and type to change the layer name.

3 Click in the Show/Hide All Layers column to toggle between showing (◆) and (✕) hiding a layer.

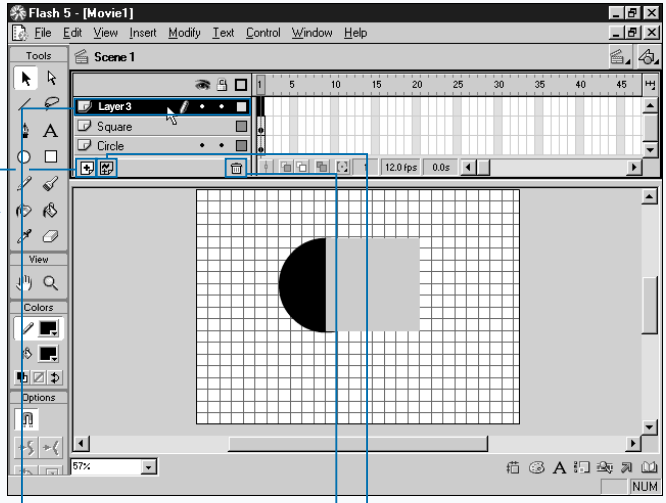
4 Click in the Lock/Unlock All Layers column to toggle between locking (🔒) and unlocking (◆) layers.

5 Click in the Show Layers as Outlines column to toggle between Show Normal (□) and Show As Outline (□).

Extra

When you draw on an object using the Line, Oval, Rectangle, Pencil, or Brush tool, the object segments. Each segment is an individual object that you can select, move, and reshape. When you cover object A with object B, object B connects to, segments, and erases object A. Place objects on separate layers to prevent them from connecting, segmenting, and erasing.

You can use the Layer Properties dialog box to change layer properties. To open the Layers Properties dialog box, choose Modify ➪ Layer from the menu. Use the Name field to name the layer. Select Show to show the layer. Select Lock to lock the layer. Select the Type of layer. Choose from Normal, Guide, Guided, Mask, and Masked. Use the Outline Color option to select the color of the outline. To view a layer as an outline, select View Layer As Outlines. Select a Layer Height. Choose from 100 percent, 200 percent, or 300 percent.

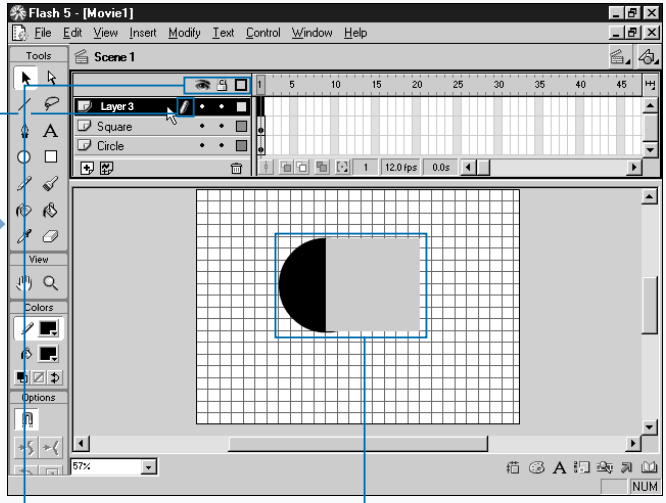


6 Click the layer and drag to change the order of a layer.

7 Click ⊞ to insert a layer.

8 Click ⊞ to add a guide layer.

9 Click 🗑 to delete a layer.



10 Click the Show/Hide, Lock/Unlock, or Outlines icon to affect every layer in the column.

■ The pencil icon indicates this is the active layer.

■ Items on lower layers appear to be behind items on higher layers.

CREATE GRAPHICS

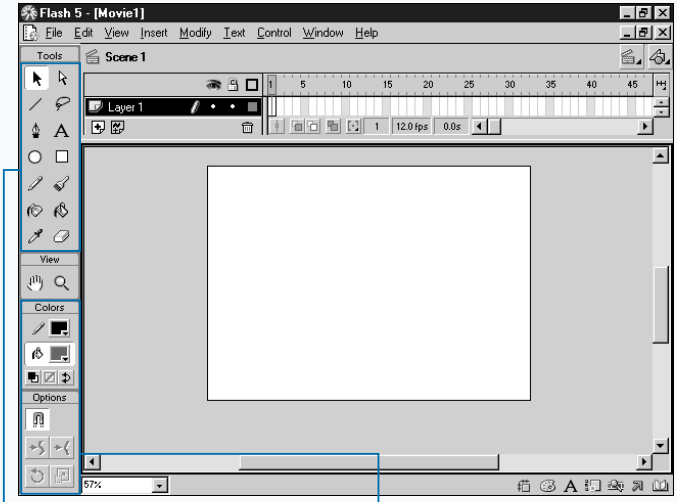
Flash provides you with several tools you can use to create or modify graphics. You use the Line tool to create straight lines, the Oval tool to create circles and ellipses, and the Rectangle tool to create squares and rectangles. You use the Paint Bucket and Brush tools to add color to images.

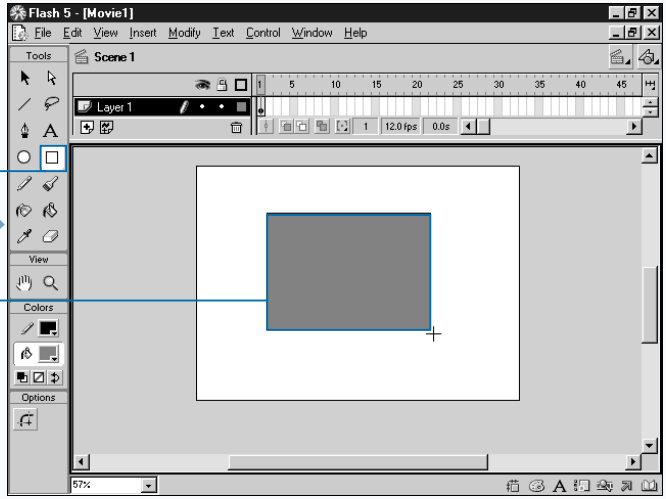
The Pen tool gives you the ability to draw straight lines and smooth curves. You use the Subselect tool to adjust those lines. Stroke lines outline a graphic. You use the Ink Bottle tool to change the stroke line color, width, or style. Fill colors fill the interior of a graphic. You use the Dropper tool to copy fill and stroke attributes from one object to another. The Eraser tool gives you the ability to erase.

You can use the Arrow and Lasso tools to select objects on the Flash Stage. You can group, move, copy, or delete selected objects. Grouping enables you to manipulate several objects as a single object. You use the Arrow tool to select a button or movie clip when you want to associate ActionScript with a button or movie clip.

Flash tools have modifiers you can use to set tool options. Color boxes enable you to set the stroke and fill colors associated with each tool. You can swap the stroke and fill colors, set the color to no color, or set the colors to the default colors. The Arrow tool provides you with options for smoothing, straightening, rotating, and scaling objects. The Rectangle tool provides an option for rounding corners. You can set the size and shape of the Brush. The Eraser offers several modes and a faucet that enables you to erase the selected area. When using the Pencil, you can choose to have straight lines, smooth lines, or free-form lines.

CREATE GRAPHICS





1 Click to select a tool.

2 Click to select modifiers.

3 Move to the Stage and drag to create a shape.

■ You can create rectangles with the Rectangle tool ( ), ellipses with the Oval tool ( ), straight lines with the Line tool ( ), paint with the Brush tool ( ), or free-form lines with the Pencil tool ( ), or you can erase with the Eraser tool ( ).

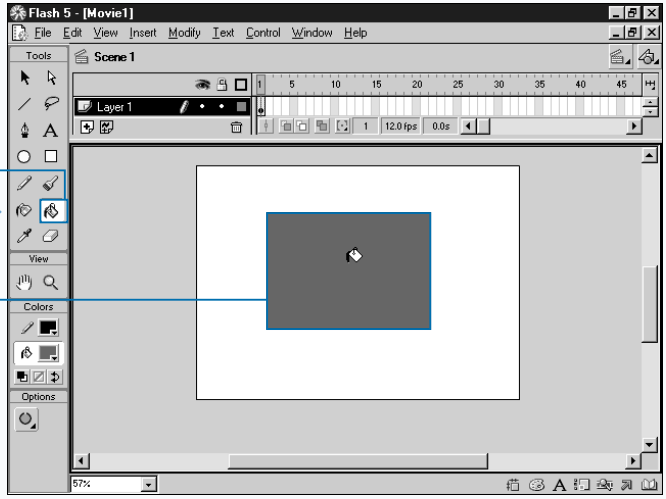
■ You can create a square with or a circle with by holding down the Shift key as you drag.

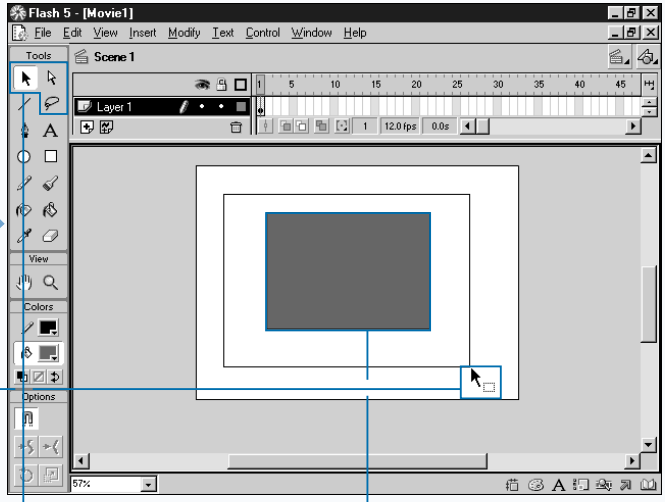
Extra

To hide the Toolbox, click Window ➪ Tools from the menu. To bring the Toolbox back, click Window ➪ Tools again.

Use the Hand tool to move the Stage. Use the Zoom tool to zoom in on the Stage. You can also click View ➪ Zoom In to zoom in on the Stage, or View ➪ Zoom Out to zoom out. To change the magnification, click View ➪ Magnification and select a magnification level.

You can click Windows ➪ Panels ➪ Fill to open the Fill panel. The Fill panel enables you to select color, gradients, or bitmap fills. You can create your own gradients. You can click any bitmap in the Library as a bitmap fill. You can click Windows ➪ Panels ➪ Stroke to select a stroke color, style, or line weight. Clicking Windows ➪ Panels ➪ Mixer enables you to create colors.





■ If you selected the Ink Bottle ( ), Paint Bucket ( ), or Dropper tool ( ), move to the Stage and click the mouse.

■ You can click the stroke line to change the stroke color with , the fill color to change the fill with , or set a stroke or fill color with .

4 Click to select the Lasso or Arrow tool.

■ The pointer changes to the icon that represents the tool you selected.

5 Move to the Stage and drag the mouse to select an object.

■ Use to place a rectangle around the selected area.

■ Use to draw the area you want to select.

CREATE A SYMBOL

Creating and using symbols enables you to share graphics, movie clips, buttons, and sounds among movies. A symbol is a reusable graphic, button, movie clip, or sound. Every Flash movie has a Library, which stores symbols. When you create a symbol, it automatically becomes part of the Library.

Graphics are static images. You can place graphics on the Timeline of a movie, but you cannot attach ActionScript to a graphic.

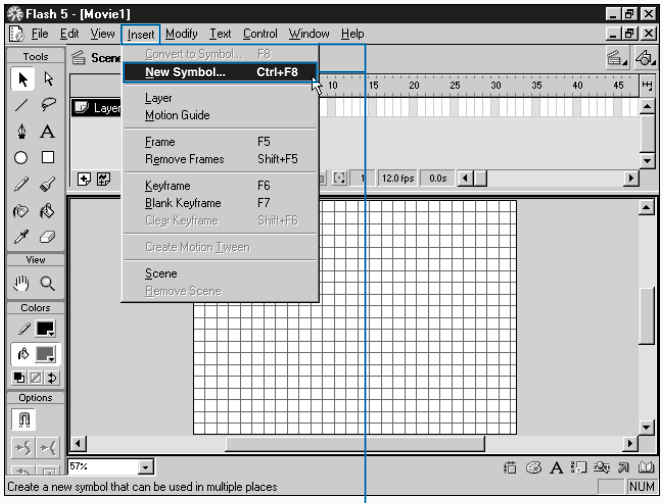
Buttons are interactive graphics that can respond to user actions. You use ActionScript to give Flash instructions on how to respond when the user clicks, rolls the pointer over, or performs some other action in relation to a button. Buttons have their own Timeline.

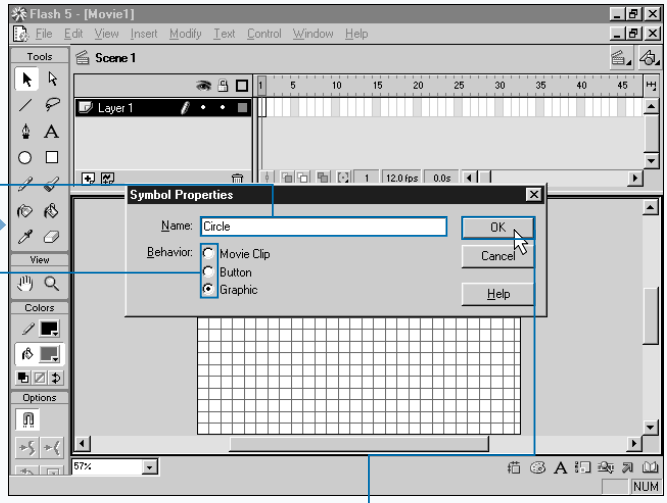
Movie clips are reusable animations that have their own Timeline. You can place a movie clip on a button Timeline to create an animated button. You use ActionScript to give Flash instructions on how to respond when the user clicks, rolls the pointer over, or performs some other action in relation to a movie clip.

Symbols speed up the playback of your movies, because the browser only needs to download them once. Using symbols enables you to share graphics, movie clips, buttons, and sounds among movies. You can set up permanent libraries and link to items in the permanent libraries from any Flash movie. You can open any Flash movie as a shared library to make the library items from that movie available to the current movie. Flash ships with several libraries containing buttons, graphics, and movie clips that you can use in your own movies.

You can create a symbol, import a symbol, or convert an object on the Stage into a symbol. You create symbols in the symbol-editing mode.

CREATE A SYMBOL





■ Make sure nothing on the Stage is selected.

1 Click Insert ➪ New Symbol to open the Symbol Properties dialog box.

■ The Symbol Properties dialog box appears.

2 Type a symbol name.

3 Click to select a behavior.

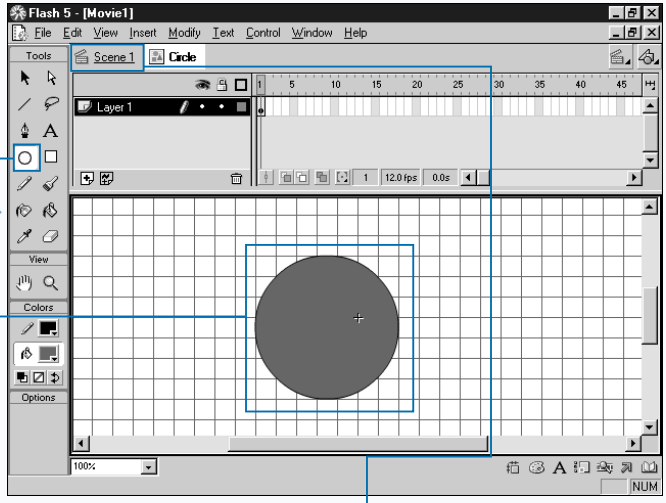
4 Click OK.

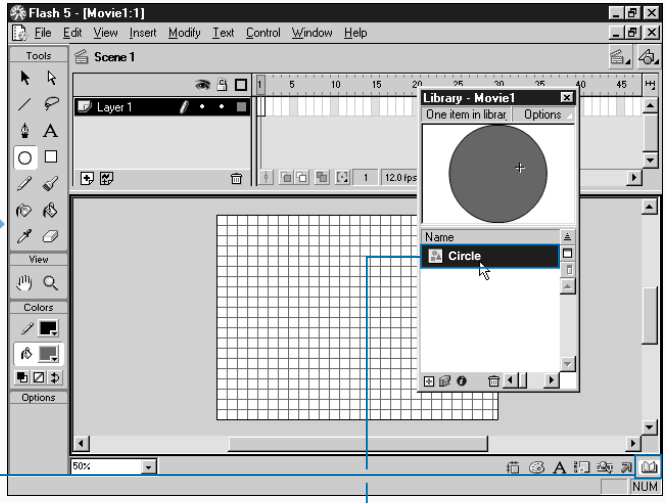
■ Flash changes to the symbol-editing mode.

Extra

You can convert any selection on the Stage into a symbol. To reduce the file size of your movies, consider turning background images into symbols so you can reuse them. To convert an object on the Stage into a symbol, use the Arrow tool to select the object. Click Insert ➪ Convert to Symbol to open the Symbol Properties dialog box. Click to select Movie Clip, Graphic, or Button as the behavior. Click OK.

When you create a symbol from a selection on the Stage, the selection becomes an instance of the symbol. You will no longer be able to edit the object directly. To edit the symbol, you must move to the symbol-editing mode. To move to the symbol-editing mode, use the Arrow tool to select the symbol and click Edit ➪ Edit Symbols. When you have completed editing your symbol, click Edit ➪ Edit Movie to exit the symbol-editing mode. When you modify a symbol, Flash updates every instance of the symbol in the movie. When editing a symbol, you can use all of the drawing tools and you can import content.





5 Click to select a drawing tool and create the symbol content.

6 Create a symbol.

7 Click the Scene button to exit symbol-editing mode.

■ The main Timeline returns.

8 Click the Show Library icon ( ) to open the Library.

9 Click the symbol name to see the symbol in the Item Preview window.

■ The symbol you created appears in the preview window.

CREATE AN INSTANCE

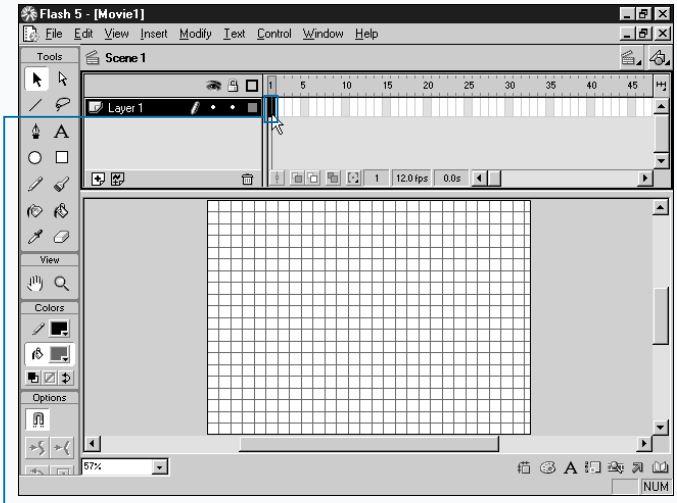
Creating and using instances of a symbol can dramatically reduce the size of your Flash movie. An instance is a copy of a symbol. Each symbol can have an unlimited number of copies — or instances. You can change the color, tint, size, shape, rotation, function, and other properties of an instance. Changes you make to an instance do not change the symbol itself or other instances of the symbol.

Brightness refers to the relative lightness or darkness of an image. Black has a brightness of -100 percent. White has a brightness of +100 percent. Tint refers to the color or hue of an instance, for example, red, blue, or green. Alpha refers to the transparency of an instance. An instance with an alpha property of 0 percent is completely transparent. An instance with an alpha property of 100 percent is completely opaque. You use the Effect panel to change the brightness, tint, or alpha property of an instance.

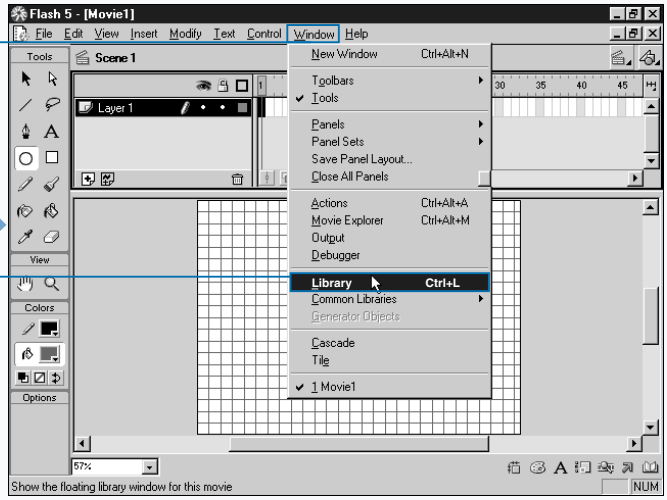
An instance can have a behavior type of graphic, button, or movie clip. Instances inherit their behavior type from the symbol. To change an instance behavior type, use the arrow tool to select the instance, then open the Instance panel. In the Instance panel Behavior field, select from Graphic, Button, or Movie Clip.

Using instances of a symbol dramatically reduces the file size of your movie, because saving instances of an image requires less space than saving a complete description of an image each time you use it. You create an instance by opening the Library and dragging the instance from the Library onto the Stage.

CREATE AN INSTANCE



1 Click to select a keyframe.

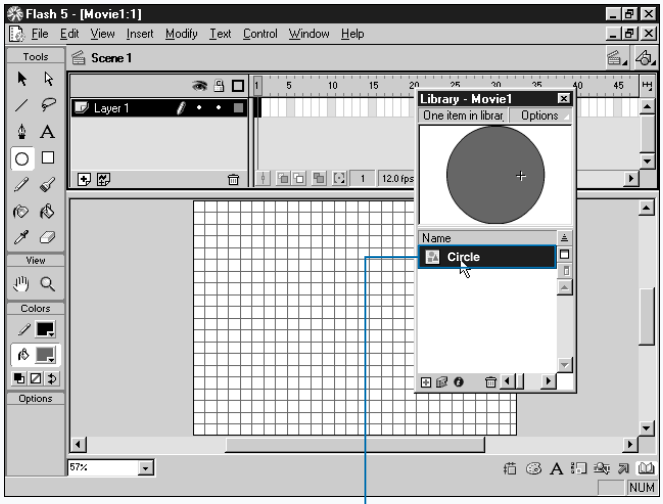


2 Click Window ➪ Library to open the Library.

Extra

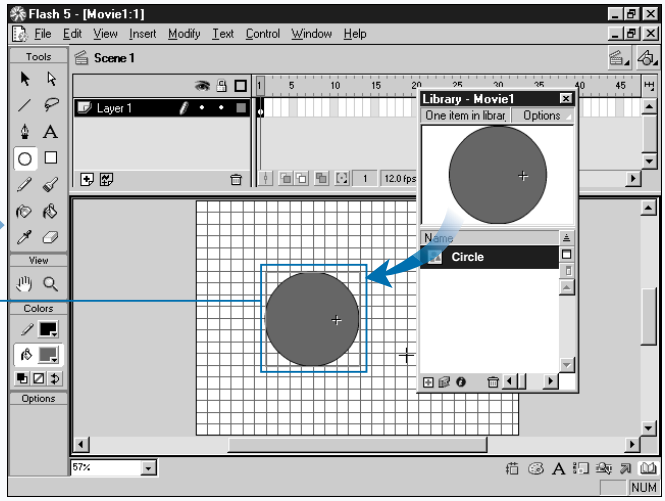
You use the Effect panel to change the brightness, tint, or alpha property of an instance. To open the Effect panel, click Window ➪ Panels ➪ Effect. Select Brightness and use the brightness slider to set the brightness to a value between -100 and 100. Select tint to change color using a color box or by setting Red, Green, and Blue values. Select Alpha and use the Alpha slider to set the brightness to a value between 0 and 100 percent. Select Advanced to adjust both the color and alpha values.

You can use the Instance panel to change the behavior type of an instance. To open the Instance panel, click Window ➪ Panels ➪ Instance or click Modify ➪ Instance on the menu. Use the Arrow Tool to select the instance for which you want to change the behavior. Use the Behavior field to change the behavior of your instance.



■ The Library window appears.

3 Click the symbol name.



4 Drag the symbol from the Library onto the Stage.

■ An instance of the symbol appears on the Stage.

CREATE A BUTTON

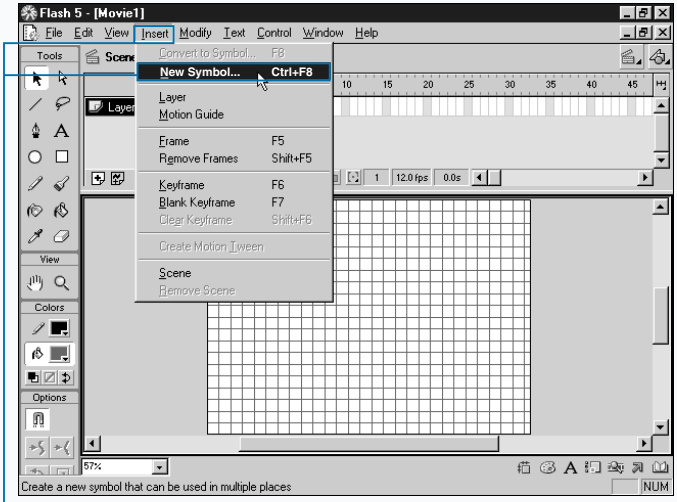
Buttons enable you to add interactivity to your Flash movie. The user clicks a button to cause an action to occur. For example, the user can click a button to open a Web page.

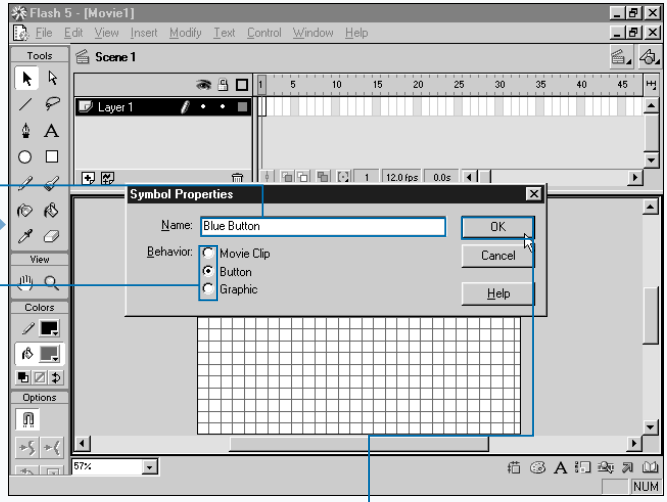
Buttons have four states: Up, Over, Down, and Hit. The Up state is how the button appears to the user when the pointer is not over the button. The Over state is how the button appears to the user when the pointer is over the button. The Down state is how the button appears to the user when the user clicks the mouse while the pointer is over the button. The Hit state defines the area that responds to user actions. Make sure that your Hit state is at least large enough to encompass the graphics used in the other three states. You can make the Hit states larger than the other states. The Hit state is not visible to the user.

When you create a button, Flash presents you with a four-frame timeline, with one frame representing each of the four button states. To define the appearance of a button state, you can create or import a graphic for each of the four states. You can use any graphic or movie clip when defining the appearance of a button state. You use movie clips to create animated buttons. You cannot use another button to define a button state. You can enable a button to test it.

You use ActionScript to define the action that will occur when the user clicks the button. You can use buttons to analyze user input, respond to user input, load or unload a movie, or perform a myriad of other tasks.

CREATE A BUTTON





1 Click Insert ➪ New Symbol to open the Symbol Properties dialog box.

■ The Symbol Properties dialog box appears.

2 Type a symbol name.

3 Click to select Button as the behavior (○ changes to ◉).

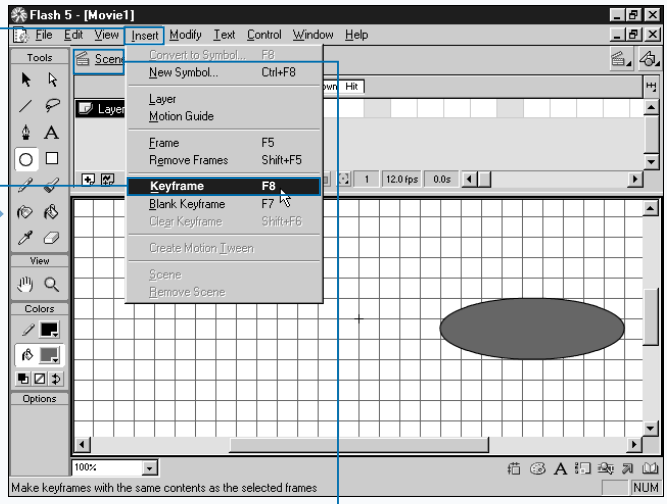
4 Click OK.

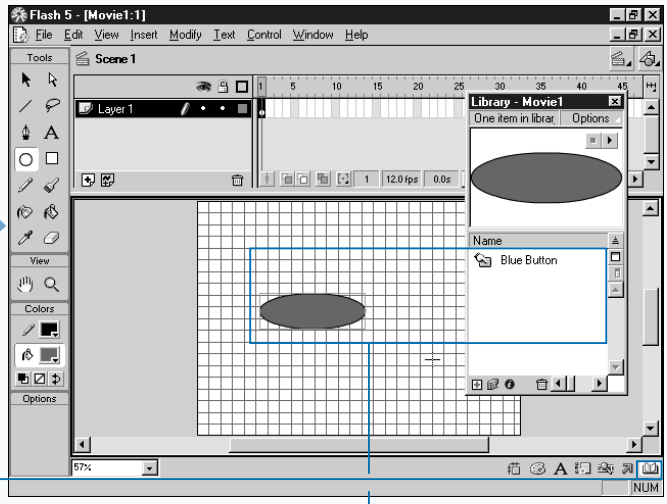
■ Flash moves to the symbol-editing mode.

Extra

Enabling a button lets you test the button. Flash disables buttons by default, making it easier for you to work with them while you create your movie. To enable a button for testing, click Control ➪ Enable Simple Buttons. A check mark appears next to the selection on the menu. When you move your pointer over the button, the Over state appears. When you click while the pointer is over the button, the Down state appears. To select an enabled button, use the Arrow tool to draw a rectangle around the button. To move an enabled button, select it with the Arrow tool and use the arrow keys to move the button around the Stage. To disable a button after enabling it, click Control ➪ Enable Simple Button again. When you are creating a movie, it is best to leave your buttons disabled and only enable your buttons for testing.

You can import the image you use to define a button state. To import an image click File ➪ Import, locate the image you want to import, and click Open.





5 Define the appearance of the Up state by drawing or placing an instance on the Stage.

6 Click Insert ➪ Keyframe.

■ Flash inserts a keyframe and copies the Up state image to the Over state frame. You can change the image if you want.

■ Repeat steps 5 and 6 for the Down and Hit states.

7 Click the Scene button to exit the symbol-editing mode.

■ The main Timeline returns.

8 Click the Show Library icon ( ) to open the Library.

9 Drag an instance of the button from the Library onto the Stage.

■ An instance of the button appears on the Stage.

CREATE ANIMATION

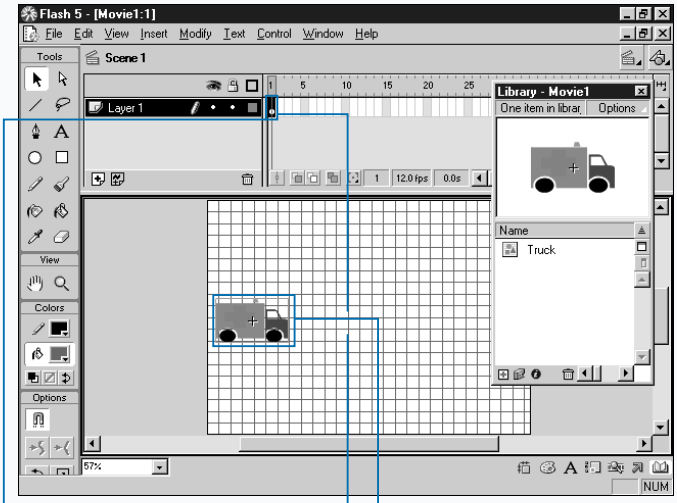
You use Flash to create animation. You can rotate or move objects, fade them in and out, or change their size or shape. You can animate objects independently or have them move and change in concert.

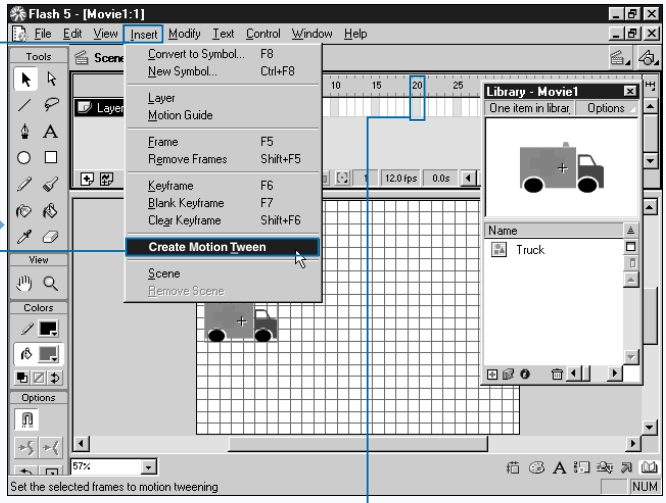
You create animation by changing the content of successive frames. Flash utilizes two types of animation: frame-by-frame and tweened. In frame-by-frame animation, you place an image in each successive frame of the animation. In tweened animation, you place an image in a start frame and in an end frame. Flash then creates the images in the frames in between by varying size, location, or other attributes. Tweened animation utilizes significantly less file space than frame-by-frame animation, because Flash stores only the values for changes in the animation. With frame-by-frame animation, Flash stores the values for each frame.

The two types of tweened animation are motion and shape. With motion tweening, you set the position, rotation, skew, color, or size of an object in a start frame; then you change the position, rotation, skew, color, or size of an object in an end frame. Flash interpolates the frames in between. With shape tweening, you draw a shape in a start frame and then draw another shape in an end frame. Flash again interpolates the frames in between, causing the shape in the start frame to turn into the shape in the end frame.

The start and end frames of a tween must be keyframes. If you change the number of frames between the start and end frames, Flash automatically tweens the frames again. You can tween instances and text.

CREATE ANIMATION





1 Click to select a keyframe.

2 Draw an object on the Stage or drag an instance from the Library.

3 Click to select the Keyframe in which you placed the object.

4 Click Insert ➪ Create Motion Tween.

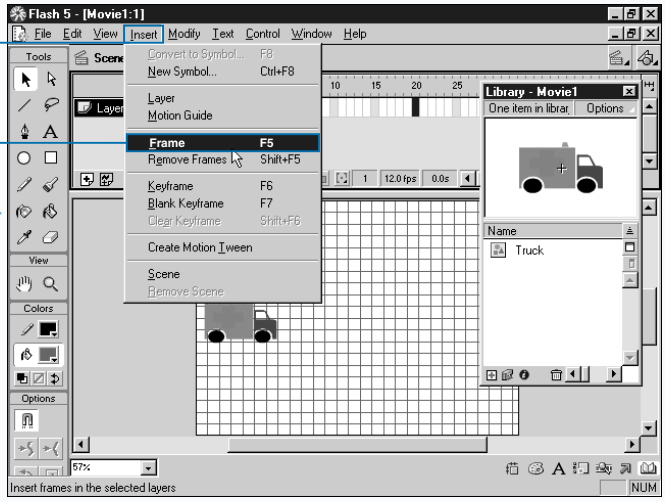
■ If you draw an object on the Stage, Flash automatically converts the object to a symbol and names the object Tween1.

5 Click in the frame in which you want the animation to end.

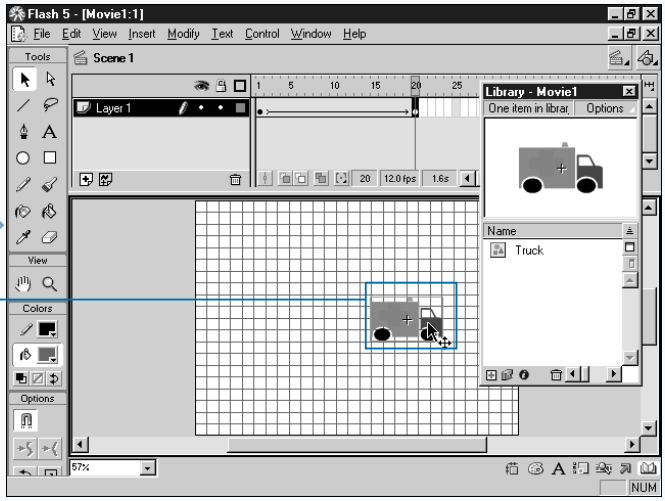
Extra

You can use the Frame panel to adjust a tween. To open the Frame panel click Modify ➪ Frame from the menu.

FIELD	FUNCTION
Label	Give a label name to a frame.
Tweening	Select a tween type of None, Motion, or Shape.
Scale	Select if tweening the size of the object.
Easing	Type -1 to -100 to start the tween slowly and accelerate at the end. Type 0 to 100 to start the tween rapidly and decelerate at the end.
Rotate	Select CW to rotate the object clockwise. Select CCW to rotate the object counter clockwise. Select Auto to rotate object once in the direction requiring the least motion.



6 Click Insert ➪ Frame.



■ A broken line appears on the Timeline. The line indicates that you have begun a tween.

7 Move, adjust the size, change the properties, or rotate the object you want to tween.

■ An arrow line appears. An arrow line indicates that your tween is complete.

ADD SOUND

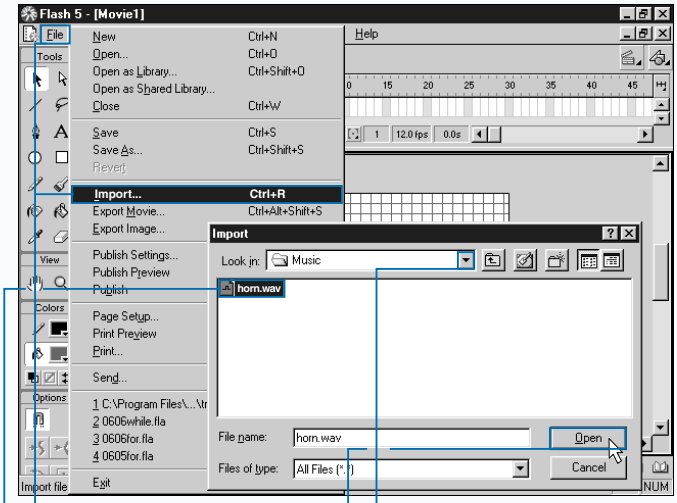
You can add spice to your Flash movie by adding sound. Flash lets you loop, edit, and compress your sound files and create sound effects.

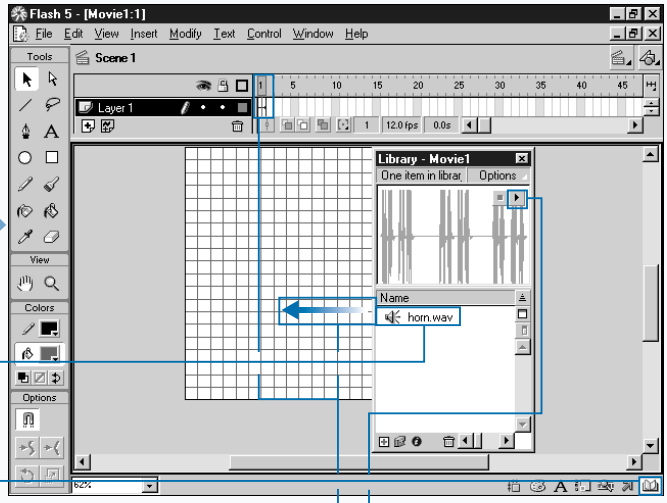
You can import sounds. Imported sounds are stored in the Library. Sounds can be either event sounds or stream sounds. Event sounds are continuous play; they continue to play regardless of what happens in the movie. Event sounds must download completely before they begin. You synchronize Stream sounds with an event. Stream sounds can begin playing before they download completely.

Flash provides compression options that enable you to control the quality and size of sounds in your movie. You set the compression options for individual sounds by double-clicking the Sound icon in the Library to open the Sound Properties dialog box. You define the settings for all sounds in the Publish Settings dialog box.

In the Sound panel, you can choose a sound effect, a synchronization type, and the number of times a sound should loop. There are four synchronization types: Event, Start, Stop, and Stream. When the playhead enters an Event sound's frame, the sound begins and then plays in its entirety. If an Event sound has not completed playing when the movie stops, the sound continues to play. The Start synchronization is the same as the Event synchronization, except if the sound is already playing when the playhead enters the frame, a new instance of the sound begins. The Stop option stops the playing of a sound. Flash synchronizes a Stream sound with an event; a Stream sound cannot play longer than the number of frames it occupies. A Stream sound stops when the movie stops.

ADD SOUND





1 Click File ➪ Import.

■ The Import dialog box appears.

2 Click to select your sound file.

■ You may need to click ▼ to locate your sound file if it does not appear in the Import window.

3 Click Open to open the file.

■ Flash sends the sound file to the Library.

4 Click to open the Library.

5 Click the sound name to select the sound.

6 Click Play (▶) to hear the sound.

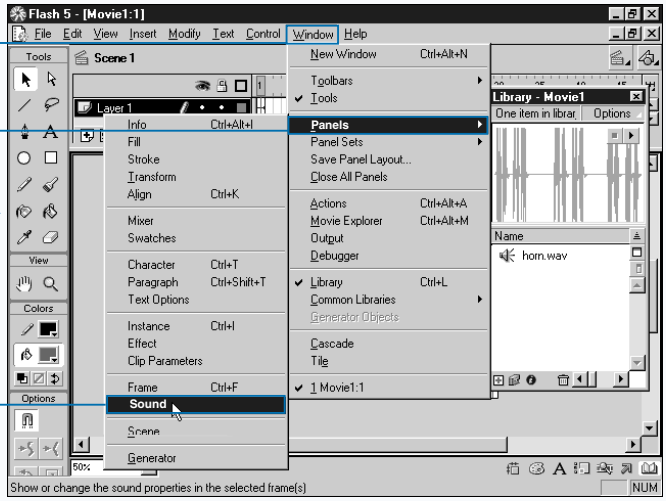
7 Drag the sound onto the Stage.

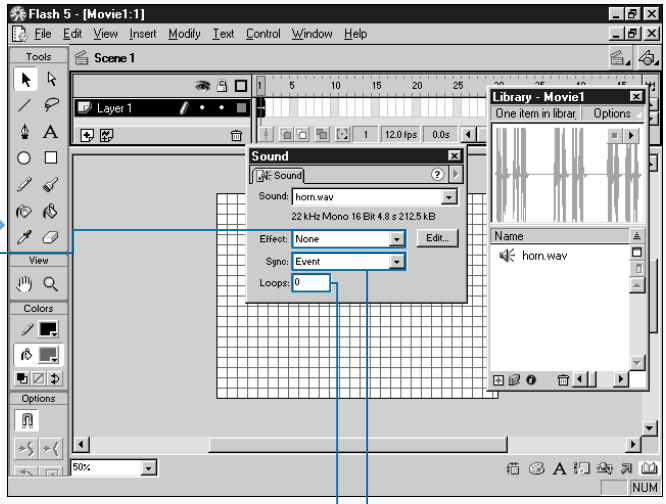
■ The sound appears on the Timeline.

Extra

The Effect field of the sound panel provides the following sound effect options:

EFFECT	DESCRIPTION
None	No sound effect.
Left Channel	The sound plays in the left channel only.
Right Channel	The sound plays in the right channel only.
Fade Left to Right	The sound gradually shifts from the left channel to the right channel.
Fade Right to Left	The sound gradually shifts from the right channel to the left channel.
Fade In	The volume of the sound gradually increases.
Fade Out	The volume of the sound gradually decreases.
Custom	Lets you create channel and fade effects.





8 Click Window ➪ Panels ➪ Sound to open the Sound panel.

■ The Sound panel appears.

9 Click ▾ to select an effect.

10 Click ▾ to select a synchronization option.

11 Type the number of times the sound should loop.

■ The sound is now part of your movie.

CREATE A MOVIE CLIP

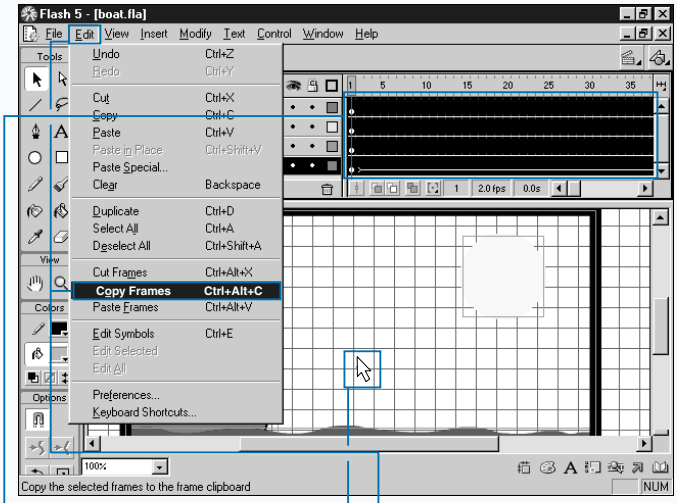
Movie clips enable you to reduce the size of your movie by utilizing multiple instances of the same clip. Movie clips are mini Flash movies with their own Timeline and their own properties. You can turn any Flash animation into a movie clip symbol and then use multiple instances of the movie clip. To distinguish between the multiple instances, you give each instance a name.

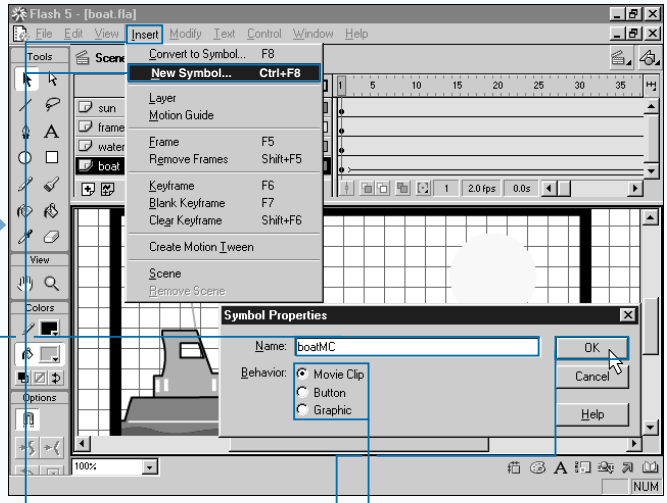
You can embed a movie clip symbol within a Flash movie. Moreover, you can embed multiple instances of a movie clip symbol within each other. You can place a movie clip on the Timeline of the main movie or on the Timeline of another movie clip. Creating a movie clip is easy. First, you create your animation. Then you copy your animation and turn it into a symbol.

To keep your file size down, use movie clips for repetitive actions such as the spinning of a propeller on an airplane. You can use ActionScript to manipulate movie clips to create complex nonlinear interactive movies. See Chapter 7 for more information on working with the Movie Clip object. You can change the position, appearance, and other properties of a movie clip. See Chapter 3 for more information on movie clip properties. Using a movie clip to define the appearance of a button state enables you to create animated buttons.

You can also change your movie clip into a smart clip. A smart clip is a movie clip with ActionScript that designers can reprogram without using ActionScript themselves. With smart clips, designers who do not know ActionScript can easily utilize its power.

CREATE A MOVIE CLIP





1 Create an animation.

Note: See page 14 for information on creating an animation.

2 On the Timeline, select every frame you want to include in the movie clip.

3 Click Edit ➪ Copy Frames to copy the frames you selected.

4 Deselect your selection by clicking anywhere on the Stage.

5 Click Insert ➪ New Symbol to open the Symbol Properties dialog box.

6 Type a symbol name.

7 Click to select Movie Clip as the behavior (○ changes to ◉).

8 Click OK.

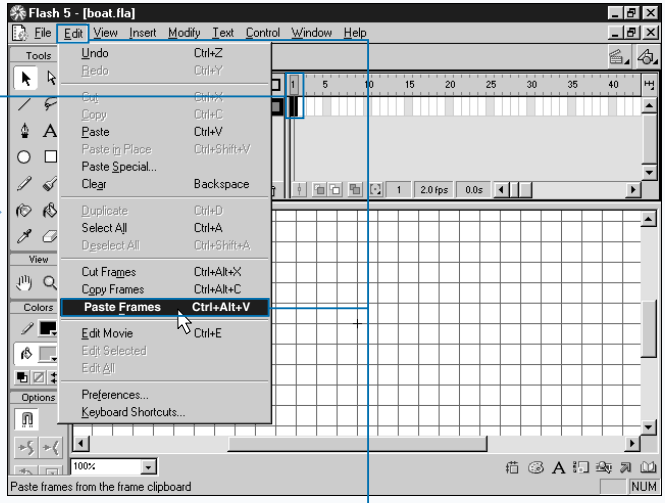
■ Flash changes to the symbol-editing mode.

Extra

Creating an animated button is easy. Start by creating your animation. Convert your animation to a movie clip. Create a button. Use the animation to define the Up, Over, Down, or Hit state.

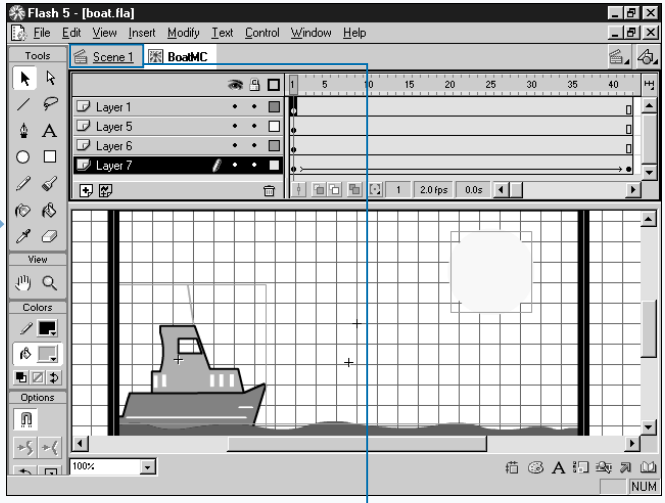
To select everything on every layer, click Edit ➪ Select All on the menu. To deselect everything on every layer, click Edit ➪ Deselect All. To select a layer, click the layer name. To select a frame, click in the frame. To select a block of frames, click in the start frame, hold down the shift key, and then click in the end frame. To add to a selection, hold down the shift key as you make your selection.

To distinguish between instances of a movie clip, you must give each instance a name. To reference a movie clip in ActionScript, in most cases the movie clip must have a name. To name a movie clip instance, use the Arrow tool to select the instance, click Window ➪ Panels ➪ Instance to open the Instance panel, and then type a name in the Name field.



9 Click in Frame 1 of Layer 1 on the Timeline.

10 Click Edit ➪ Paste Frames.



■ Flash pastes the frames you copied into the Timeline of the movie clip symbol.

11 Click the Scene button to exit the symbol-editing mode.

■ Your movie clip appears in the Library.

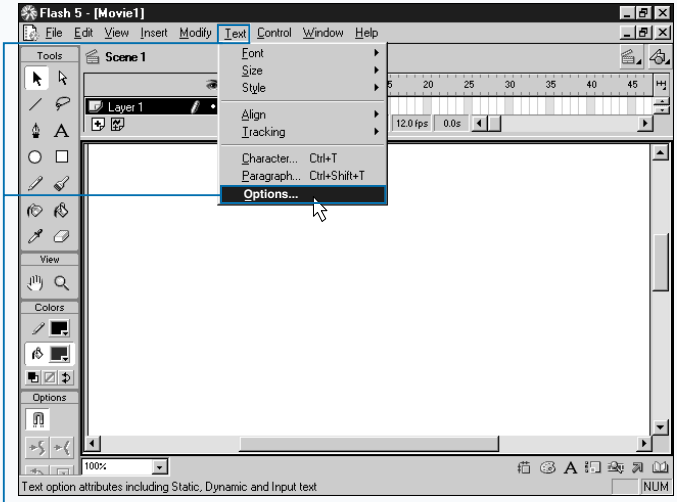
CREATE STATIC TEXT BOXES

Text boxes enable you to include text in your movie, accept user input, and display dynamic text. You use static text boxes to display text that will not change, dynamic text boxes to display text that will change, and input text boxes to request information from the user. Your text box can be single line, fixed width, or editable. Single line text boxes expand as you type. A round handle in the upper-right corner of a text box indicates that the box is a single line text box. Fixed width text boxes wrap words as you type. A square handle in the upper right corner of a text box indicates that the text box is a fixed width text box. Editable text boxes let you accept user input or dynamically display text. A square or round handle in the bottom right corner of the text box indicates that the text box is an editable text box.

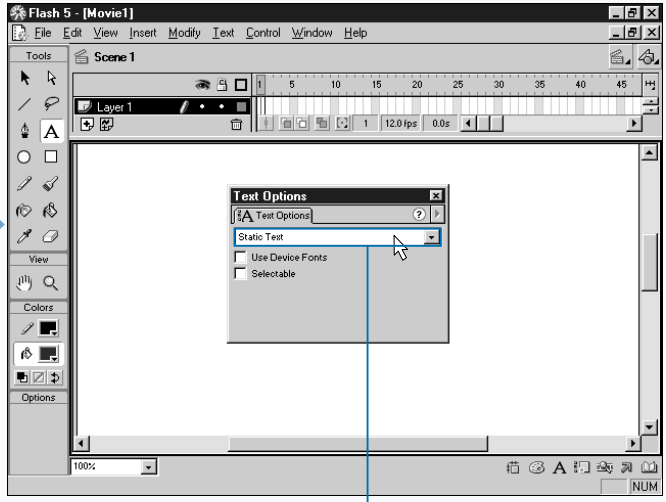
Flash gives you the ability to set the font, size, style, and alignment of your text, using the Text menu options. You can set the color of the font by using the color modifier.

You can use embedded or device fonts. When you use a font that you have installed on your system, Flash embeds the font in the movie. Flash does this to ensure that the font will display properly as the movie plays. As an alternative to an embedded font, you can use a device font. When using device fonts, Flash uses the font on the local computer that most closely matches the device font you chose. Because Flash does not embed device fonts, they reduce your file size. Also, if the point size of your font is less than 10, using a device font can yield a crisper, sharper text. Flash supplies you with three device fonts: _sans, _serif, and _typewriter.

CREATE STATIC TEXT BOXES



1 Click Text ➪ Options to open the Text Options panel.



■ The Text Options panel appears.

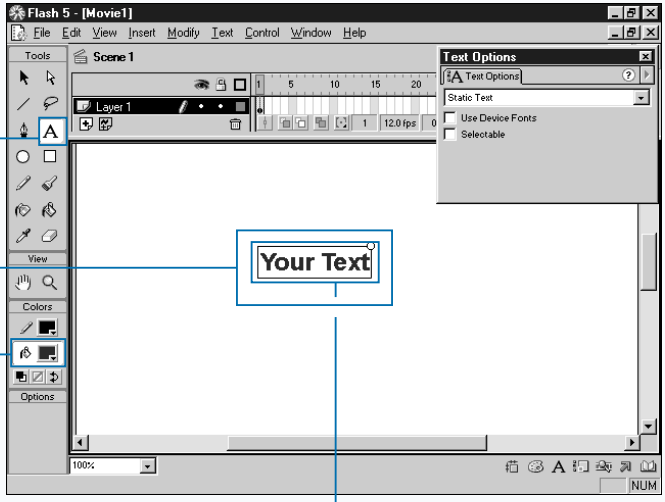
2 Click ▼ to select Static Text from the drop-down list.

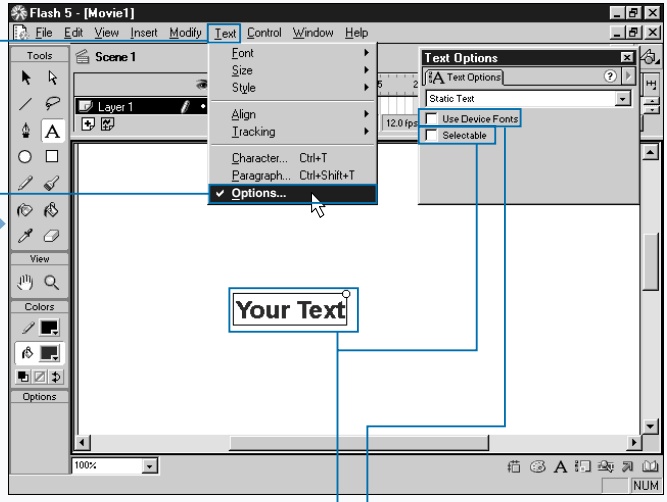
Extra

You can make your text selectable. Making text selectable enables the user to click and drag to highlight text in your movie.

You antialias your text to smooth the edges. To antialias, click View ➪ Antialias on the menu. Antialiasing works best with larger fonts. If you have large amounts of text, antialiasing can slow down the playback of your movie.

You can set the alignment, margins, indents, and spacing of your text boxes. You use alignment to align your text with the left edge, right edge, or center of the text box. You can also use the alignment to justify your text. Use the margin settings to specify the amount of space from the edge of the text box and where the text begins or ends. Use indent to specify where the first line of text for each paragraph will begin. Use spacing to set the amount of space between each line of text. To set the alignment, margins, indents, and spacing, click Text ➪ Paragraph from the menu.





3 Click the Text tool (A) to select it.

4 Click in the color box ( ) to select the color of your font.

5 Click the Stage for a single line text box or click the Stage and drag for a fixed width text box.

Note: See page 23 for information on creating a dynamic text box, and page 22 for information on creating an input text box.

6 Type your text.

7 Click Text ➪ Option to set your text options.

■ The Text Options panel appears.

8 Click Use Device Fonts to use device fonts.

9 Click Selectable to make text selectable.

■ The text appears on the Stage.

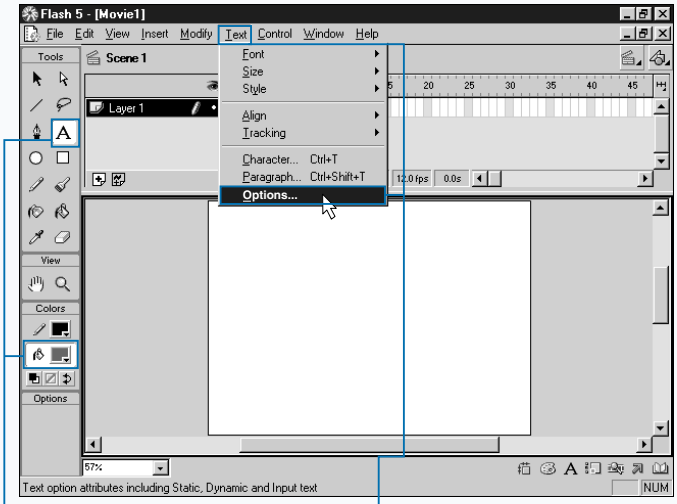
CREATE INPUT TEXT BOXES

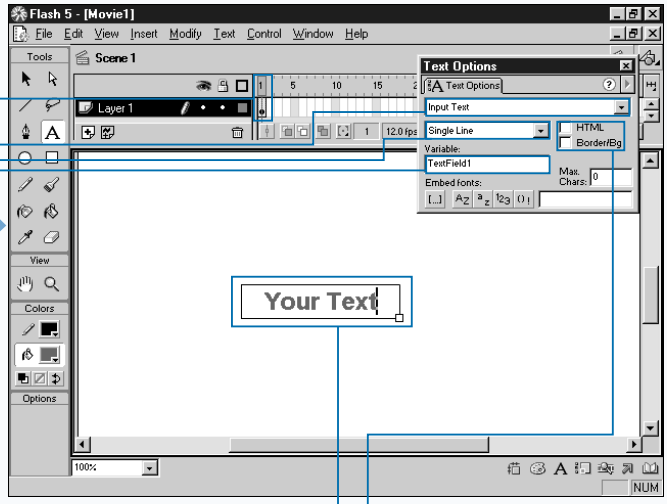
You use input text boxes to request information from the user. You will frequently use input text boxes when creating forms. Using the Text options panel, you associate each input text box with a variable. Variables store information. To learn more about variables, see Chapter 4. Flash assigns the information the user types in an input text box to the variable you specify. You can use ActionScript to read the user input and respond to the user.

When creating an input text box, you can set several options in the Text options panel. In the drop-down list, select from Single Line, Multiline, or Password. Select Single Line to display user input on a single line. If the text is too long to fit in the text box, the text box will scroll. Select Multiline and check the Word Wrap box to display the user input on multiple lines and have the input wrap at the end of the box. If you do not select Word Wrap, Flash will move to a new line when the user presses Enter. Select Password to have asterisks display as the user types. This option enables you to create a form with password protection.

Select HTML to save the formatting with the HTML tags. Select Border/Bg to display a background behind and a border around the text. Type the name you want to give the variable in the Variable field. Type the maximum number of characters the user can enter in the Max. Chars. field. Click one or more Embed Fonts buttons to specify which font characters Flash should embed.

CREATE INPUT TEXT BOXES





1 Click the Text tool (A).

2 Click to select a font color.

3 Click Text ➪ Options to open the Text Options panel.

■ The Text Options panel appears.

4 Click to select Input Text from the drop-down list.

5 Click to select from Single Line, Multiline, or Password.

6 Type a variable name.

7 Click to select HTML, Border/Bg, or Word Wrap.

8 Click the Stage for a single line text box or click the Stage and drag for a fixed width text box.

■ The text box appears on the Stage.

CREATE DYNAMIC TEXT BOXES

You can use dynamic text boxes to display text that updates or to display text in response to user input. Using the Text Options panel, you associate each dynamic text box with a variable. Variables store information. Flash assigns the information in the box to the variable you specify. To learn more about variables, see Chapter 4.

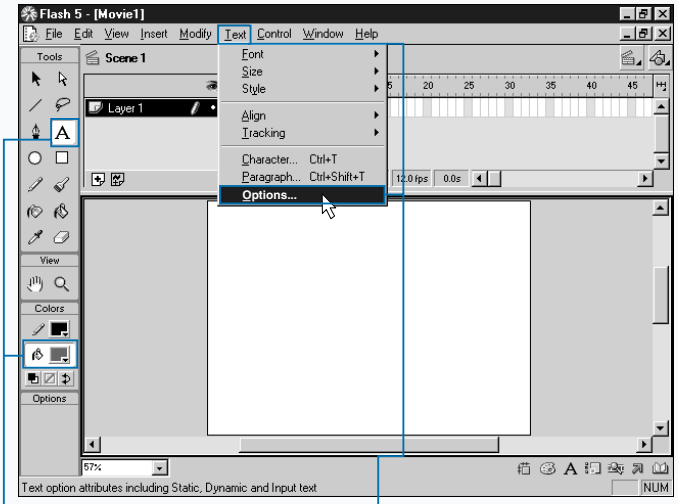
When creating a Dynamic text box, you can set several options by using the Text Options panel. In the drop-down list, you select from Single Line or Multiline. You can select Single Line to display output on a single line. You can select Multiline and check the Word Wrap box to display the text on multiple lines and have the text wrap at the end of the box.

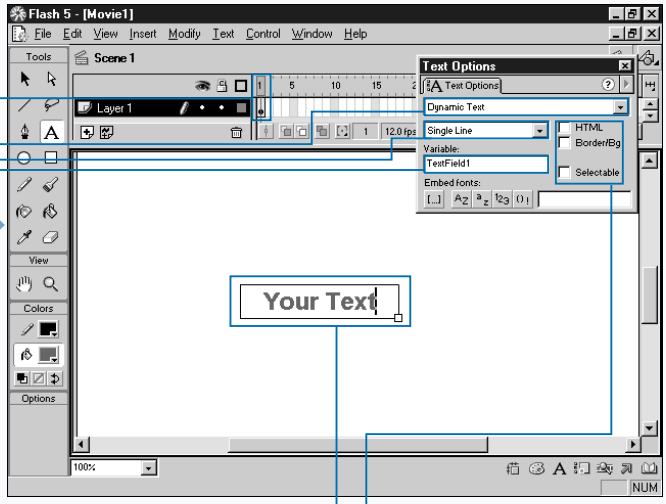
Select HTML to save the formatting with the HTML tags. Select Border/Bg to display a background behind and a border around the text. Type the name you want to give the variable in the Variable field. Click to select Selectable to allow the user to select the text. Click one or more Embed fonts buttons to specify which font characters Flash should embed.

Extra

If you select HTML when you select Dynamic Text or Input Text in the Text Option panel, Flash with create or save simple HTML formatting tags. This allows you to save the font name, style, color, and size associated with the text. The following HTML tags are supported by Flash: `<A>`, `<B>`, `<FONT COLOR>`, `<FONT FACE>`, `<FONT SIZE>`, `<I>`, `<P>`, `<U>`.

CREATE DYNAMIC TEXT BOXES





1 Click A.

2 Click to select a font color.

3 Click Text ➪ Options to open the Text Options panel.

■ The Text Options panel appears.

4 Click to select Dynamic Text from the drop-down list.

5 Click to select Single Line or Multiline.

6 Type a variable name.

7 Click to select HTML, Border/Bg, or Selectable.

8 Click the Stage for a single line text box or click the Stage and drag for a fixed width text box.

■ The text box appears on the Stage.

USING THE ACTIONS PANEL

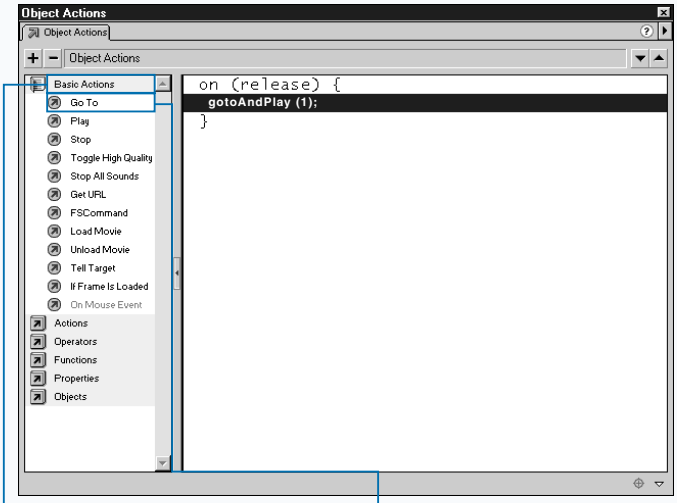
ActionScript is the scripting language that enables you to add interactivity to your Flash movie and to control Flash objects. You can use ActionScript statements to instruct Flash on how to respond to the user mouse clicks and key presses, or you can use ActionScript to have your movie respond to information provided by the user.

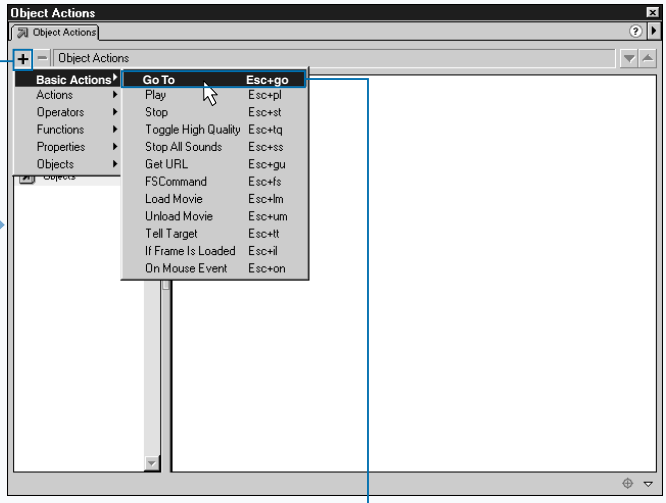
You can create ActionScript in the Action panel. ActionScript is always associated with a frame, button, or movie clip. Using the Arrow tool, you select the frame, button, or movie clip with which you want to associate ActionScript; then you open the Action panel. The Action panel has two modes: Normal and Expert. In Normal Mode, fields in a Parameters panel prompt you for the correct statement arguments. In Expert Mode, you write and edit your statements in a text box. Working in Expert Mode is similar to using a text editor.

In Normal Mode, you can select statements from the Toolbox list on the right side of the Action panel. The Toolbox list divides statements into the following categories: Basic Actions, Actions, Operators, Functions, Properties, and Objects. You can double-click statements to select them. You can also select statements by clicking the Add a Statement button, which categorizes statements the same way the Toolbox list does. You use the Delete a Statement button to delete statements. The Parameters panel prompts you for the arguments related to the statements you select.

After you select a statement, the statement appears in the Actions list on the right side of the Action panel. You can change the order of statements by clicking the Change the Statement Order buttons. The up arrow moves the statements up. The down arrow moves the statements down. You can also move a statement by clicking it and dragging.

USING THE ACTIONS PANEL





1 In the Toolbox list, click the category name to display the statements in the category.

2 Double-click a statement to select it.

3 Click the Add a Statement button (+) to display statement categories.

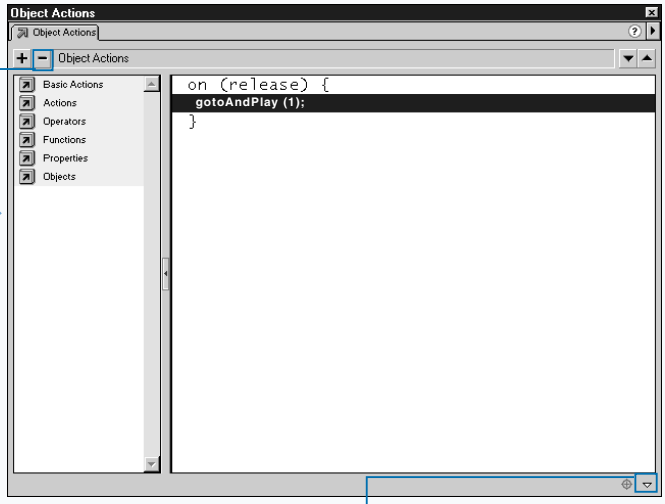
4 Click a statement from the drop-down menu.

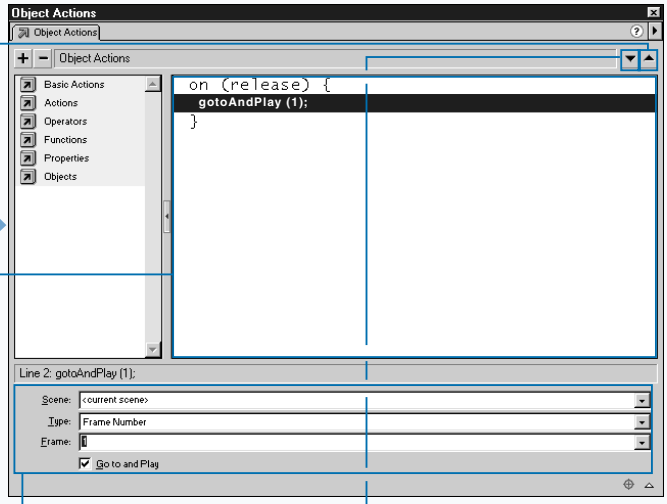
Extra

You can write ActionScript by using a text editor and using the `#include` action to call the text file that contains your ActionScript. The syntax for the `#include` action is

```
#include "filename.as"
```

You can name your file anything you want. Use `AS` for the file extension. The `filename.as` argument represents the relative path to the file. If the movie file and the ActionScript text file are in the same folder, the path is `filename.as`. If the ActionScript text file is in a subfolder, the path to the file is `foldername/filename.as`. When you test, publish, or export your movie, the text file must be present. ActionScript replaces the `#include` statement with the contents of the file.





5 Click the Delete a Statement button (−) to delete a statement.

6 Click ▽ to open the Parameters panel.

■ The Parameters panel appears at the bottom of the panel.

■ The Action list displays the script.

7 Click this Change the Statement Order button (▲) to move a statement up.

8 Click this Change the Statement Order button (▼) to move a statement down.

CONTINUED

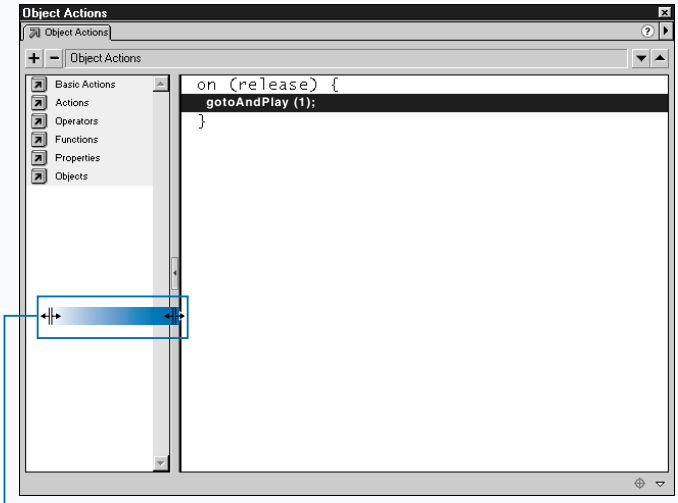
USING THE ACTIONS PANEL (CONTINUED)

You can change the size of the Toolbox list by dragging the vertical splitter bar that appears between the Toolbox list and the Actions list. You can expand or collapse the Toolbox list by clicking the left or right arrow button that appears on the splitter bar.

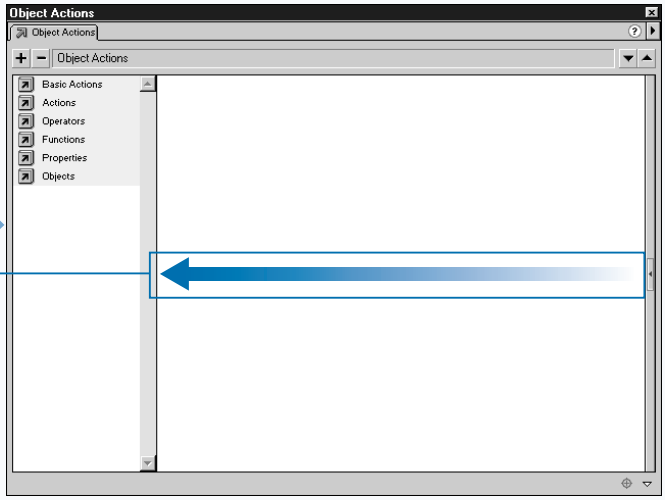
In Expert Mode, you type your statements in the Actions list just as you would if you were using a text editor. You can also add statements by selecting them from the Toolbox list or by using the Add a Statement button. In Expert Mode, there is no Parameters panel to prompt you for arguments, the Change the Statement Order buttons are inactive, and the Delete a Statement button is not available. In addition, Expert Mode does not have a Basic Actions category.

You can switch between Normal and Expert Mode. Changing modes might change the format of your script, so it is best to stick to a single mode per script. You cannot convert Expert Mode scripts with errors to Normal Mode. You can convert Normal Mode scripts with errors to Expert Mode, but you must correct the errors before exporting the script. When you switch from Normal Mode to Expert Mode, Flash preserves the indentation and formatting. When you switch from Expert Mode to Normal Mode, Flash strips any white space you have added and any indentations you have made.

USING THE ACTIONS PANEL (CONTINUED)



9 Drag the splitter bar to change the size of the Toolbox list.



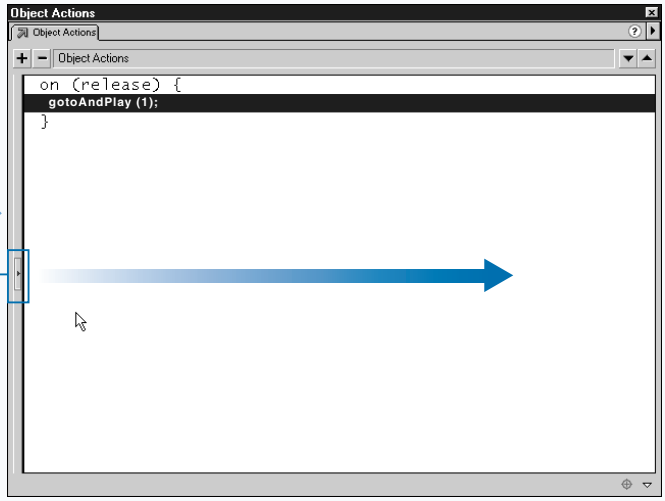
10 Click to collapse the Action list.

Extra

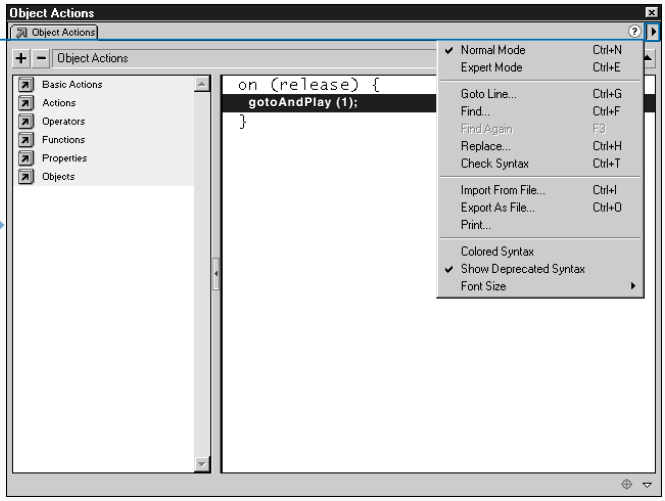
In addition to Normal Mode and Expert Mode, the Options menu in the Action panel offers several other features. Use GoTo Line to move to a specified line in your script. Use Find to find a string of characters. Use Find Again to continue your search. Use Replace to find and replace a string of characters. Use Print to print your script. Choose Font Size from the Options menu to select a small, medium, or large font.

Click Check Syntax to have Flash check your script for syntax errors. Select Colored Syntax to have the various elements of your statements appear in different colors. This helps you debug your code. When you turn on Colored Syntax, keywords and predefined identifiers appear in blue, properties appear in green, comments appear in magenta, and quoted strings appear in gray. A check mark next to Colored Syntax means it is on; no check mark means it is off.

Use Show Deprecated Syntax to have all deprecated syntax display in the Toolbox list with green highlighting. This feature works only if you have the Export Version in the Publish Settings dialog box set to Flash 5. If you set the Export Version to Flash 4 or lower, ActionScript highlights in yellow any syntax not available in the version you selected.



11 Click ▸ to expand the Action list.



12 Click ▸ to change modes or select options.

■ Use the Actions panel to add ActionScript.

ASSIGN ACTIONSCRIPT TO A BUTTON

Assigning actions to buttons enables you to create objects users can drag and objects that respond to user input or perform myriad other tasks in response to mouse clicks or pointer movements. You should assign actions to the instance of the button — not to the Up, Over, Down, or Hit frames on the button Timeline. If you assign an action to a button instance, other instances of the symbol are not affected.

When assigning actions to buttons, you must use a handler to determine the mouse event that will trigger the action. All button handlers begin with the word `on` and end with the event to which the handler responds enclosed in parentheses. In Normal Mode, Flash assigns the `on(release)` handler by default.

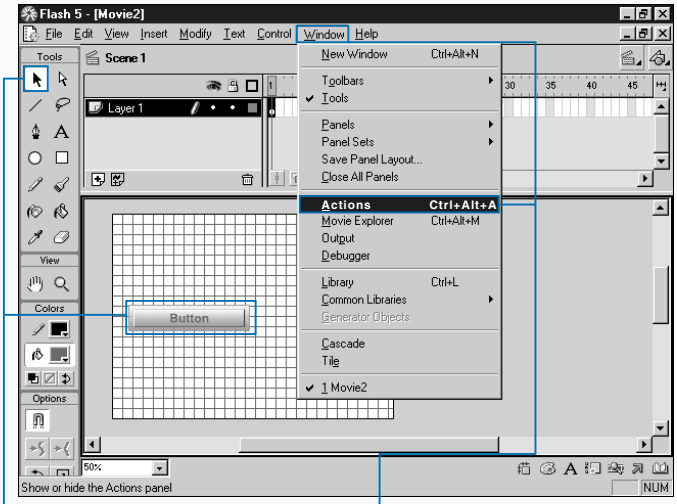
The `on(release)` handler performs specified actions when the pointer is over the button and the user releases the mouse. The `on(press)` handler performs specified actions when the pointer is over the button and the user presses the mouse. Use the `on(releaseOutside)` handler to perform specified actions when the user drags the pointer outside the button area and then releases the mouse.

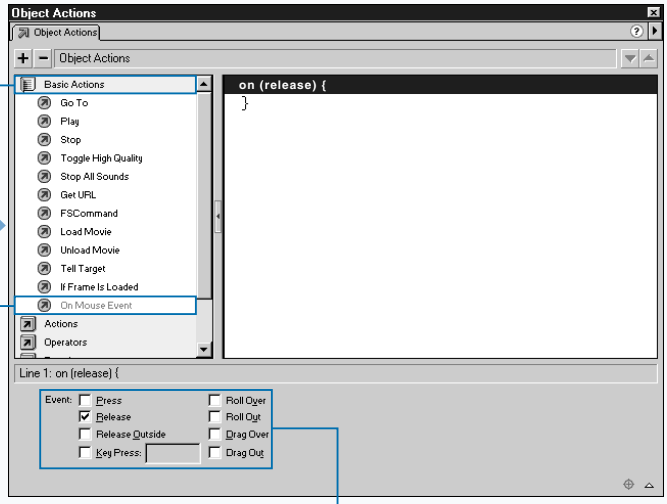
You can use the `on(rollOver)` handler to perform specified actions when the user rolls the pointer over the button. You can use the `on(rollOut)` handler to perform specified actions when the user rolls the pointer over and then outside of the button area.

Use the `on(dragOver)` handler to perform specified actions when the user clicks the button, drags the pointer away from the button, and then drags the pointer back over the button. Use the `on(dragOut)` handler to perform specified actions when the pointer is over the button and the user presses the mouse and then drags the pointer outside the button area.

The `on(keyPress, "key")` handler performs specified actions when the user presses a specified key.

ASSIGN ACTIONSCRIPT TO A BUTTON





1 Use the Arrow tool ( ) to select a button.

2 Click Window ➪ Actions to open the Object Actions panel.

■ The Object Actions panel appears.

3 Click Basic Actions.

4 Double-click On Mouse Event to select a handler.

■ ActionScripts defaults to `on(release)`.

5 Click a check box to select a handler.

■ Use handlers to specify when Flash should execute ActionScript.

ASSIGN ACTIONSCRIPT TO A MOVIE CLIP

Assigning ActionScript to a movie clip enables you to have your movie perform a variety of tasks when the user clicks or rolls the pointer over the movie clip. If you assign an action to a movie clip instance, other instances of the symbol are not affected.

When assigning actions to movie clips, you must use a handler to determine the mouse event that will trigger the action. All movie clip handlers begin with the word `onClipEvent` and end with the event to which the handler responds enclosed in parentheses. In Normal Mode, Flash assigns the `onClipEvent(load)` handler by default if you select OnClipEvent from the Action list or if you select an action without specifying a handler.

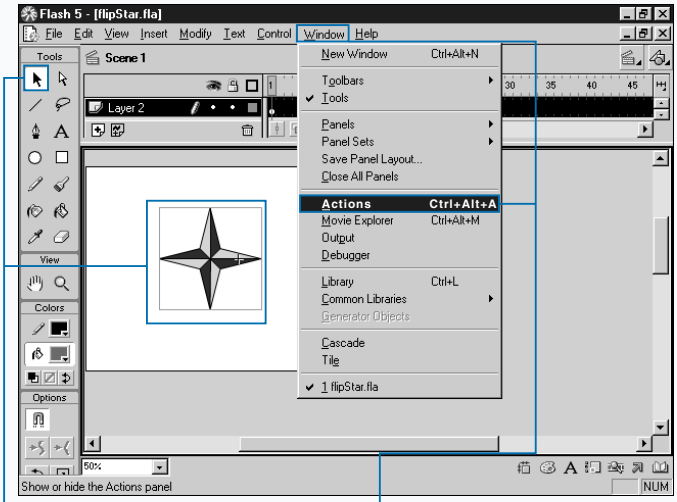
The `onClipEvent(load)` handler executes specified actions the first time the movie clip appears on the Timeline. The `onClipEvent(unload)` handler executes specified actions in the first frame after you remove the movie clip from the Timeline.

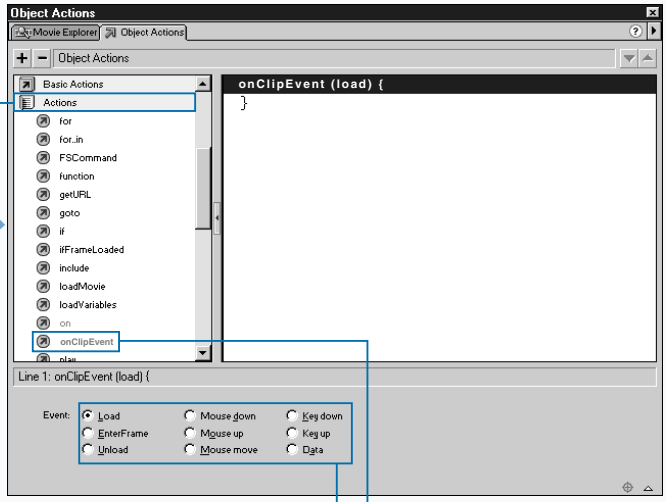
Use the `onClipEvent(enterFrame)` handler to perform specified actions each time the playhead enters the frame. Use the `onClipEvent(mouseMove)` handler to perform specified actions every time the user moves the mouse. Use the `onClipEvent(mouseDown)` handler to perform specified actions when the user presses the mouse. Use the `onClipEvent(mouseUp)` handler to perform the specified actions when the user releases the mouse.

The `onClipEvent(keyDown)` handler performs the specified actions when the user presses any key. The `onClipEvent(keyUp)` handler performs the specified actions when the user releases any key. The `onClipEvent(data)` handler performs the specified actions when ActionScript receives data.

You assign ActionScript to a movie clip by selecting the movie clip and opening the Object Actions panel.

ASSIGN ACTIONSCRIPT TO A MOVIE CLIP





1 Use ↖ to select the movie clip.

2 Click Window ➪ Actions to open the Object Actions panel.

■ The Object Actions panel appears.

3 Click Actions.

4 Double-click onClipEvent to select the `onClipEvent` action.

5 Click to select a handler (○ changes to ◉).

■ Use handlers to specify when Flash should execute ActionScript.

ASSIGN ACTIONSCRIPT TO A FRAME

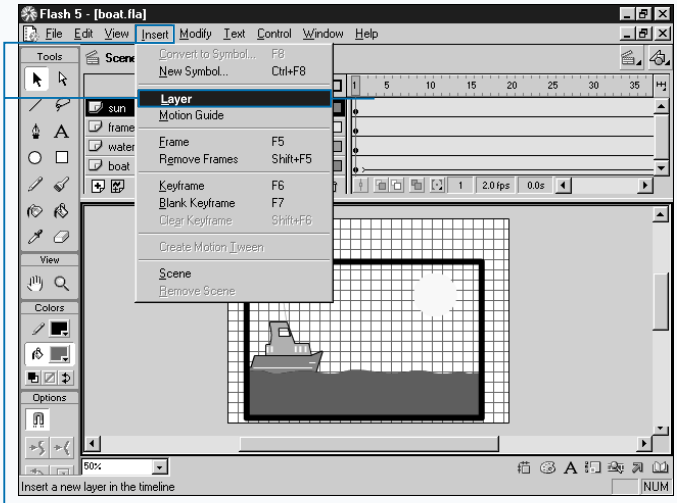
You can associate ActionScript with a frame. The actions execute when the playhead enters the frame. This is useful when you want to use ActionScript to initialize your variables, create a loop within a movie, or perform other tasks on entry into a frame.

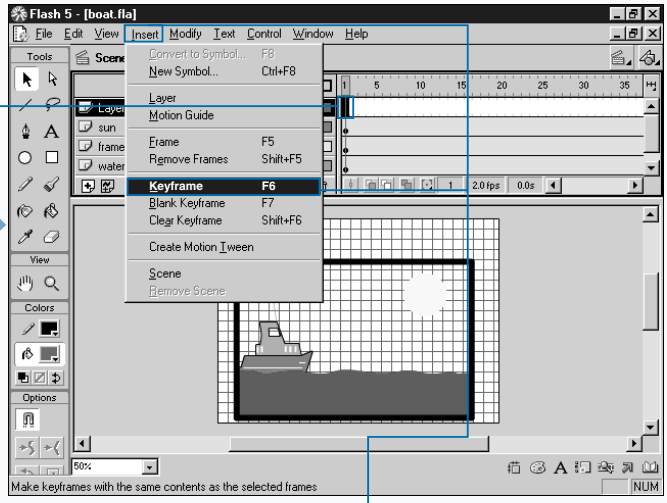
You must assign ActionScript to a keyframe. Keyframes mark changes in the action and store ActionScript. By default, the first frame in the Timeline is a keyframe. You create additional keyframes by selecting a frame and clicking Insert ➪ Keyframe from the menu. If you try to assign an action to a frame that is not a keyframe, Flash automatically assigns the ActionScript to the previous keyframe on the Timeline.

You should place frame actions on their own layer. Frames with actions in them display with a small *a*. For more on layers, see page 4.

You can assign actions to a frame by clicking in a frame to select it and then opening the Action panel. You use the Action panel to enter your ActionScript. If you select multiple keyframes, Flash dims the Action panel and you will not be able to enter your ActionScript.

ASSIGN ACTIONSCRIPT TO A FRAME





1 Click Insert ➪ Layer to create a new layer.

■ A new layer appears above the active layers.

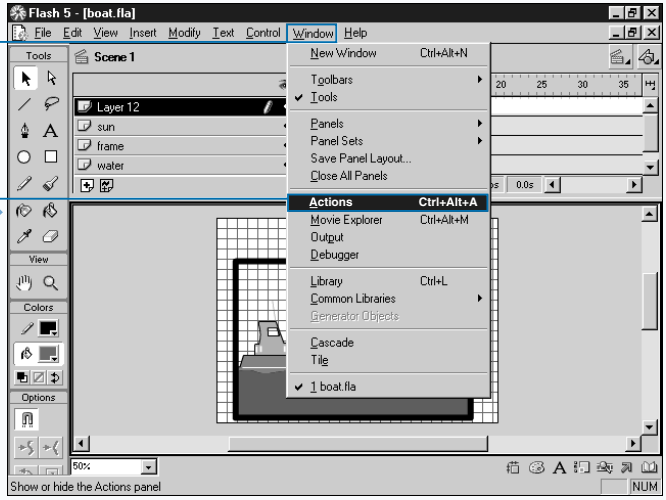
2 Click to select a frame.

3 Click Insert ➪ Keyframe to create a keyframe.

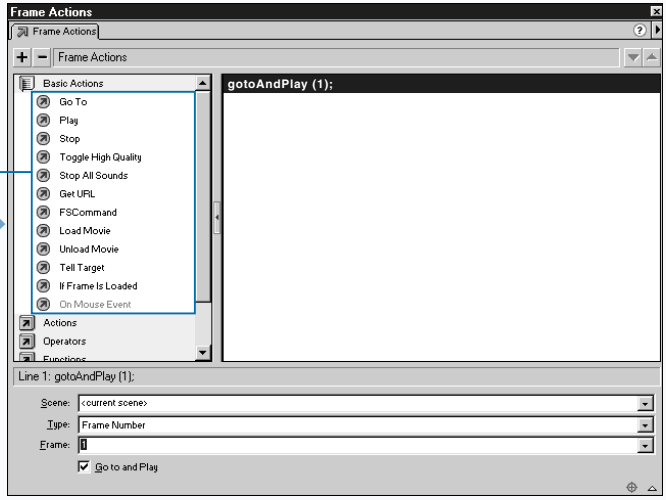
Extra

Flash gives consecutive numbers to the frames in the Timeline. When you are referring to frames in ActionScript, labeling the frames is preferable to referring to a frame by a frame number. If you add frames to or remove frames from your movie, the frame numbers change, but a label associated with a frame remains the same. Put frame labels on their own layer. For more on layers, see page 4.

To create a frame label, select the frame to which you want to assign a label, click Window ➪ Panels ➪ Frame to open the Frame panel, and then type a label name in the Label field.



4 Click Window ➪ Actions to open the Frame Actions panel.



5 Click to select your actions.

■ Use the Frame Actions panel to enter your ActionScript.

PUBLISH MOVIES

Publishing your movie enables you to present it to your audience. You can present your movie in several formats including a Flash movie for the Web; a GIF, JPEG, PNG, or QuickTime animation; an HTML document; or as a stand-alone executable for Windows or Macintosh.

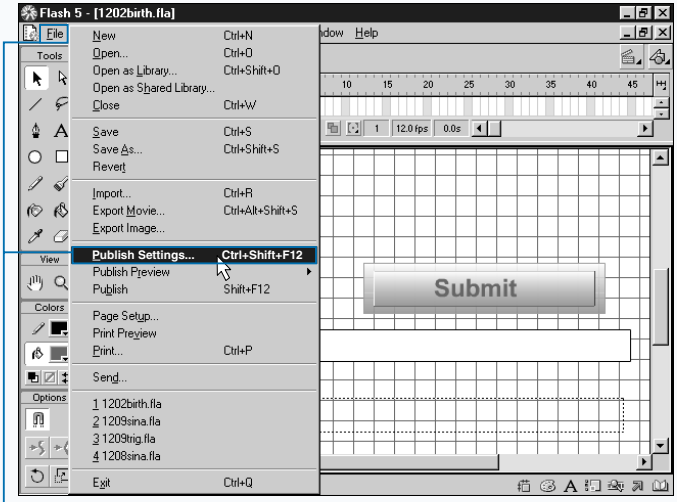
You use the Publish Settings dialog box to publish your movie. You select the formats. Flash assigns a default file name. You can use the default name or you can assign a filename.

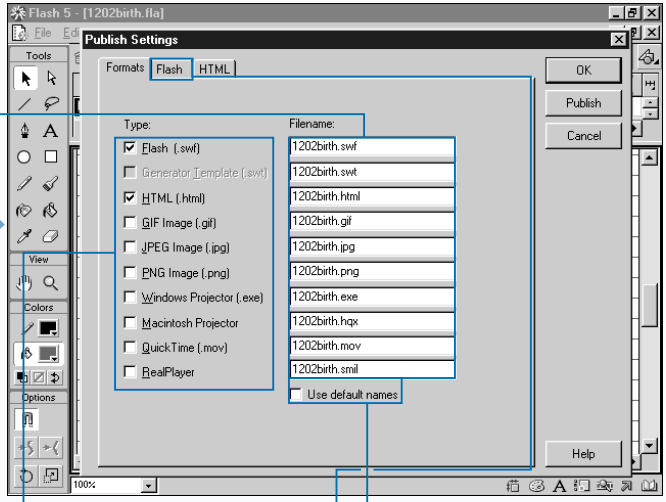
When publishing your movie for the Web, you can set several options including the order in which the layers load. The Load Order option controls which layer Flash Player draws first when the user plays the movie over a slow modem or network connection. Additionally, you can select the following options: Generate Size Report, Omit Trace Actions, Protect from Import, and Debugging Permitted.

Selecting Generate Size Report creates a TXT file containing detailed information on the size of each frame, scene, and object in your movie. Omit Trace Actions causes Flash to ignore any `trace` actions included your movie. For more information on `trace` actions, see Chapter 13. When you save your movie, Flash assigns the movie an FLA extension. You can modify an FLA file. When you publish your movie as a Flash movie for the Web, Flash assigns the movie an SWF extension. Protect from Import prevents others from importing your SWF file and converting it to an FLA file. Debugging Permitted allows the activation of Debugger. If you select this option, you can require a password for its use. See Chapter 13 for more information on debugging.

A JPEG Quality slider is also available and enables you to adjust the quality of JPEG graphics — the higher the quality the larger the file size.

PUBLISH MOVIES





1 Click File ➪ Publish Settings to open the Publish Settings dialog box.

■ The Publish Settings dialog box opens.

2 Click in the Type Boxes to select file types (☐ changes to ☑).

3 Type in the Filename field to change the filename.

4 Click Use Default Names to use the default filenames.

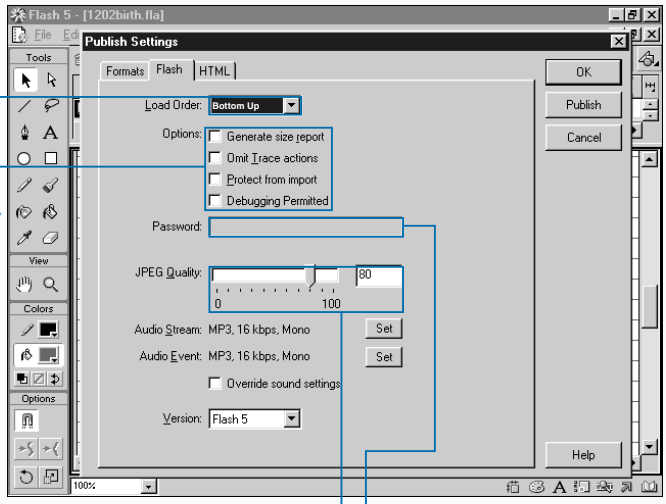
5 Click the Flash tab to set the options for the SWF file.

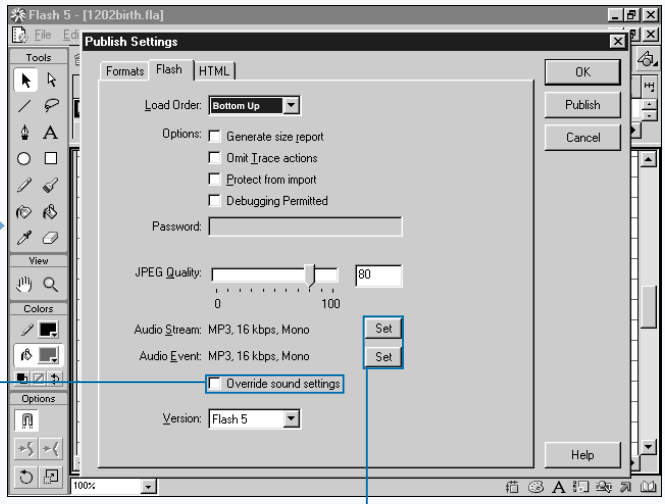
Extra

To have your Flash movies load quickly, you want to keep your file size as small as possible. Sounds can increase file size significantly. You can adjust the sample rate and compression of a sound to reduce the file size. The lower you set the sample rate and compression, the lower the quality of sound, and the smaller the size of your file. You may want to experiment to obtain a suitable tradeoff between sound quality and file size.

When creating an HTML document to display a Flash movie, you use the `object` and `embed` tags. Internet Explorer uses the `object` tag on Windows, while the `embed` tag is used by Netscape Navigator on Windows and the Macintosh and by Internet Explorer on the Macintosh. Internet Explorer for Windows uses ActiveX to play Flash content, while all other browsers use the Flash plugin.

If you change the width and height of your movie, you can use the Scale option to place the movie in the Web browser. Selecting default will display your movie within the boundaries you specify, maintaining the original aspect ratio. Selecting No Border will display your movie within the boundaries you specify; however, Flash will crop the movie, if necessary. Selecting Exact Fit will place the movie within the boundaries you specify, but will not maintain the original aspect ratio. Selecting this option may cause distortion.





■ The Flash option screen appears.

6 Click ▾ to set the Load Order.

7 Click to select the Generate Size Report, Omit Trace Actions, Protect from Import, or Debugging Permitted options.

8 If you selected Debugging Permitted, enter a password if you want.

9 Set the JPEG Quality using the JPEG Quality slider.

10 Click to override current sound setting.

11 Click Set to open the Sound Setting dialog box for Stream or Event sounds.

CONTINUED

PUBLISH MOVIES (CONTINUED)

If you did not specify the sample rate and compression for sound files using the Sound Properties dialog box, you can set them in the Publish Settings dialog box or you can use the Publish Settings dialog box to override previous settings. Use Audio Stream to set the sample rate and compression for stream sounds. Use Audio Event to set the sample rate and compression for event sounds.

You can select the version of Flash for which you want to publish your movie. Actions available in Flash 5 may not be available in earlier versions.

To play your movie in a Web browser, the movie must be part of an HTML document. Selecting HTML as the format in the Publish Settings dialog box creates an HTML document that includes your Flash movie.

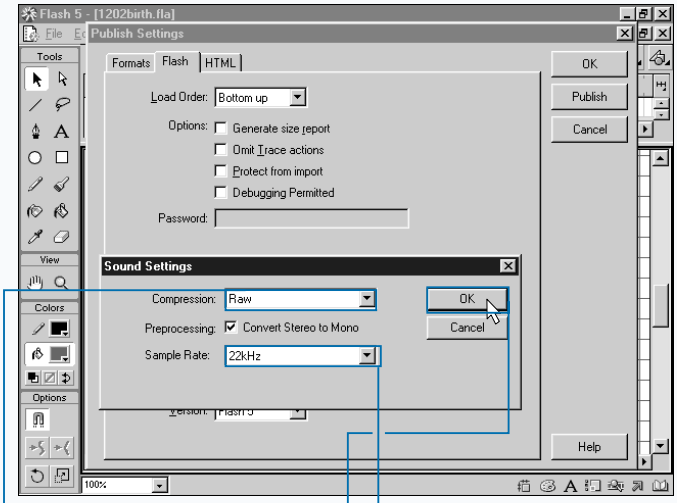
Flash provides you with several templates you can use to create your HTML document. Pressing the Info button, in the Publish Settings dialog box, provides you with a description of each of the templates.

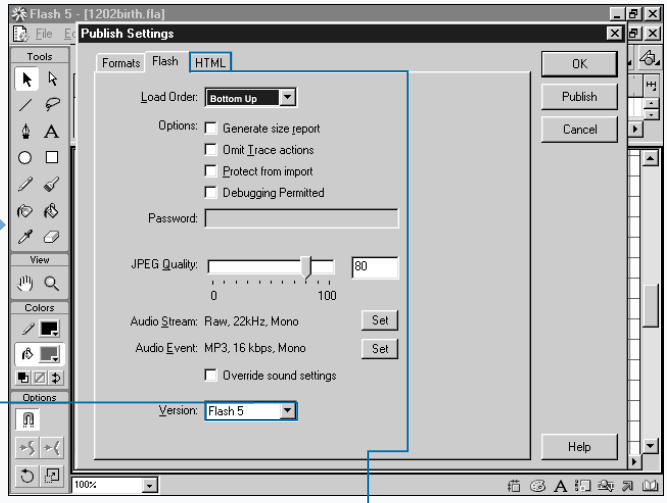
In your HTML document, you can have the size of the movie match the size of the movie you created, or you can specify the size in pixels or as a percentage of the original movie.

The Publish Settings dialog box contains four playback options. You can pause the movie until the user presses a button or selects play from the shortcut menu. You can have the movie loop when it reaches the last frame. You can display a shortcut menu when the user right-clicks in Windows or control-clicks on a Macintosh. And, in Windows, you can use device fonts if the font you selected is not available on the system of the user.

The Publish Settings dialog box also has options that enable you to set the quality, window mode, alignment, and scale.

PUBLISH MOVIES (CONTINUED)





■ The Sound Settings dialog box appears.

12 Click ▼ to select a Compression type.

13 Click ▼ button to select a Sample Rate.

14 Click OK to close the dialog box.

15 Click ▼ to select the version of Flash for which you are authoring.

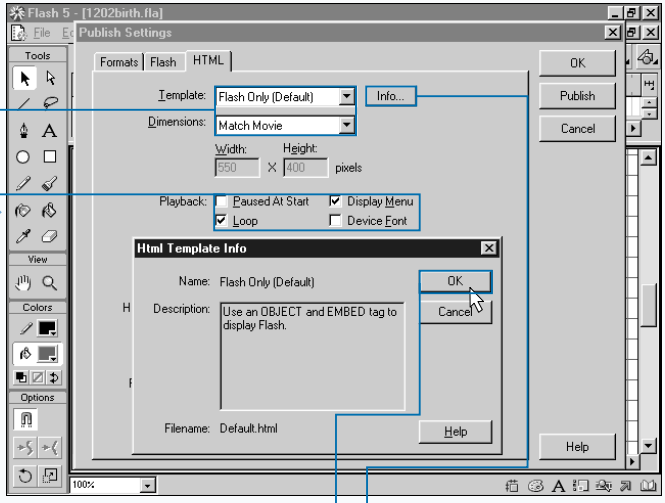
16 Click the HTML tab to display the HTML options.

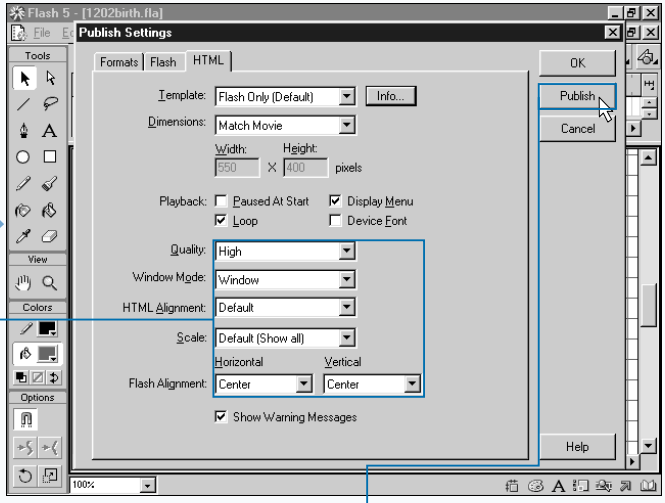
Apply It

You can use the code shown here to manually add a Flash Movie to an HTML document.

```
<OBJECT CLASSID="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" WIDTH="100"
    HEIGHT="100" CODEBASE="http://active.macromedia.com/flash5/cabs/
    swflash.cab#version=5,0,0,0">
    <PARAM NAME="MOVIE" VALUE="yourmovie.swf">
    <PARAM NAME="LOOP" VALUE="false">
    <PARAM NAME="QUALITY" VALUE="high">
    <PARAM NAME="PLAY" VALUE="true">

    <EMBED SRC="yourmovie.swf" WIDTH="100" HEIGHT="100" PLAY="true"
        LOOP="false" QUALITY="high"
        PLUGINSPAGE="http://www.macromedia.com/shockwave/
download/index.cgi?P1_Prod_Version=ShockwaveFlash">
    </EMBED>
</OBJECT>
```





■ The HTML options appear.

17 Click ▼ to set the Template and Dimensions options.

18 Click to select the Playback options (☐ changes to ☑).

19 Click Info for a description of the template.

20 Click OK to close the HTML Template Info dialog box.

■ The dialog box closes.

21 Click ▼ to select the Quality, Window Mode, HTML Alignment, Scale, or Flash Alignment options.

22 Click Publish to publish your movie.

■ Flash creates an HTML document that you can use to publish your movie on the Web.

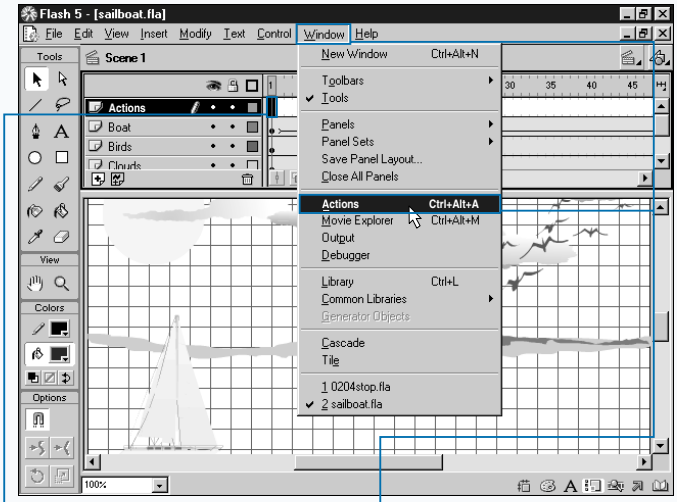
INTRODUCTION TO ACTIONSCRIPT SYNTAX

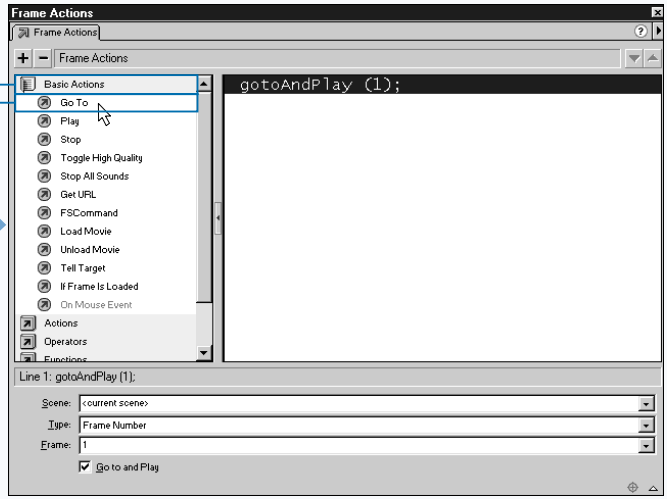
ActionScript is the language used to communicate with Flash. ActionScript can consist of a single statement or a series of statements. Statements provide Flash with instructions. Statements execute from top to bottom unless you issue a statement telling Flash to execute in another order. You can use the `if` action and loops to change the normal top to bottom order of execution.

Like spoken languages, ActionScript has rules of punctuation and grammar. These rules comprise the syntax of the language. You must follow these rules for your script to execute properly. For starters, each ActionScript statement ends in a semi-colon. However, if you omit the semicolon, the script will still compile successfully. You group ActionScript into blocks of code. Enclose each block of code in curly braces. Additionally, in ActionScript, only keywords are case sensitive. For a complete list of keywords, see Chapter 14.

An *argument* — also referred to as a parameter — is a value associated with an action or function. These values clarify or provide additional instructions to Flash. For example, you use the `gotoAndPlay` action to tell flash to start playing a movie. The `gotoAndPlay` action takes two arguments: `scene` and `frame`. You use the `scene` argument to tell Flash the scene you want to play. You use the `frame` argument to tell Flash the frame in which you want the movie or movie clip to begin playing. You separate the arguments associated with an action or function with commas and enclose them in parentheses. The values you assign to an argument can be either a literal value or an expression. Enclose literal values in quotes. Do not enclose expressions in quotes. For example, `gotoAndPlay("Scene 2", 5);` tells Flash to go to Scene 2 and start playing the movie in Frame 5.

INTRODUCTION TO ACTIONSCRIPT SYNTAX





1 Select the frame, button, or movie clip to which you want to add ActionScript.

■ This example uses a frame.

2 Click Window ➪ Actions to open the Actions panel.

Note: If you associate an action with a button or movie clip, the Object Actions panel will open. If you associate an action with a frame, the Frame Actions panel will open.

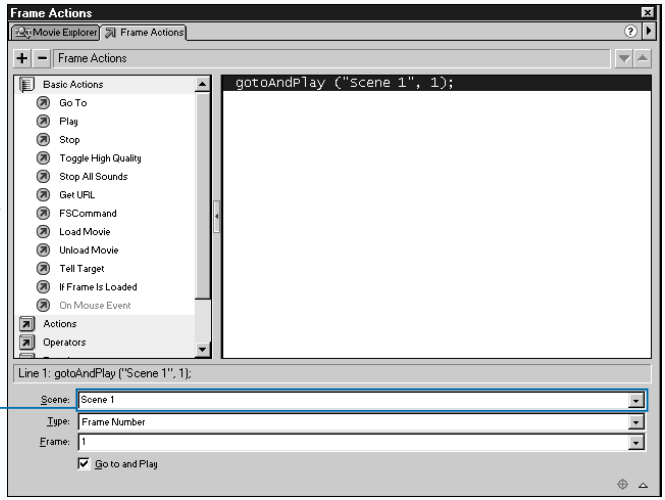
3 Click Basic Actions.

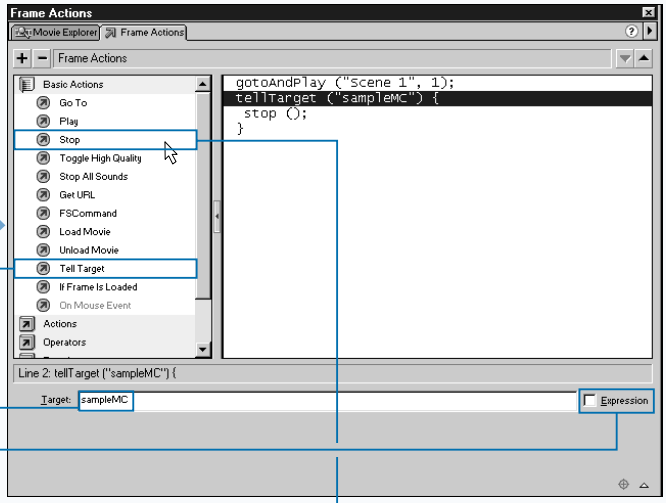
4 Double-click Go To to select the `gotoAndPlay` action.

■ The parameters panel opens.

Extra

An expression is any statement that Flash evaluates and returns a value. For example, with Flash you can store information in variables. You can assign x = 2. X is the variable and 2 is the value. Every time Flash sees the variable x, it evaluates it and returns 2. X is an expression. Mathmatical equations are also expressions. The equation 2 + 3 is an expression. You can use variables and properties to form an expression. For example, x + 2 is an expression consisting of the variable x and the value 2. The property `_currentframe` returns frame number in which the playhead is currently located. Both `_currentframe` and `_currentframe + 1` are valid expressions. You can use expressions when Flash asks for an argument. When using the Action Panel in Normal Mode, if you are entering an expression, select the expression check box. If you are entering a literal value, do not check the expression check box.





5 Click ▾ to select Scene 1.

■ Quotes enclose the literal value.

■ A semicolon ends the statement.

■ Parentheses enclose the arguments and a comma separates the arguments.

6 Double-click Tell Target.

7 Type **sampleMC** in the Target field.

8 Select Expression if your entry is an expression.

■ When you select Expression, Flash removes the quotes.

9 Double-click Stop.

■ Curly braces surround the block of code.

TEST A MOVIE

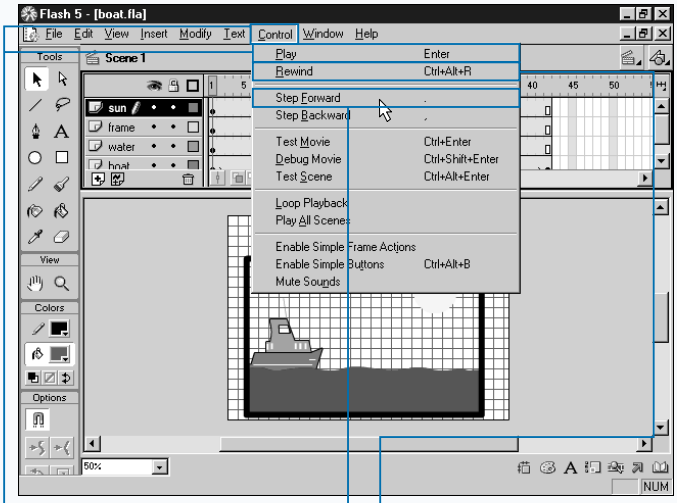
As you create your Flash movie, you will want to test it to ensure that animations and scripts work properly. You can test your movie in the authoring environment, in the test environment, or in a Web browser.

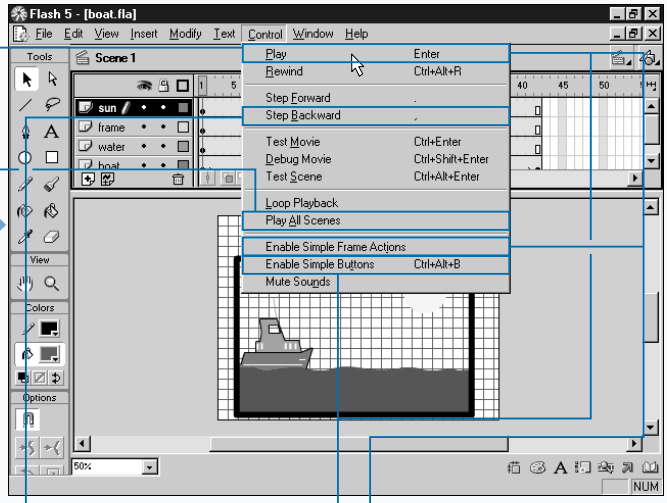
To test your movie in the authoring environment, use the commands on the Control menu. The Play option plays your movie. The Rewind option returns your movie to the first frame. The Step Forward option moves your movie forward one frame. The Step Backward option moves your movie backward one frame. Loop Playback causes your movie to play continuously. The Play All Scenes option causes all scenes to play. The Mute Sounds option causes your movie to play without sound. By default, Flash disables buttons and actions, enabling you to manipulate buttons, movie clips, and frames as you work. Select Enable Simple Buttons and Enable Simple Frame Actions to test buttons and actions in the authoring environment.

When you click File ➪ Save, Flash saves the movie you are authoring in FLA format and appends an FLA extension to the filename. To view your animation on the Web, your file must be in its final format — the SWF format.

Not all animations and scripts work in the authoring environment. You must export your movie to its final SWF format for some animations and scripts to work. Clicking Control ➪ Test Movie from the menu exports your movie to its final format, creating an SWF file in accordance with the setting you choose in the Publish Settings dialog box, and moves you to the test environment. Clicking Control➪Test Scene from the menu also creates a SWF file. Control, Test Scene enables you to test the current scene.

TEST A MOVIE





1 Click Control to display the Control menu.

2 Click Play to play your movie.

3 Click Rewind to return to the first frame.

4 Click Step Forward to move forward one frame.

5 Click Step Backward to move backward one frame.

6 Click Play All Scenes and click Play to play all scenes.

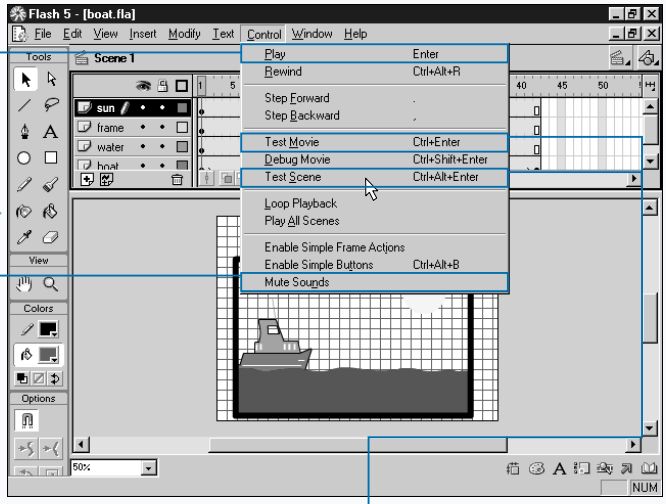
7 Click Enable Simple Frame Actions and click Play to test actions.

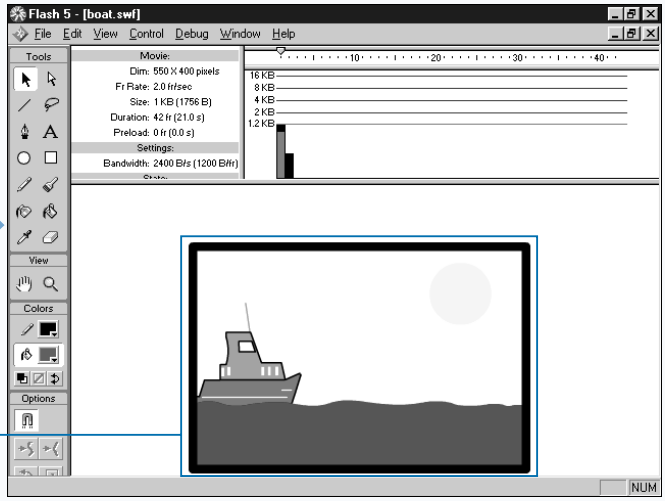
8 Click Enable Simple Buttons to test buttons.

Extra

You can play your movie using the Controller Toolbar. Click Window ➪ Toolbar ➪ Controller in Windows or click Window ➪ Controller on the Macintosh to open the Controller bar. The Stop button will stop your movie. The Rewind button will return your movie to frame 1. Step Back steps your movie back one frame. Play plays your movie. Step Forward steps you forward one frame. Go To End takes you to the last frame.

To test your movie in a Web browser, click File ➪ Publish Preview ➪ Default from the menu. Flash opens your movie in your default Web browser.





9 Click Mute Sounds or Play to have your movie play without sound.

10 Click Test Movie or Test Scene to move to the test environment.

■ This is the test environment.

■ In the test environment, you can test your movie in its final form.

ENTER THE TEST ENVIRONMENT

Any data in a frame not downloaded when your movie reaches the frame will cause your movie to pause until the data downloads. In the test environment, you can view a graphical representation of movie performance at various modem speeds. This enables you to see which frames may cause your movie to pause.

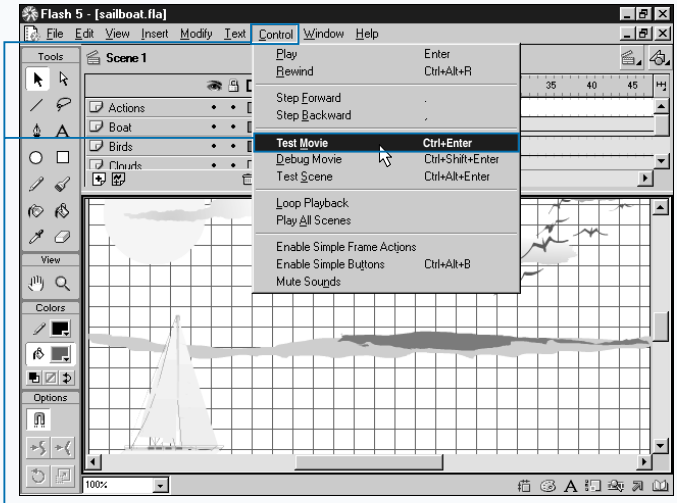
You select the modem speed you want to simulate. Typically, the stated modem speed is higher than the typical speed a user experiences. The Bandwidth Profiler estimates typical Internet speed and calculates performance based on that speed. For example, Bandwidth profiler uses 2.3KB/s for a 28.8 modem. The Bandwidth Profiler also has an option, which enables you to specify the modem speed and set the typical performance you want to test.

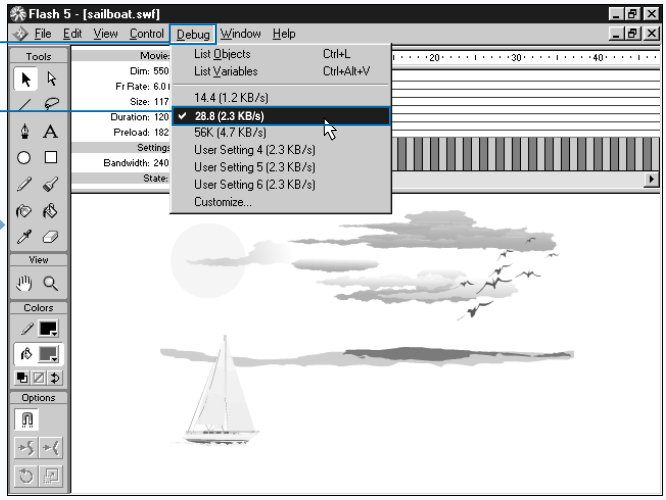
The left side of the Bandwidth Profiler displays the movie clip dimension, frame rate, size, duration in frames and seconds, and the preloaded frames in seconds. The right side of the Bandwidth Profiler displays a graph. Each bar in the graph represents a frame. If the bar is below the red line, the frame will stream in real-time. If the bar is above the red line, the frame must wait to stream.

You can use the streaming bar to simulate the number of frames loaded and the frame currently playing. You can click on any bar to display information about the frame. You can also view a graphical representation of each frame.

The test environment also has options that enable you to play, rewind, step forward, step backward, or loop the playback of your movie.

ENTER THE TEST ENVIRONMENT





1 Click Control ➪ Test Movie to move to the test environment.

■ Your movie plays automatically.

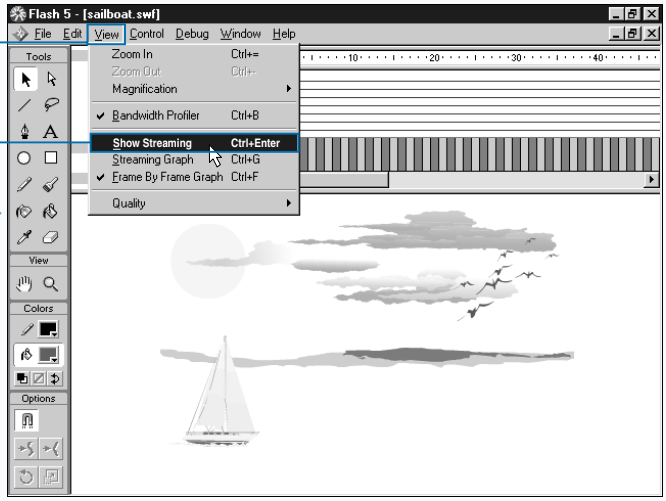
2 Click Debug and select the modem speed at which you want to test your movie.

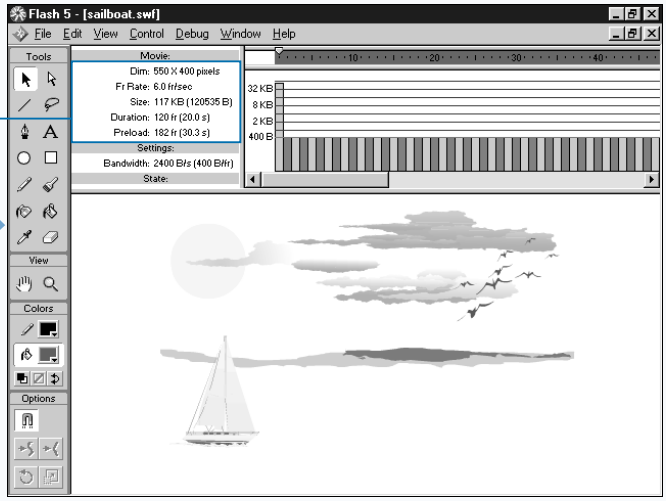
Extra

When in the test environment, you can quickly move to the Actions panel by clicking Window ➪ Actions from the menu.

You can zoom in on your movie by clicking View ➪ Zoom In. You can zoom out by clicking View Zoom Out. Set your movie quality by clicking View ➪ Quality from the menu.

You can choose to display the Status bar, Controller, or Main toolbar by clicking Window ➪ Toolbars from the menu. You can use the Controller to play, rewind, step forward, step backward, or loop the playback of your movie. You can use the Main toolbar to open a file. The Status bar displays the state of the Caps Lock and Num Lock keys.





3 Click View ➪ Show Streaming to simulate the streaming of your movie.

■ Flash simulates the streaming of your movie at the modem speed you selected.

■ The bandwidth profiler provides you with information about the dimension, size, duration, and preload of your movie.

ADD COMMENTS

Adding comments to your Flash movie is essential when you are collaborating with others in its creation or when your script is complex and you want to document the purpose of each step. Adding comments is also a good programming practice. The syntax for adding a comment is

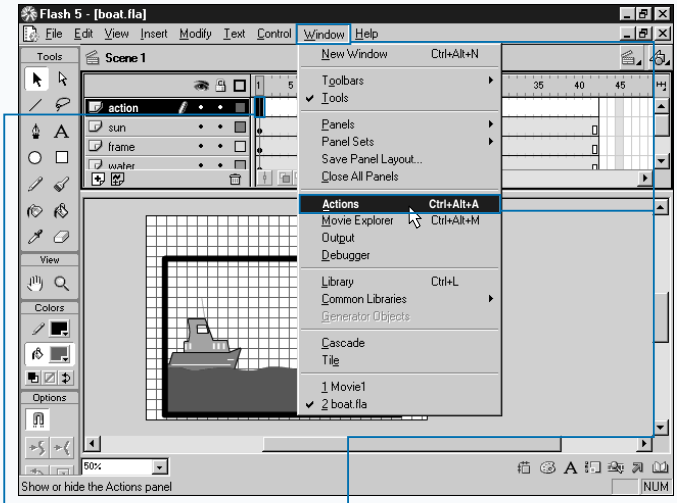
`// comment` or `/* comment */`.

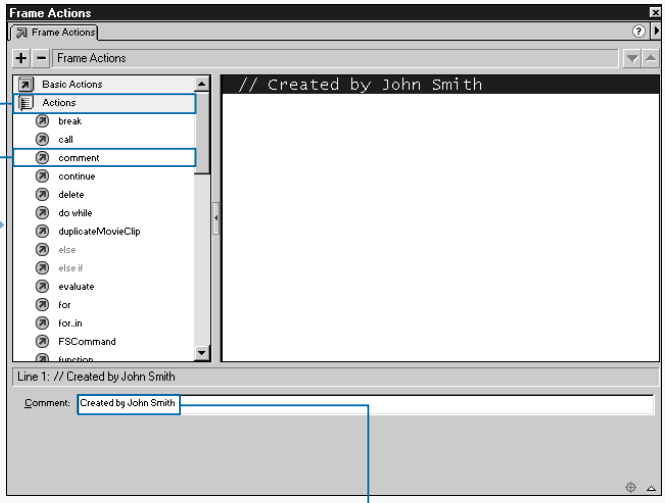
You can precede single-line comments with `//` or you can precede them with a `/*` and end them with a `*/`. For multiline comments, begin each comment with a `/*`, precede subsequent lines with an `*`, and end with an `*/`. You cannot select the `/*` syntax from the Toolbox list in Normal Mode. You can add comments to any button, frame, or movie clip action. If you have colored syntax turned on, comments appear in magenta in the Actions list. When you execute your script, Flash ignores comments. When you are debugging, it is sometimes useful to comment out sections of code so you can execute the remaining code without the commented section. This can help you locate problems.

You can place comments anywhere in your script. Comments do not affect the size of your exported movie. A frequent use of comments is documenting the writer of the script, the date written, its purpose, and any revisions.

In Flash, you can also add comments to frames on the Timeline. Here also, comments are useful when you are working in a collaborative environment. Flash does not export frame comments with the movie, so they do not affect the size of the exported movie. You should put comments on a separate layer.

ADD COMMENTS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

■ This example uses a frame.

2 Click Window ➪ Actions to open the Actions panel.

3 Click Actions.

4 Double-click comment.

5 Type your comment.

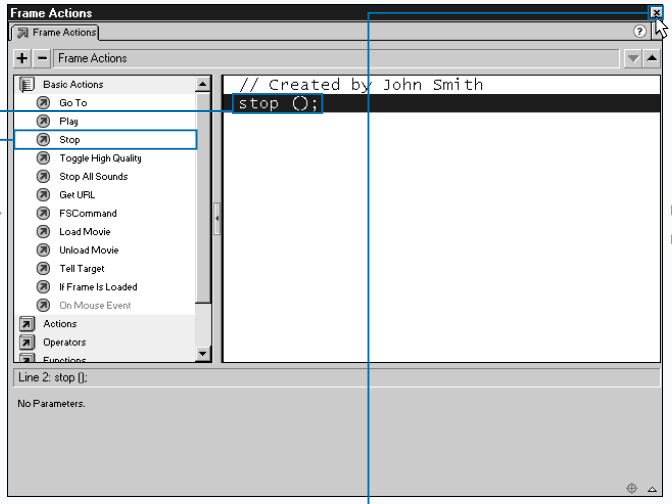
Extra

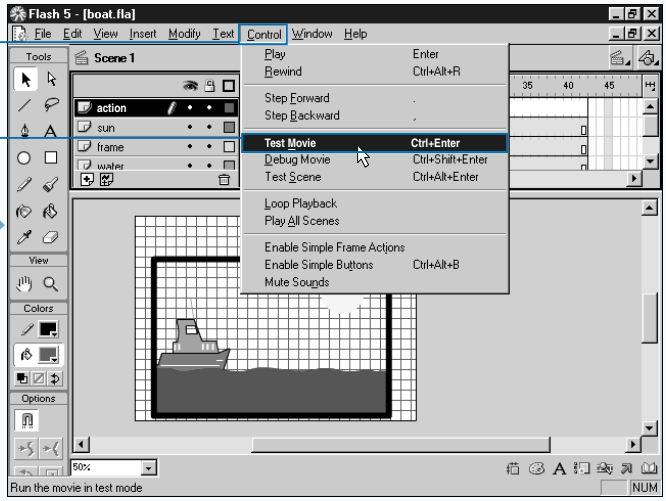
You can add comments to Timeline frames. To create a frame comment, create a comment layer, select the frame to which you want to add a comment, click Windows ➪ Panel ➪ Frame to open the Frame panel, and type `//` followed by your comment in the Label field.

You can also add comments using the syntax that follows:

```
/* comment
* comment
* comment
 */.
```

This syntax enables you to spread a comment over several lines.





6 Click to select the statements you want to execute.

7 Click ☒ to close the Actions panel.

8 Click Control ➪ Test Movie to move to the test environment.

9 Test your movie.

Note: See page 38 for information on testing your movie.

■ When you execute your script, Flash ignores comments.

STOP A MOVIE

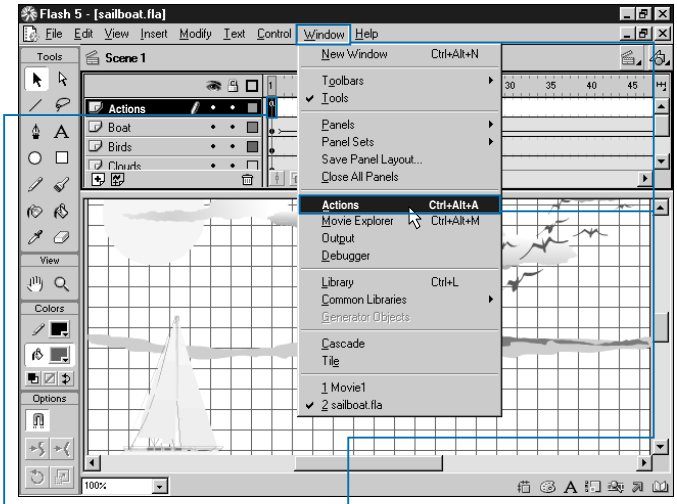
As soon as you load a Flash movie, it begins to play and continues to play unless you stop it. You can use the `stop` action to stop a movie. The syntax for the `stop` action is

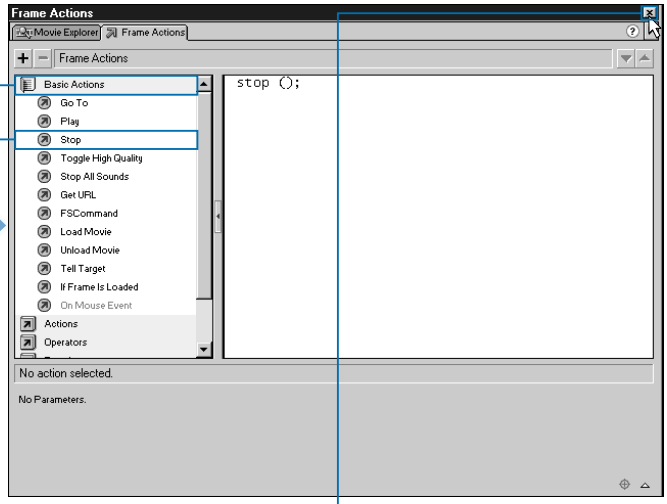
```
stop();.
```

The `stop` action does not take any arguments, so it has no parameters. You can use the `stop` action to stop the main Timeline or to stop a movie clip. If you want to stop the main Timeline, simply issue a `stop` action. If you want to stop a movie clip, the movie clip must be on the Stage, it must have an instance name, and you must target the movie clip. You name a movie clip in the Instance panel. For more about naming movie clips, see Chapter 1. Targeting a movie clip means you designate the movie clip on which you want ActionScript to perform an action. For more about targeting movie clips, see Chapter 10.

You can stop a movie or movie clip at any point. Movies begin playing as soon as they load. You can stop a movie in the first frame and have users press a button when they are ready to start it. Or you can stop a movie at the end of a scene and create buttons to determine the next scene to play. You can assign the `stop` action to any button, frame, or movie clip; however, you will most frequently use the `stop` action when you want to use buttons to control the action in your movie.

STOP A MOVIE





1 Click the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file stop.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a frame.

■ Flash movies begin playing upon load.

2 Click Window ➪ Actions to open the Actions panel.

3 Click Basic Actions.

4 Double-click Stop.

■ The `stop` action appears in the Action list.

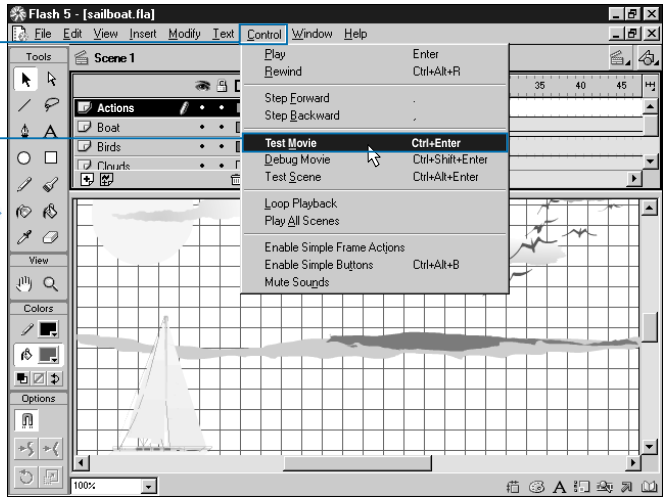
5 Click ☒ to close the Actions panel.

Extra

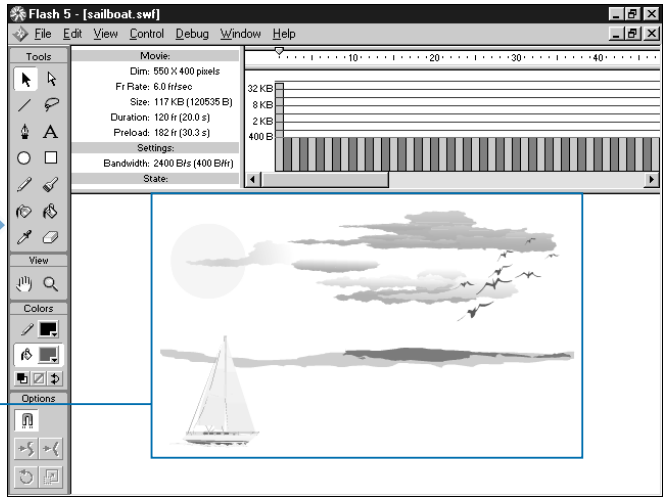
You can use a script similar to the one shown here to stop a movie clip. In this example, the script is associated with a button. The `on(release)` action tells Flash to begin the action when the user releases the mouse after clicking the button. The name of the movie clip is `smallMC`. The `tellTarget("smallMC")` statement tells Flash you want to perform an action on the `smallMC` movie clip. The `stop()` action tells Flash to stop the movie clip identified in the `tellTarget` action. In this example, Flash stops `smallMC`.

Example:

```
on (release) {
    tellTarget ("smallMC") {  stop ();
  }
}
```



6 Click Control ➪ Test Movie to move to the test environment.



7 Test your movie.

Note: See page 38 for information on testing your movie.

■ Your movie displays on the screen. Because you added a `stop` action, the movie will not play.

PLAY A MOVIE

When you stop a movie or movie clip, it remains stopped until you explicitly issue a statement to start it. You use the `play` action to start a movie. The syntax for the `play` action is

```
play();.
```

The `play` action does not take any arguments so it has no parameters. You can use the `play` action to start the main Timeline or to start a movie clip. If you want to start the main Timeline, simply issue a `play` action. If you want to start a movie clip, the movie clip must be on the Stage, it must have an instance name, and you must target the movie clip. You name a movie clip in the Instance panel. For more about naming movie clips, see Chapter 1. Targeting a movie clip means you designate the movie clip on which you want ActionScript to perform an action. You can use the `play` action with `tellTarget` to start a movie clip. For more on `tellTarget`, see Chapter 10. You can also use the MovieClip object `play` method to start a movie clip. The syntax for the `play` method is

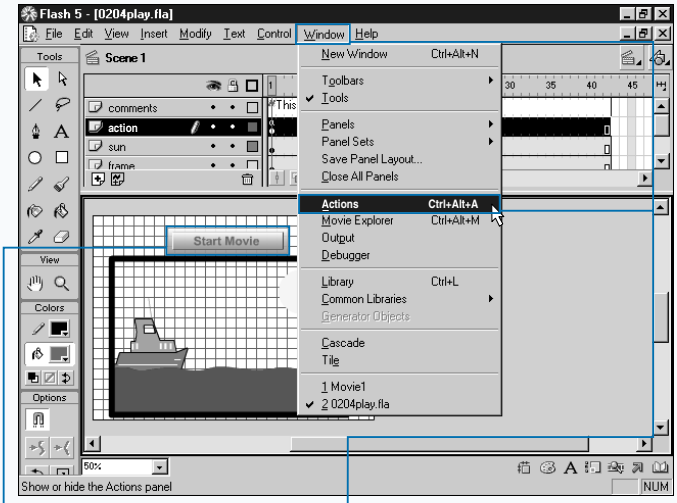
```
instanceName.play();
```

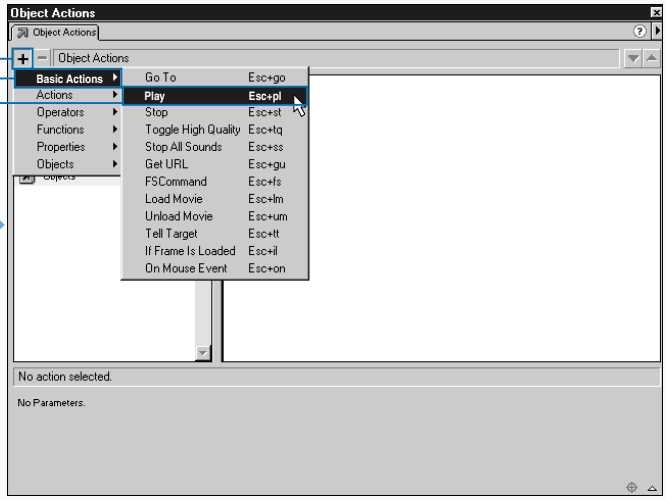
Use the `instanceName` argument to specify the movie clip you want to play. The following example starts `sampleMC`:

```
sampleMC.play();
```

You can start a movie or movie clip, at any point. For example, you can start a movie or movie clip when the user clicks a button, when the movie reaches a specified frame, or when the user presses a key. You can assign the `play` action to any button, frame, or movie clip.

PLAY A MOVIE





1 Select the frame, button, or movie clip to which you want to add ActionScript.

■ This example uses a button.

Note: This example uses file play.fla, which you can find on the CD-ROM that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

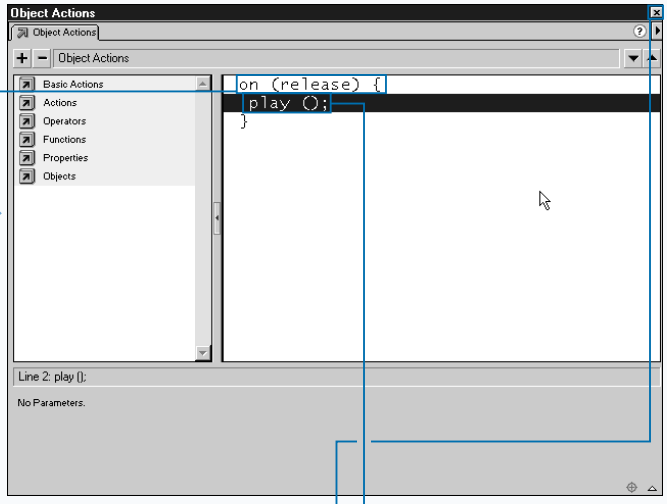
3 Click ⊞ ➪ Basic Actions ➪ Play to select the `play` action.

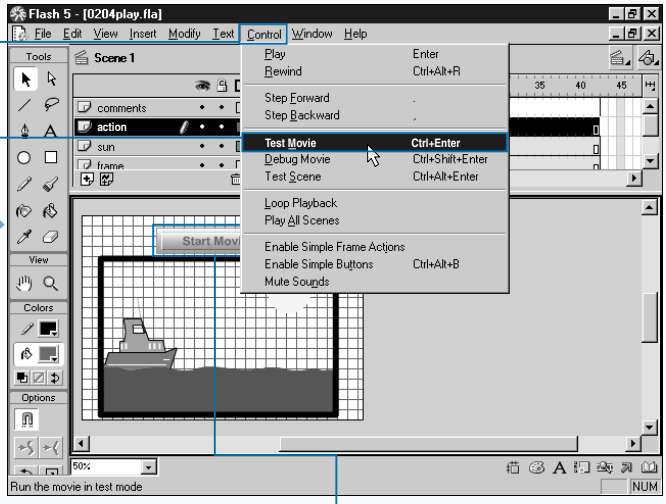
Extra

You can use a script similar to the one shown in this example to restart a movie clip. In this example, the script is associated with a button. The `on(release)` action tells Flash to begin the action when the user releases the mouse after clicking the button. The name of the movie clip is `smallMC`. The `tellTarget("smallMC")` statement tells Flash you want to perform an action on the `smallMC` movie clip. The `play()` action tells Flash to start the movie clip identified in the `tellTarget` action. In this example, Flash starts `smallMC`.

Example:

```
on (release) {
    tellTarget ("smallMC") {  play ();
  }
}
```





■ The `on (release)` handler defaults.

■ The `play` action appears in the Action list.

4 Click ☒ to close the Actions panel.

5 Click Control ➪ Test Movie to move to the test environment.

6 Test your movie.

Note: See page 38 for information on testing your movie.

■ When you click the Start Movie button, the movie plays.

JUMP TO A FRAME OR SCENE

You can use the `GoTo` action to create loops or to give the user the ability to move to a desired location at will. The `GoTo` action has two options: Go To and Play and `GoTo` and Stop. The syntax for the Go To action is

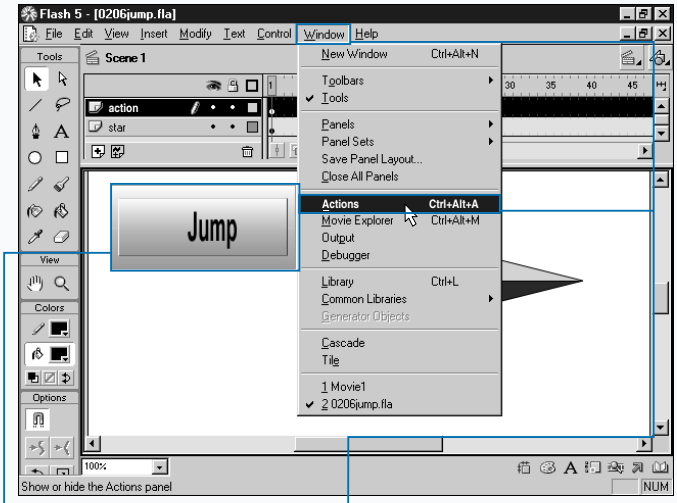
```
gotoAndPlay(scene,frame);
gotoAndStop(scene,frame);
```

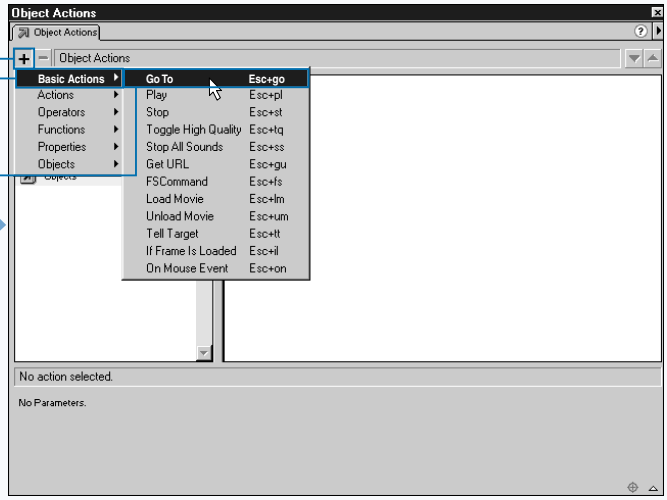
You use the `scene` argument to tell ActionScript the name of the scene to which you want to send the playhead. The `scene` argument is optional. If you do not enter a scene name, the playhead goes to the specified frame in the current scene. You use the `frame` argument to tell ActionScript the frame to which you want to send the playhead. It is best to use a label instead of a frame number to identify the frame. Labels move with frames, but frame numbers change if you add, remove, or change the location of a frame. If you use a frame number to identify a frame, a change in the frame number can cause an error in your script. You use the Frame panel to create a frame label. For more about frame labels, see Chapter 1.

You can use an expression to identify the frame you want to go to, for example, `gotoAndStop(_currentframe + 10)`. The _current frame property retrieves the number of the current frame. The example adds 10 to the current frame. It tells ActionScript to move 10 frames ahead and stop the movie.

You can use the `GoTo` action to create a loop by creating a frame action that goes to a prior frame. By creating buttons that take users to a particular frame or scene, you can create movies in which users can jump from place to place.

JUMP TO A FRAME OR SCENE





1 Click the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file jump.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

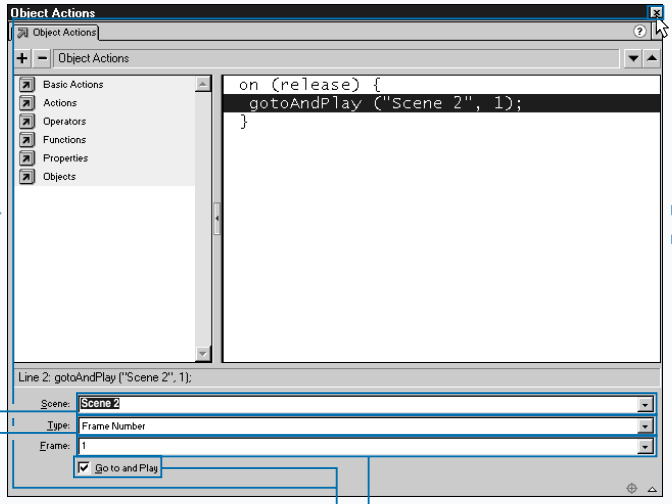
3 Click ⊞ ➪ Basic Actions ➪ Go To to select the `goto` action.

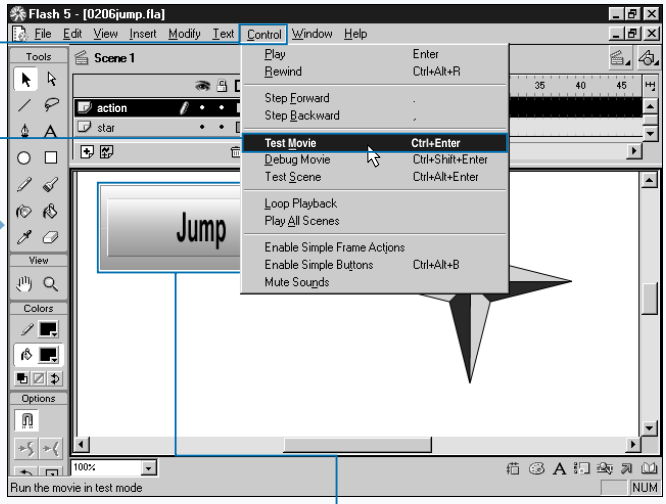
■ The `on (release)` handler defaults.

Extra

If you select the `GoTo` action in Normal Mode, you can select `<next scene>` or `<previous scene>` in the Scene field to retrieve the `nextScene()` or `prevScene()` actions, respectively. The `nextScene()` action sends the playhead to the first frame of the next scene and stops the playhead. The `prevScene()` action sends the playhead to the first frame of the previous scene and stops the playhead.

If you select the `GoTo` action in Normal Mode, you can select Next Frame or Previous Frame in the Type field to retrieve the `nextFrame()` and `prevFrame()` actions, respectively. The `nextFrame()` action sends the playhead to the next frame and stops the playhead. The `prevFrame()` action sends the playhead to previous frame and stops the playhead.





■ The `GoTo` action appears in the Action list.

4 Click ▼ to select the scene to which you want to go.

5 Click ▼ to select a frame type.

6 Type a frame number or label.

7 Click to select Go to and Play (☐ changes to ☑).

8 Click ☒ to close the Actions panel.

9 Click Control ➪ Test Movie to move to the test environment.

10 Test your movie.

Note: See page 38 for information on testing your movie.

■ When you click the jump button, you move to the scene specified in the `goto` action.

SET MOVIE QUALITY

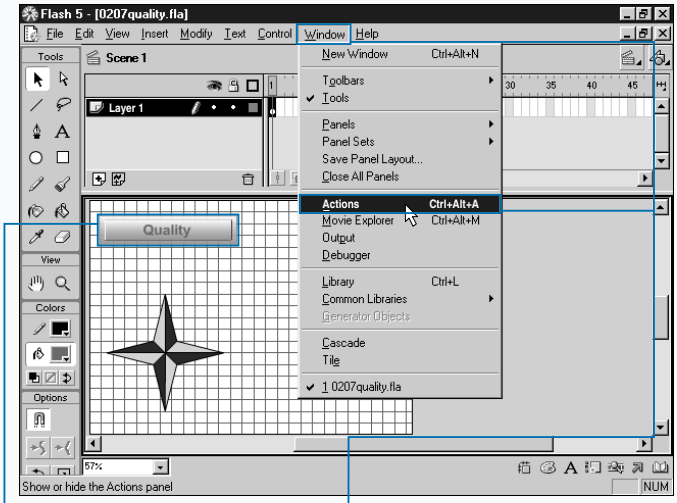
Flash uses antialiasing to smooth the edges of images. With antialiasing, Flash displays crisp, clear images. However, because antialiasing requires a faster processor, it can slow down the playback of a movie. You can use the `toggleHighQuality` action to enable users to toggle antialiasing off and speed up the playback of the movie. The syntax for `toggleHighQuality` is

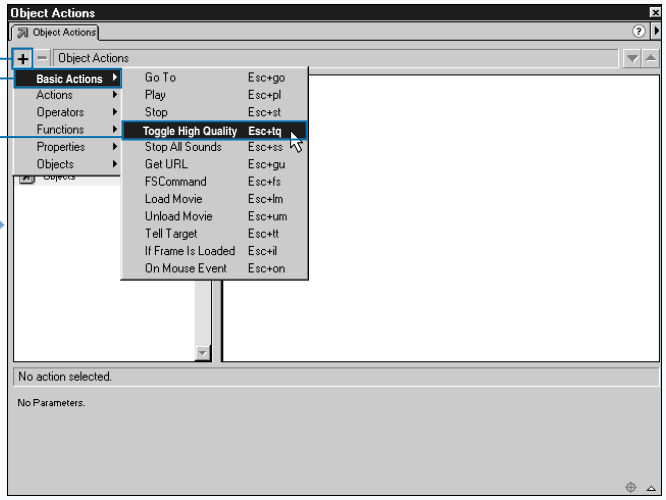
```
toggleHighQuality();.
```

The `toggleHighQuality` action does not take any arguments, so it has no parameters. Turning off antialiasing affects all of the movies in the player. You cannot adjust the quality of a single movie or movie clip. If you assign `toggleHighQuality` to a button, users can toggle antialiasing on and off by clicking the button. If antialiasing is on, clicking the button turns it off. If antialiasing is off, clicking the button turns it on. When a Flash movie is viewed in the stand-alone player, clicking View ➪ High Quality from the menu toggles antialiasing on and off.

You can create an HTML file to publish your Flash movie on the Web. You can use Flash to create your HTML file by clicking File ➪ Publish Settings and using the Publish Settings dialog box. If you click the HTML tab in the Publish Settings dialog box, you can set the Flash parameters for the HTML document. Among the parameters you can set is the `quality` parameter. The `quality` parameter specifies the level of antialiasing to use when your movie plays on the Web. You choose the level you want to use. You can also set the quality of your movie manually by using the `$QU` variable. If you do not specify a quality value when you create your HTML document, Flash uses the default setting of High.

SET MOVIE QUALITY





1 Click the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file quality.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

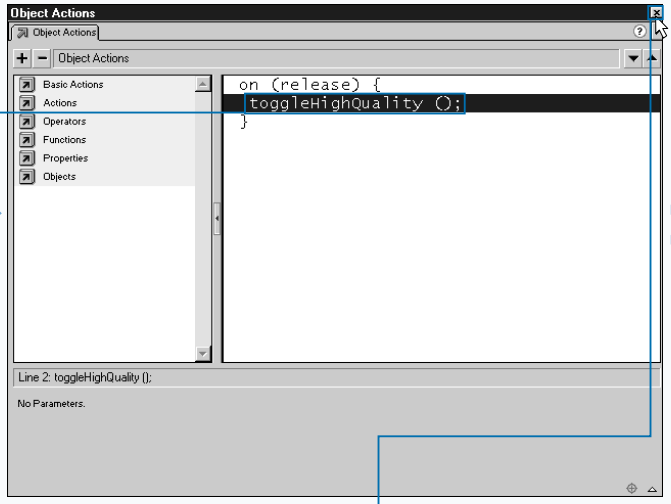
2 Click Window ➪ Actions to open the Actions panel.

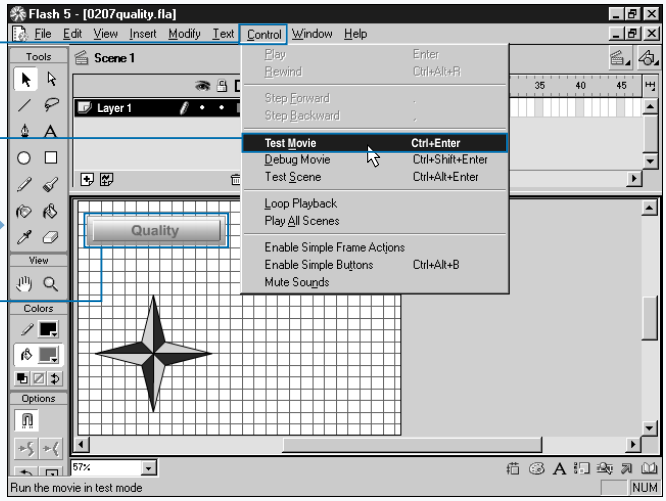
3 Click ⊞ ➪ Basic Actions ➪ Toggle High Quality to select the `toggleHighQuality` action.

Extra

When setting the `quality` parameter in the Publish Settings dialog box, you can select from the following options:

OPTION	DESCRIPTION
Low	Favors playback speed over appearance. Never uses antialising.
Auto Low	Begins with antialiasing turned off. If Flash Player determines that the processor can handle antialiasing, Flash Player turns on antialiasing.
Auto High	Begins with antialiasing turned on. If the Flash Player determines that the processor cannot handle antialiasing, Flash Player turns off antialiasing.
Medium	Uses some antialiasing. Never smooths bitmaps.
High	Uses antialiasing in every instance. This setting favors appearance over playback speed. If the movie does not contain any animation, it smooths bitmaps. If the movie does contain animation, if does not smooth bitmaps.
Best	All output is antialiased. This option does not consider playback speed.





■ The `on (release)` handler defaults.

■ The `toggleHighQuality` action appears in the Action list.

4 Click ☒ to close the Action panel.

5 Click Control ➪ Test Movie to move to the test environment.

6 Click the Quality button to test your movie.

Note: See page 38 for information on testing your movie.

■ Clicking the Quality button toggles antialiasing on and off.

OPEN A WEB PAGE

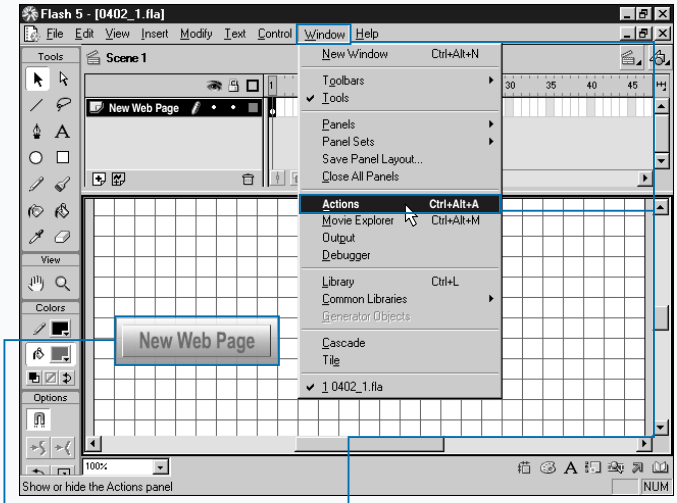
You can use the `getURL` action to open a Web page in a browser window and to pass variables to another application. The `getURL` action is useful when you want the user to move from your movie to another Web page. The syntax for the `getURL` action is

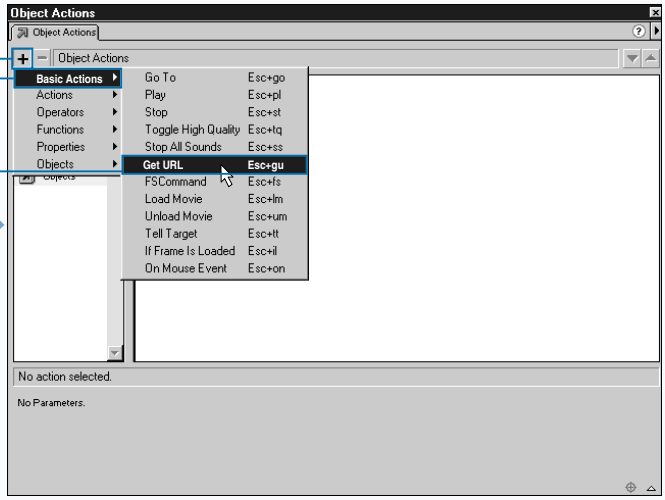
```
getURL(url, window, variables);.
```

You use the `URL` argument to specify the Web address of the page you want to open. You use the window argument to specify the window or frame where the page will open. Choosing `_self` opens the page in the current frame or window, `_blank` opens the page in a new window, `_parent` opens the page in the parent of the current frame, and `_top` opens the page in the top-level frame of the current window.

You can also use the `getURL` action to send variables to an application at the specified URL. You can include the variables you wish to send in your script, or you can use dynamic and input text boxes to create your variables. See Chapter 1 for more about dynamic and input text boxes. See Chapter 4 for more about variables. You use the variable argument to specify the method you wish to use to send variables. In Normal mode, the Variable field in the parameter pane presents you with three choices: Send Using Get, Send Using Post, and Don't Send. Use Send Using Get to append a small number of variables to the end of the URL. Use Send Using Post to send variables separate from the URL. Send using Post enables you to send a larger number of variables. Use Don't Send if you do not want to send any variables.

OPEN A WEB PAGE





1 Click the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file url.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Click ⊞ ➪ Basic Actions ➪ Get URL to select the `getURL` action.

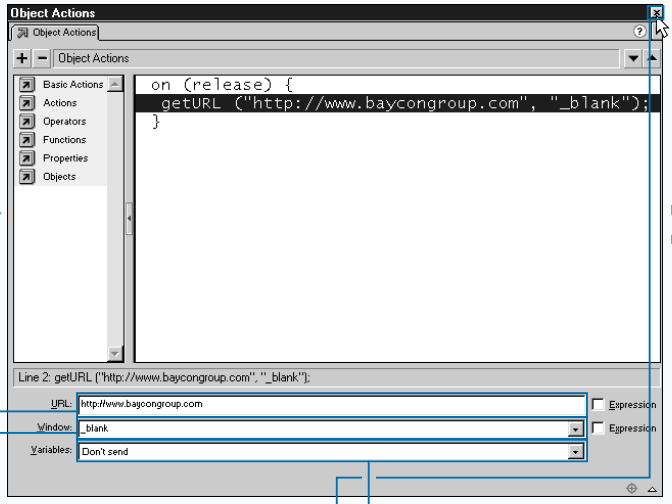
■ The `getURL` action appears in the Action list.

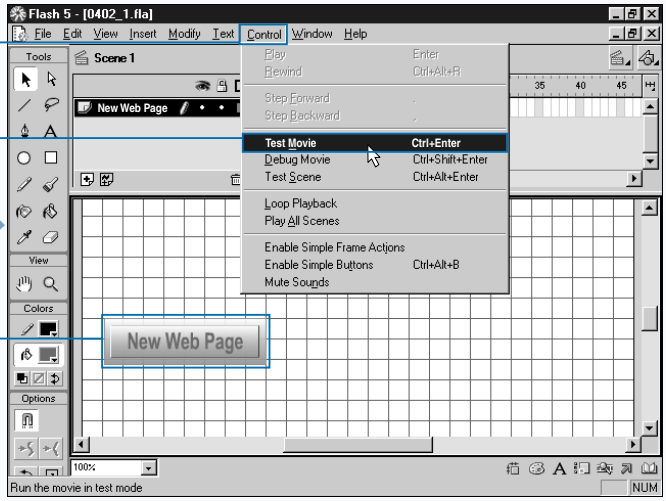
■ The `on (release)` handler defaults.

Extra

You can also open a Web page by creating a link to the Web page in a text field. To create a link, use the Arrow tool to select a Text field. Click Text ➪ Character from the menu to open the Character panel. In the URL field, type the URL to which you want to link.

You can use `getURL` to create menus. Simply use the `getURL` action with a series of buttons. You can easily create many types of menus using Flash. For a rollover menu use the botton `on rollover` handler. Use can use the `_visible` property to make menu options visible on press or on rollover. You can also display images on the screen when the user rolls over the menu option.





4 Type the URL to which you want to go.

5 Click ▾ to select the window or frame in which the page will open.

6 Click ▾ to select the method you want to use to send variables.

7 Click ☒ to close the Actions panel.

8 Click Control ➪ Test Movie to move to the test environment.

9 Click the New Web Page button to test your movie.

Note: See page 38 for information on testing your movie.

■ You can click the New Web Page button to open a Web page.

COMMUNICATE WITH THE FLASH PLAYER

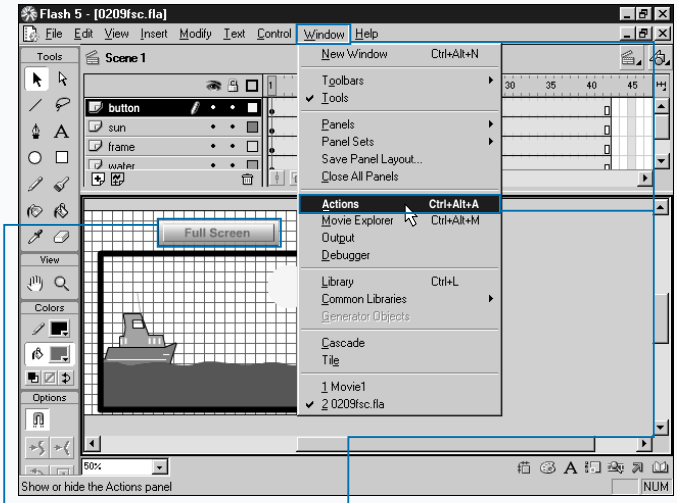
Flash Player is a stand-alone projector that you can use to view Flash movies. Flash Player enables you to view Flash movies outside of Flash without a Web browser. Flash Player is a stand-alone player, similar to a movie projector. If you have Flash Player, you do not need anything else to view your movie. You can create movies specifically for Flash Player. Simply select Windows Projecter or Machintosh Projector in the Publish Settings dialog box.

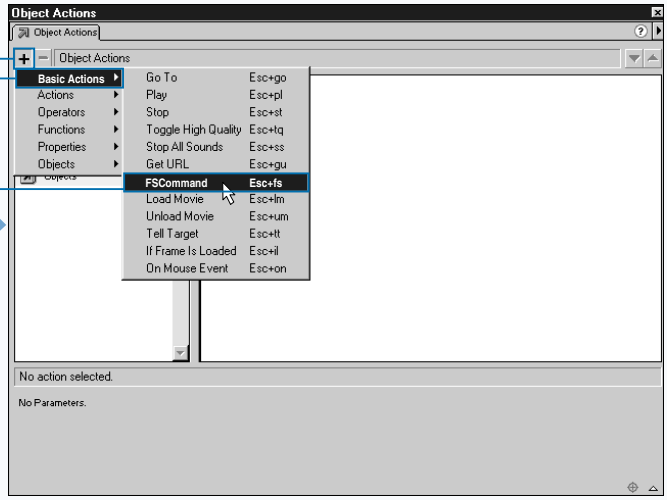
Flash player has a filename of `FlashPla.exe` in Windows and `FlashPlayer` on a Macintosh computer. You can use the `fscommand` to communicate with the Flash Player. The syntax for `fscommand` is

```
fscommand(command,argument).
```

You can use the command argument to tell ActionScript what command you want to execute. You can choose from `quit`, `exec`, `fullScreen`, `allowScale`, and `showMenu`. You can use argument parameters to specify the arguments associated with the command you want to execute. The `quit` option closes the projector window. The `exec` command starts an application from the projector. You can use the path to the application as the argument. The `fullScreen` command controls the size of the projector screen. You can use `true` as the argument for full-screen; use `false` for normal view. The `allowScale` option controls the scaling of the movie. You can use `true` as the argument if you want the animation to scale with the size of the screen; use `false` if you do not want the animation to scale with the size of the screen. The `showMenu` command controls the right-click menu and the menu bar. You can use `true` as the argument to display the menus; use `false` to hide the menus.

COMMUNICATE WITH THE FLASH PLAYER





1 Click the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file fsc.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Click ⊞ ➪ Basic Actions ➪ FSCommand to select the `fscommand` action.

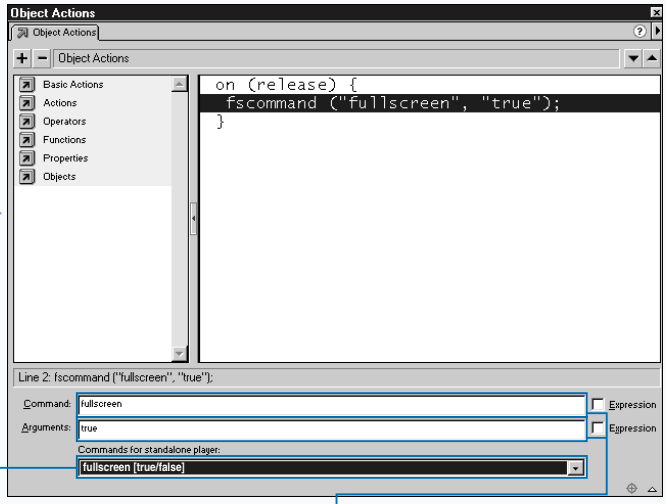
■ The parameters panel opens.

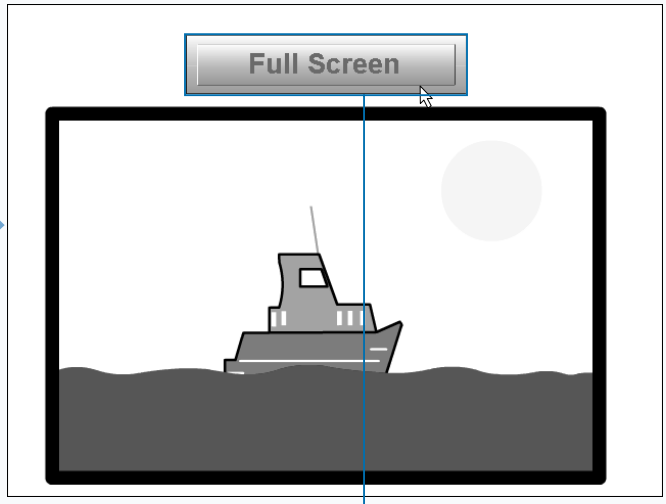
■ The `on (release)` handler defaults.

Extra

You can also use the `fscommand` to send messages to a scripting language such a JavaScript. You can pass any two arguments using the `fscommand` action to any JavaScript function that handles the `fscommand` action. The command and argument parameters send the arguments to the JavaScript function `Fscommand`.

The `fscommand` action invokes the JavaScript function `moviename_Dofscommand`. If you publish your movie using the Flash with `fscommand` template in the HTML Publish Settings dialog box, the movie's Name and ID attributes will be the filename.





■ The `fscommand` action appears in the Action list.

4 Click ▾ to select a command for a standalone player.

5 Type a command in the command field.

6 Type an argument in the argument field.

7 Publish your movie.

8 Open your Flash Player and test your movie.

■ Your movie plays.

■ You can click the Full Screen button to change your movie to full screen mode.

CREATE OBJECTS USERS CAN DRAG

You can use the `startDrag` action to create objects users can drag. Any movie clip or button can be draggable. The syntax for the `startDrag` action is

```
startDrag(target,lock,left,top,right,bottom)
```

Use the `target` argument to specify the path to the movie clip or button you want to make draggable. You can use the `this` keyword as the target argument. Using the `this` keyword makes the object to which you attach ActionScript draggable.

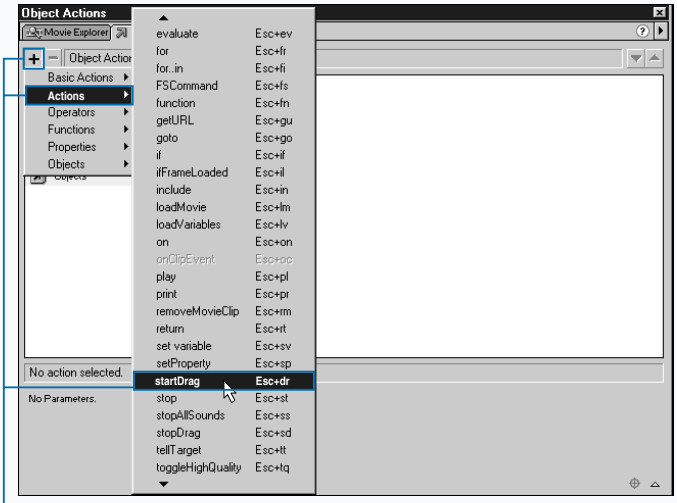
Use the `lock` argument to specify the Boolean value `true` if you want Flash to lock the pointer to the center of the object as the user drags. Use the Boolean value `false` if you want Flash to lock the pointer to the point at which the user pressed the mouse button. The `lock` argument is optional.

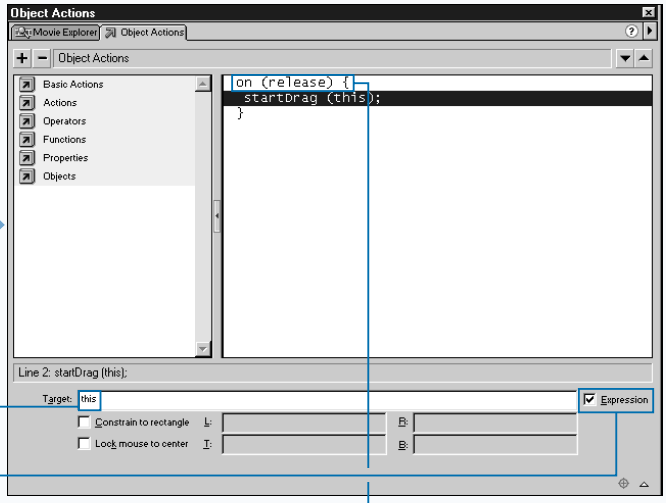
Use the `left`, `right`, `top`, and `bottom` arguments to specify the rectangle within which the user can drag the object. You specify the area by using the number of pixels from the top and left borders of the movie. The upper left corner of the movie has a value of 0, increasing as you move downward. The upper left corner of the movie has a value of 0, increasing as you move across. These arguments are optional.

Only one object can be draggable at a time. An object remains draggable until you execute a `stopDrag` action or another `startDrag` action. You use the `stopDrag` action to stop the current drag operation.

The syntax for the `stopDrag` action is `stopDrag();`. The `stopDrag` action does not take any arguments, so it has no parameters.

CREATE OBJECTS USERS CAN DRAG





1 Click a button.

2 Click ⊞ ➪ Actions ➪ startDrag to select the `startDrag` action.

■ The `startDrag` action appears in the Action list.

■ The Parameters panel opens.

3 Type **this** in the Target field.

4 Click Expression.

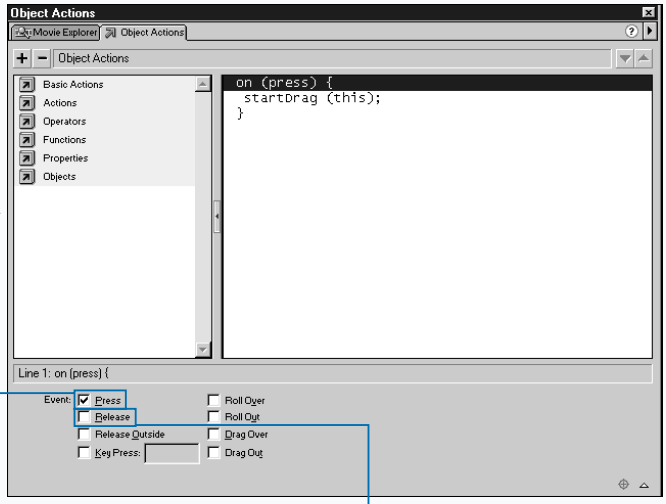
5 Click `on (release)`.

■ The handlers Parameters panel opens.

Extra

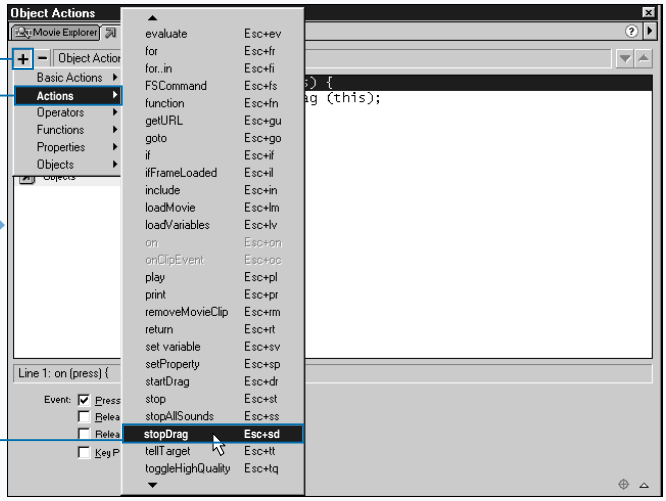
There are many uses for draggable objects. You can create a draggable object and use it as a scroll bar. The user can use the scrollbar to scroll through a block of text.

You can also use a draggabe object to increase or decrease the volume and panning of sound. For example, as the user drags the bar to the right, your sound can pan to the right speaker. As the user drags the bar to the left, your sound can pan to the left speaker. You can increase and decrease the volume as the user drags the scroll bar.



6 Click to select Press (☐ changes to ☑).

7 Click to deselect Release (☑ changes to ☐).



8 Click ⊞ ➪ Actions ➪ stopDrag.

■ The `stopDrag` action appears in the Action list.

9 Click `on (release)`.

■ The handlers parameters panel opens.

CONTINUED

CREATE OBJECTS USERS CAN DRAG (CONTINUED)

Frequently when creating a draggble object, you want to create an object that is draggable when the user clicks on it and and remains draggable until the user releases the mouse. You must begin by creating a button. However, because you cannot target a button, later you will turn the button into a movie clip. Use the `startDrag` action with the `on (press)` handler to make the object draggable when the user clicks it. Decide the name you are going to assign the movie clip. Use that name as the target argument, or use the `this` keyword. Use the `stopDrag` action with the `on (release)` and `on (rollout)` handlers to stop the action when the user releases the mouse.

Select the button on the Stage. Use the Symbol Properties dialog box to make the button a movie clip. If you targeted a movie clip, name the movie clip the name you targeted in the `startDrag` action.

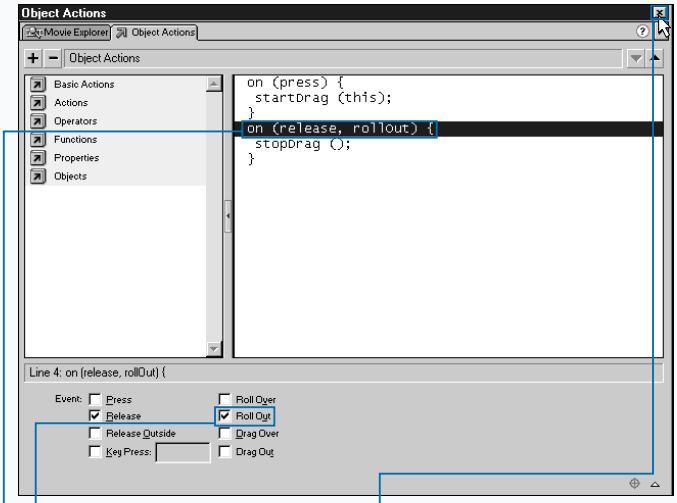
The `startDrag` and `stopDrag` actions are used to create sliders, scrollbars, panels, and many other draggable objects.

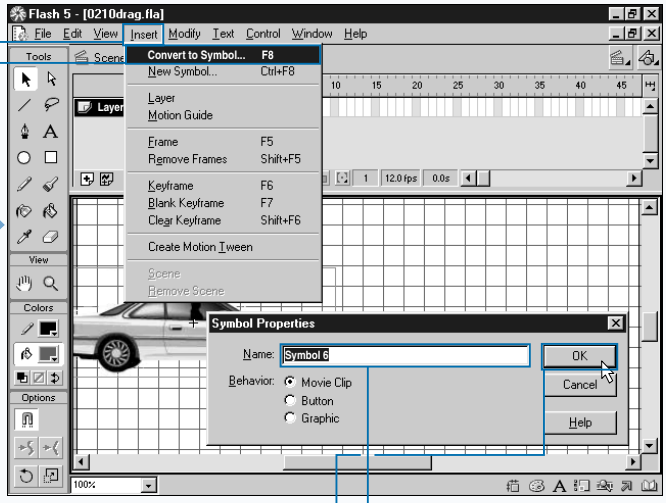
You use the `_droptarget` property to retrieve the name on the instance on which the last draggable object was dropped. The `_droptarget` property returns the absolute path of the instance using slash notation. For more information on absolute paths and slash notation, see Chapter 10. The syntax for the `_droptarget` property is

```
instanceName._droptarget;
```

Use the `instanceName` argument to specify the name of the instance that was the target of the `startDrag` action. You use the `_droptarget` action when you want your script to respond based on where the user placed the draggable object.

CREATE OBJECTS USERS CAN DRAG (CONTINUED)





10 Click to select Roll Out (☐ changes to ☑).

■ On Release is preselected.

11 Click ☒ to close the Actions panel.

12 Click Insert ➪ Convert to Symbol to open the Symbol Properties dialog box.

13 Type a symbol name.

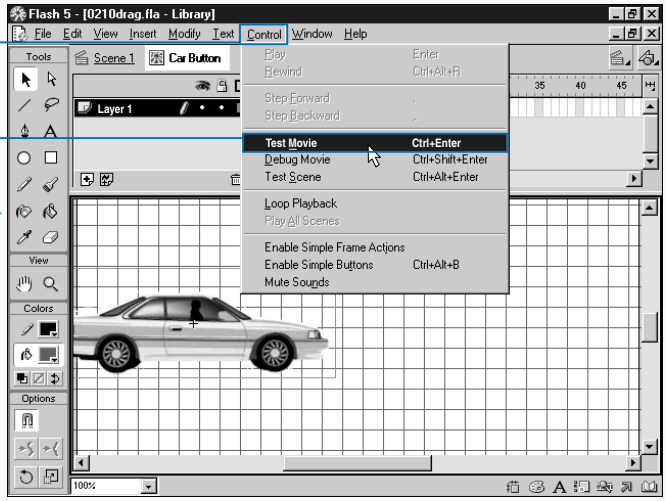
14 Click OK.

Extra

The script in this example constrains to 50 pixels over and 50 pixels down from the upper left corner of the Stage area in which the user can drag a button. The `on (press)` action tells Flash to perform an action when the user presses the mouse while the pointer is over a button. The `startDrag` action is the action Flash performs. The `this` keyword tells Flash to make the object to which the script is attached draggable. The `false` argument locks the pointer to the point at which the user pressed the mouse button. The 0, 0, 50, and 50 arguments specify the area in which the object can be dragged. The `on (release)` and `stopDrag` arguments deactivate the drag action when the user releases the mouse.

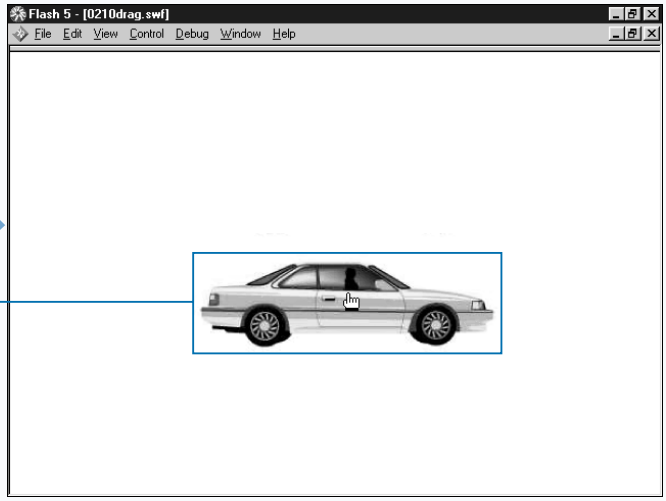
Example:

```
on (press) {
  startDrag (this, false, 0, 0, 50, 50);
}
on (release) {
  stopDrag ();
}
```



15 Click Control ➪ Test Movie to move to the test environment.

Note: See page 38 for information on testing your movie.



16 Click on the instance and drag to test your movie.

■ Your movie clip is draggable.

PRINT A MOVIE

You can use the `print` and `printAsBitmap` actions to print a movie or a movie clip. Use the `print` action to print Flash movies as vector graphics. Use the `printAsBitmap` action to print Flash movies as bitmap images. If you print a movie as a vector graphic, the artwork scales and prints clearly at any size. If you print a movie as a bitmap, the artwork does not scale and might not print clearly. If you use alpha transparencies or other color effects in your movie, you cannot use the `print` action. The syntax for the `print` and `printAsBitmap` actions is

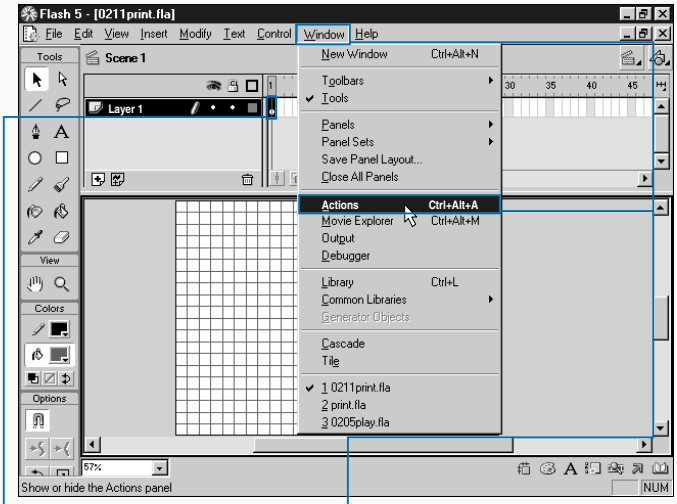
```
print (target, "bounding")
printAsBitmap (target, "bounding")
```

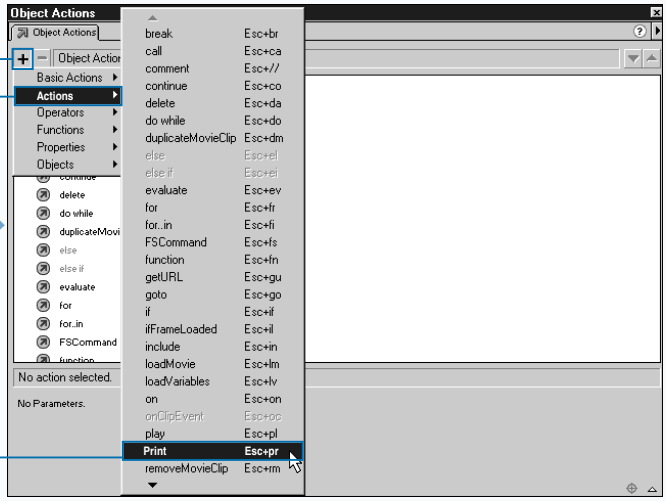
You can use the target argument to specify the level or target path of the movie or movie clip you want to print. For an explanation of levels and target paths, see Chapter 10. Use the `bounding` argument to tell Flash how to print each frame. Use `bmovie` to tell Flash to use the area of the object in a specific frame as the area to print. Select the object used to define the print area and label the frame #b. Use `bmax` to have the size of the content in each printed frame determine the print size of each frame. Use `bframe` to have each printed frame fill the printed page.

By default, all frames print. If you want only specific frames to print, give a label of #p to each frame you want to be printable.

ActionScript sometimes needs to substitute `print` and `printAsBitmap` with `printNum` and `printAsBitmapNum`, respectively. If you are entering your script in Normal Mode, ActionScript makes the substitution automatically. For an explanation of why ActionScript makes this substitution, see the Appendix.

PRINT A MOVIE





1 Click the frame, button, or movie clip to which you want to add ActionScript.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Click ⊞ ➪ Actions ➪ Print to select the `print` action.

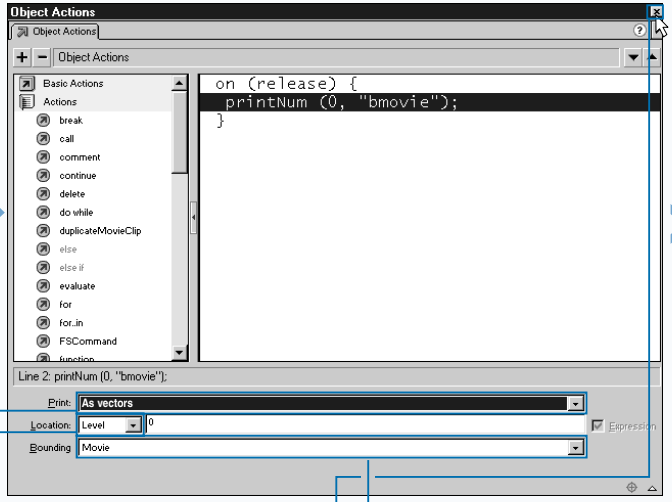
■ The `on (release)` handler defaults.

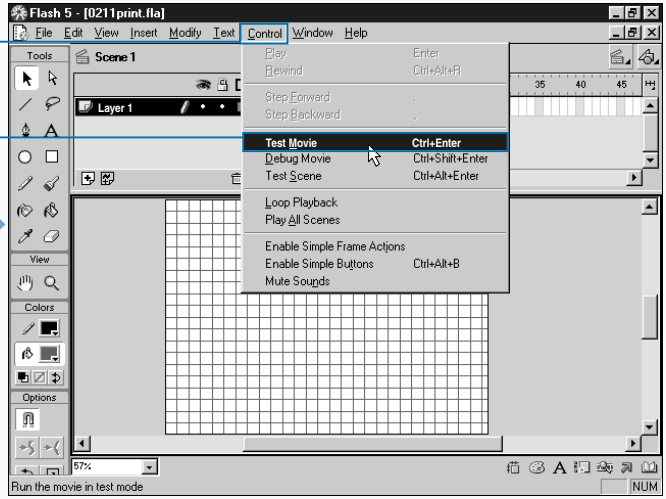
■ The `print` action appears in the Action list.

Extra

The user can print a Flash movie by using the Print command on the Flash Player context menu or by clicking File ➪ Print from the stand-alone player menu. The user activates the Flash Player context menu by right-clicking in Windows or by pressing Ctrl-click on a Macintosh. By default, Flash Player prints all frames on the main Timeline. If you use the frame label #p, Flash prints only frames labeled #p. If you use the frame label #b, Flash uses that frame as the print area. If you have not labeled a frame #b, Flash Player uses the Stage as the print area. If you do not want your movie to be printable, label a frame !#p to dim the print command on the Flash Player menu. Your users must have Flash Player version 4.0.25 for Windows or version 4.0.20 for Macintosh to take advantage of any print functionality you add to your movie.

A movie clip must be on the Stage and have an instance name to be printable. A movie must be fully loaded before it can print. You can use the `totalframes` and `_framesloaded` properties to determine if all the frames have been loaded.





4 Click ▾ to select a print method.

5 Click ▾ to select a print location.

6 Click ▾ to select a bounding method.

7 Click ☒ to close the actions panel.

8 Click Control ➪ Test Movie to move to the test environment.

9 Print your movie.

Note: See page 38 for information on testing your movie.

■ You can print your movie from the test environment by right clicking in Windows or by Control-clicking on a Macintosh to bring up the context menu. Select Print from the menu.

CHECK FRAME LOAD

You use the `ifFrameLoaded` action to make sure that the download of movie contents to the local computer has completed before you trigger an ActionScript statement or series of statements. Developers frequently use the `ifFrameLoaded` action to keep the movie from beginning until the browser has downloaded the entire movie contents. The syntax for the `ifFrameLoaded` action is

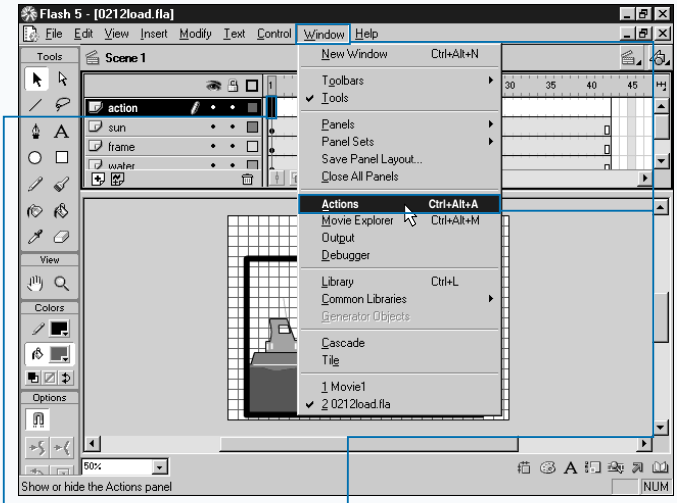
```
ifFrameLoaded(scene, frame) {
statement;};
```

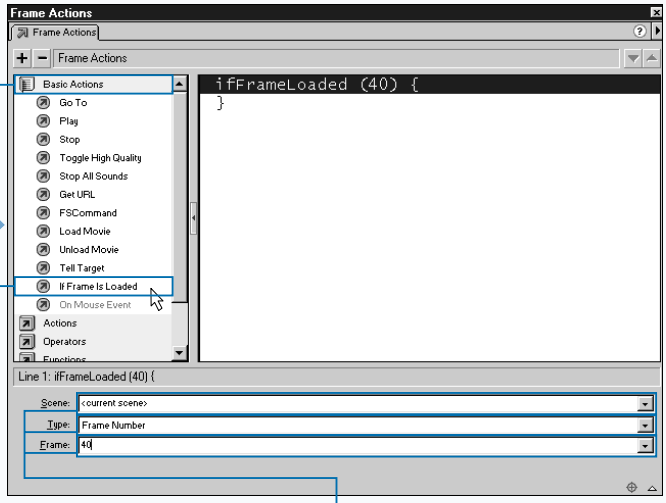
The `scene` argument is optional. Use it to specify the scene whose download you want to check. Use the frame argument to specify the frame number or frame label that Flash must load before Flash executes the statement. Use the statement argument to specify the statement or series of statements to execute.

You can create a preloader with the `ifFrameLoaded` action. A preloader is a short animation that plays while the rest of the animation is loading. For example, you can create a preloader that reads, Please Wait... Movie Loading. The preloader stays on the screen until the specified frames have downloaded completely. After the specified frames have downloaded, the rest of the animation begins. When using a preloader, place the preloader in the first frames on the Timeline. You can use the `gotoAndPlay` command and create a loop that executes until the necessary frames have loaded. See page 48 for more about `gotoAndPlay`.

You can also use the `_framesloaded` property with an `if` statement to make sure the download of movie contents to the local computer has completed before you trigger an ActionScript statement or series of statements.

CHECK FRAME LOAD





1 Click the frame, button, or movie clip to which you want to add ActionScript.

■ This example uses a frame.

2 Click Window ➪ Actions to open the Actions panel.

3 Click Basic Actions to open the Basic Actions category.

4 Double-click If Frame Is Loaded to select the `ifFrameLoaded` action.

5 Click ▾ to select a scene.

6 Click ▾ to select a frame type.

7 Click ▾ to select a frame.

Extra

You can create a preloader using the script shown here.

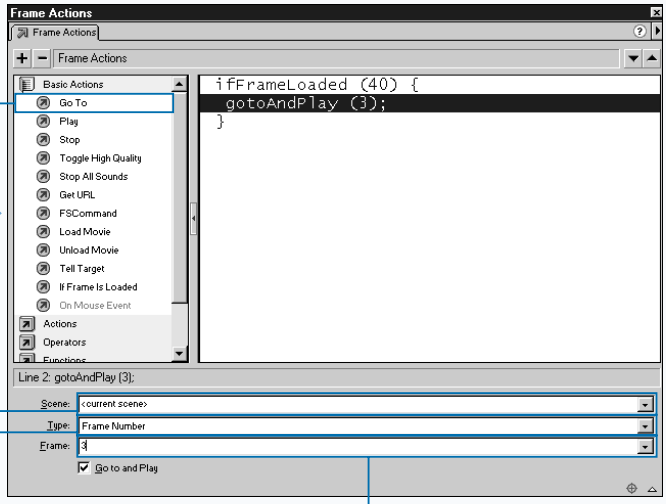
Frame 1

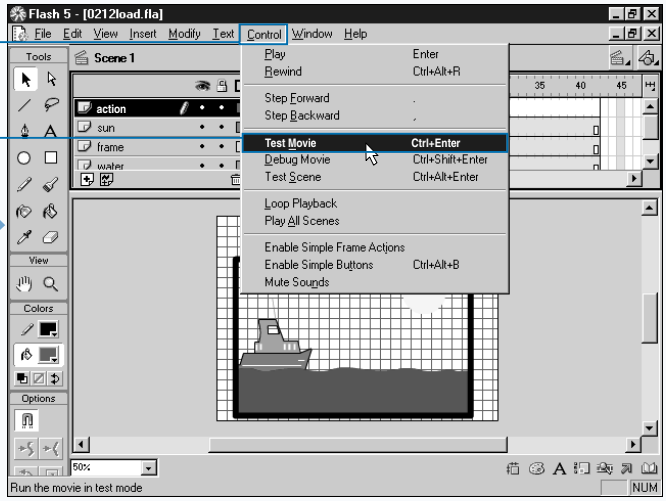
```
ifFrameLoaded(60)
  gotoAndPlay(3);
```

Frame 2

```
gotoAndPlay(1);
```

Place your preloader in frame 1. Each time the playhead enters frame 1, it checks to see if frame 60 is loaded. If frame 60 is loaded, Flash goes to frame 3 and begins playing the movie; otherwise, Flash goes to frame 2, which instructs it to loop back to frame 1.





8 Double-click Go To.

9 Click ▾ to select a scene.

10 Click ▾ to select a frame type.

11 Click ▾ to select a frame.

■ Your movie plays after the specified number of frames have loaded.

12 Click Control ➪ Test Movie to move to the test environment.

13 Test your movie.

Note: See page 38 for information on testing your movie.

INTRODUCTION TO MOVIE CLIP PROPERTIES

In Flash, the attributes of an object are called properties. Movie clips have a large number of attributes — or properties, such as height, width, location, and visibility. Using ActionScript, you can obtain the current value of a movie clip property and you can change the value of many properties. Some properties are read-only, which means that you can retrieve the value but you cannot change it.

You retrieve and change movie clip property values for a variety of reasons. Retrieving the `_x` and `_y` properties tells you the location of a movie clip. Retrieving the height and width properties tells you the size of a movie clip. Changing these properties enables you to adjust the size or move a movie clip.

To retrieve or set the property value of a movie clip, you must give the movie clip instance a name. You use the Instance panel to name movie clips instances. In the Action panel, the Toolbox list lists movie clip properties under Properties. The basic syntax for retrieving a property is to type the instance name followed by a dot and the property. The basic syntax for setting the value of a property is the instance name followed by a dot, the property, an equal sign, and the value to which you want to set the property.

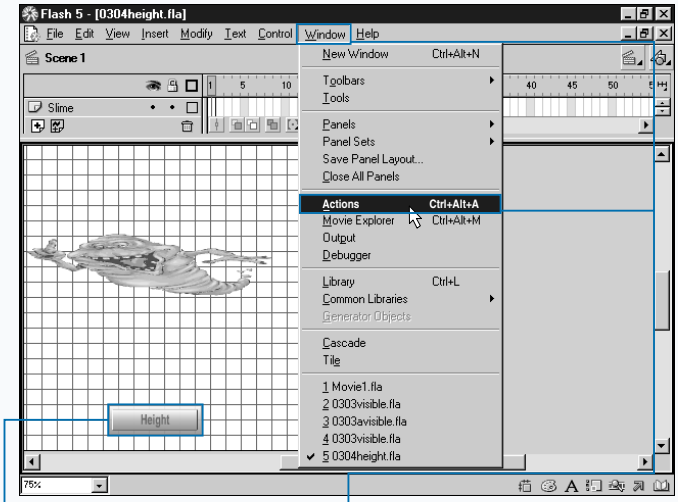
You can also use the `setProperty` action to set movie clip properties. The syntax for the `setProperty` action is

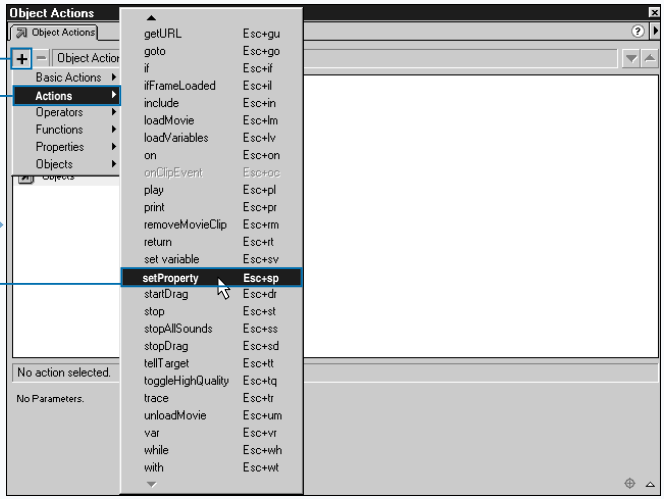
```
setProperty(target, property, value) .
```

Use the target argument to specify the instance name of the movie clip you want to target. Use the property argument to specify the property you want to set. Use the value argument to specify the value to which you want to set the property.

You can use the `getProperty` function to retrieve the value of properties. For more information on functions, see Chapter 9.

INTRODUCTION TO MOVIE CLIP PROPERTIES





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file setprop.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Click ⊞ ➪ Actions ➪ setProperty.

Extra

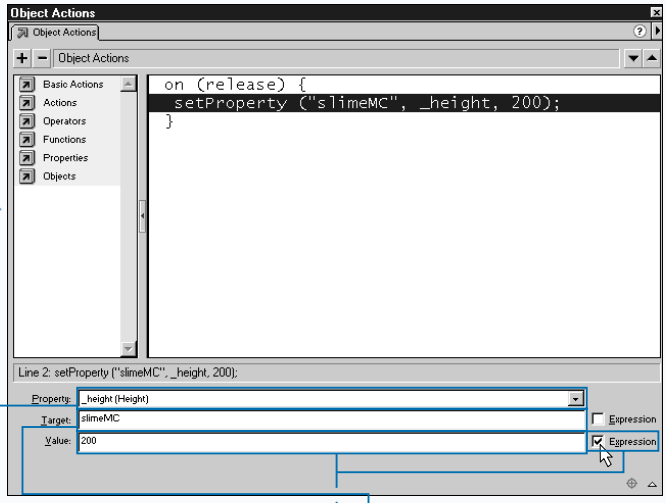
You can use the statement `instanceName._property = instanceName._property + value` to continuously update the value of a property. Here is how it works. You use = to assign a value to a property. The statement `instanceName.property` retrieves the current value of a property. You use the + to add to a value. For example, if the current value of the rotation property is 90, you can use the following to set the value to 105:

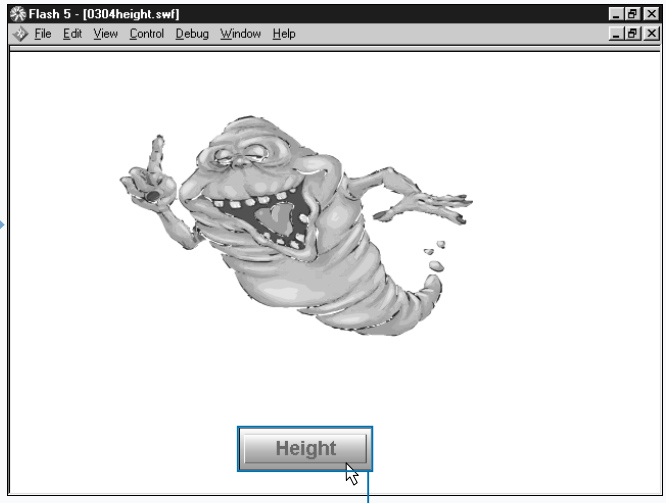
```
sampleMC._rotation = sampleMC._roation + 15
```

sampleMC._rotation	=	sampleMC._rotation	+	15
105	=	90	+	15

If you assign the above statement to a button, each click changes the rotation property value. So, after the first click the equation reads:

sampleMC._rotation	=	sampleMC._rotation	+	15
120		105	+	15





■ The default handler `on (release)` appears in the Action list.

■ The `setProperty` action appears in the Action list.

4 Click ▾ to select the `_height` property.

5 Type the movie clip instance name.

6 Type the height to which you want to set the movie clip.

7 Click to select Expression (☐ changes to ☑).

8 Move to the test environment.

Note: See page 38 for instructions on how to test a movie.

9 Click the button to test your movie.

■ The height of the movie changes to the number of pixels you specified.

NAME AN INSTANCE

To change the properties, or target a movie clip, the movie clip instance must have a unique name. Each time you drag a movie clip from the Library onto the Stage, you create a new instance of that movie clip. Naming movie clips instances enables you to distinguish each instance. Once you have named a movie clip instance, you can use ActionScript to manipulate instances independently of the other instances. You can move some instances while other instances remain motionless. You can change the color of some instances, while the other instances are unaffected.

You use the Instance panel to name instances of your movie clips. When you select movie clip as the behavior for a symbol, the instance panel includes a field that enables you to name the instance.

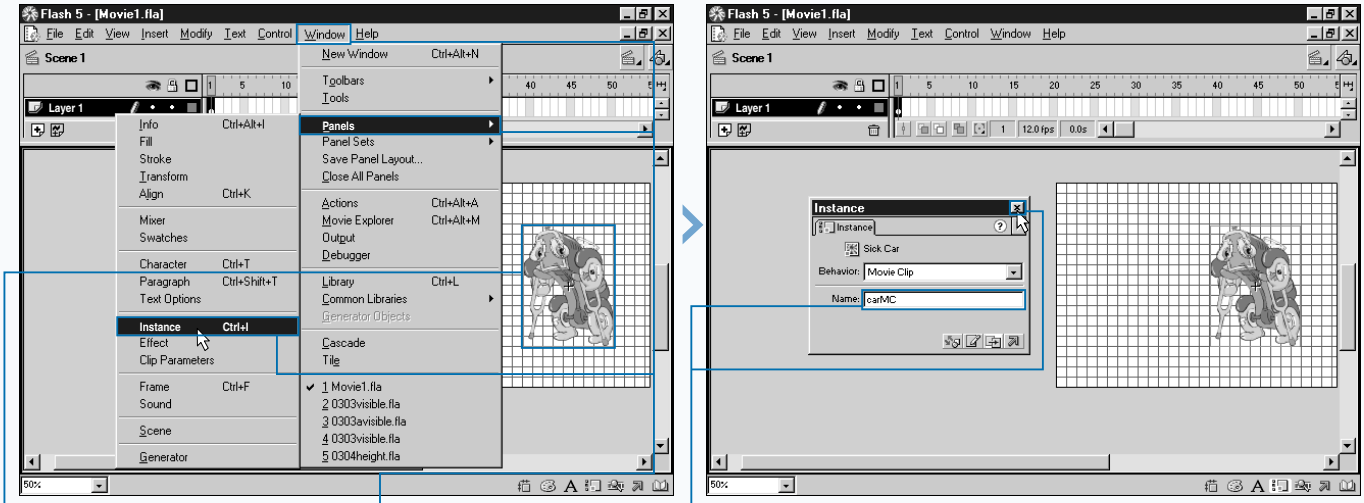
If you do not name a movie clip instance, Flash assigns the instance the name instance followed by a sequential number. You cannot view assigned names in the Instance panel. However, assigned names are viewable in Debugger. For more information on the Debugger, see Chapter 13. Do not use the name assigned to the instance by Flash in your script. Use the Instance panel to assign a name to your movie clip and use that name in your script.

You can use the `_name` property to rename a movie clip instance. The syntax for renaming a movie clip instance is

```
instanceName._name = newName; .
```

The `instance name` argument is used to specify the instance for which you want to change the name. Use the `newName` argument to specify the new name. The following changes the name of a movie clip from `sampleMC` to `yourMC`: `sampleMC._name = yourMC; .`

NAME AN INSTANCE



1 Select the movie clip instance you want to name.

2 Click Window ➪ Panels ➪ Instance to open the Instance panel.

3 Type the instance name.

4 Click ☒ to close the Instance panel.

■ You have named the movie clip instance.

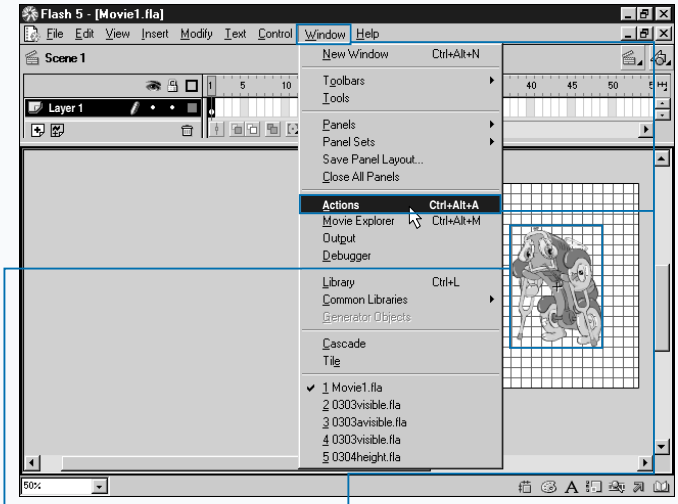
Extra

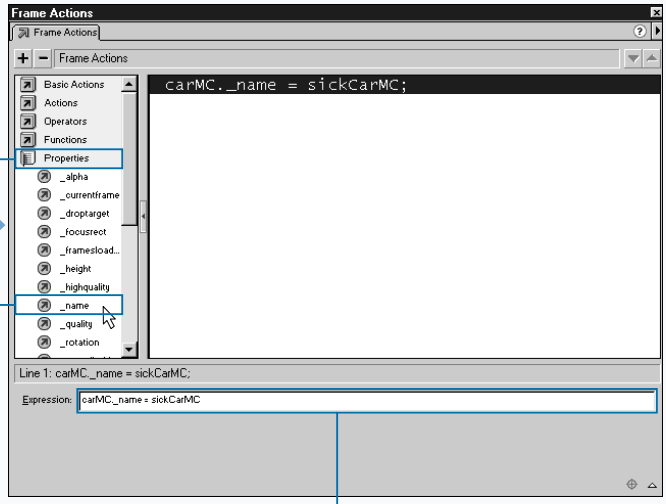
Naming a movie clip instance enables you to target the instance. You must target a movie clip instance when you want to perform actions on a specific instance. For example, if you want to make a movie clip instance named `aMC` play, you must target movie clip instance `aMC`.

You can name a movie clip instance any thing you want. You can use underscores, letters, and numbers to name a movie clip instance. However, you should start the instance name with a letter. Do not include spaces, periods, or other characters that have special meaning to Flash. Do not use a keyword to name an instance. For a list of keywords, see Chapter 14. Do not give two objects the same name. For example, do not name a variable and an instance the same thing.

It is a good idea to develop a naming convention and to stick with it. A popular convention is to capitalize the first letter of all words after the first word, for example, `birthDate` or `firstName`. Try to use meaningful names. Names that describe the instance are a good choice.

RENAME AN INSTANCE





1 Select the frame, button, or movie clip to which you want to add ActionScript.

■ This example uses a frame.

2 Click Windows ➪ Actions to open the Actions panel.

3 Click Properties to open the Properties category.

4 Double-click _name.

■ The `_name` property appears in the Actions list.

5 Rename the movie clip.

■ The movie clip is renamed.

ADJUST TRANSPARENCY

You can use the `_alpha` property to set the transparency of a movie or movie clip instance. Changing the `_alpha` property allows you to create movies and movie clips the user can see through. You can set the `_alpha` property so that movies or movie clips are anything from opaque to invisible. Setting the property to 100 makes the movie or movie clip opaque. Setting the property to 0 makes the movie or movie clip completely transparent or invisible.

Before you can change the `_alpha` property of a movie clip instance, the movie clip instance must have a name. Use the Instance panel to name your movie clip instance. The syntax for the `_alpha` property is

```
instanceName._alpha = value;.
```

The `instanceName` argument is used to specify the name of the instance for which you want to set the `_alpha` property. Omit the `instanceName` argument if you want to set the `_alpha` property for the movie. Use the `value` argument to set the `_alpha` property to a value between 0 and 100.

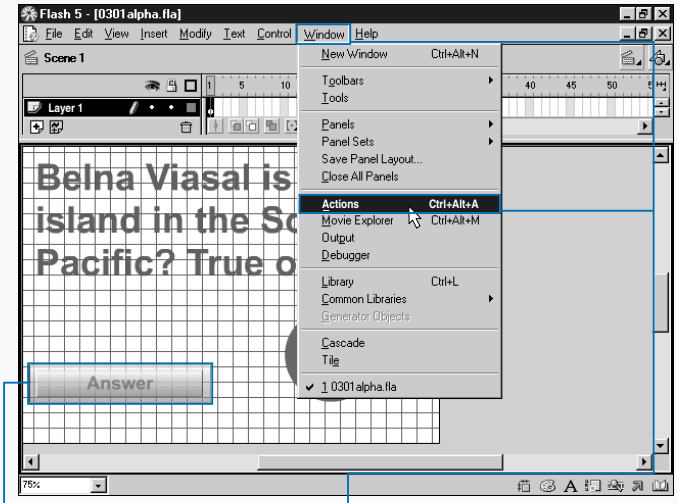
A transparent movie clip is still active. If you set the transparency of a movie clip instance associated with a button's up, down, and over state to 0, the user will not be able to see the button, but the user will be able to click the button.

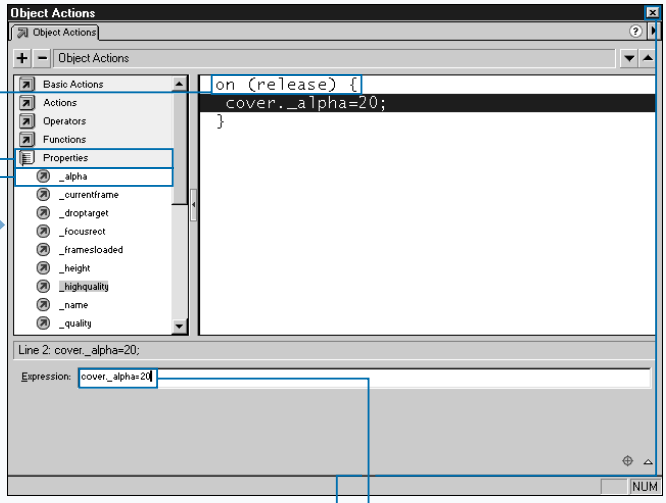
You can retrieve the `_alpha` property of a movie clip. You retrieve the `_alpha` property to determine the current value. The syntax for retrieving the `_alpha` property is

```
instanceName._alpha;.
```

Use the `instanceName` argument to specify the instance for which you want to retrieve the `_alpha` value. You may want to retrieve the `_alpha` value to assign it to a variable or you may want to retrieve the `_alpha` value so that you use it in an `if` statement. For more information on `if` statements, see Chapter 6.

ADJUST TRANSPARENCY





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file alpha.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Click Properties to open the Properties category.

4 Double-click _alpha to select the `_alpha` property.

■ The default handler `on (release)` and `_alpha` property appear in the Action list.

5 Set the `_alpha` property.

■ The instance name precedes the property name.

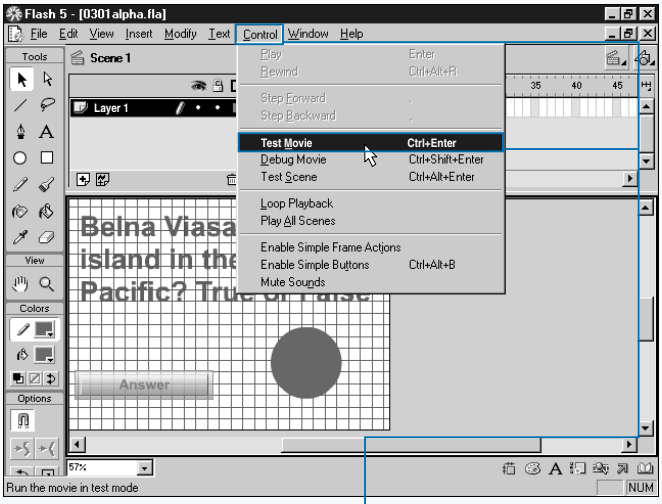
■ The value follows the property name.

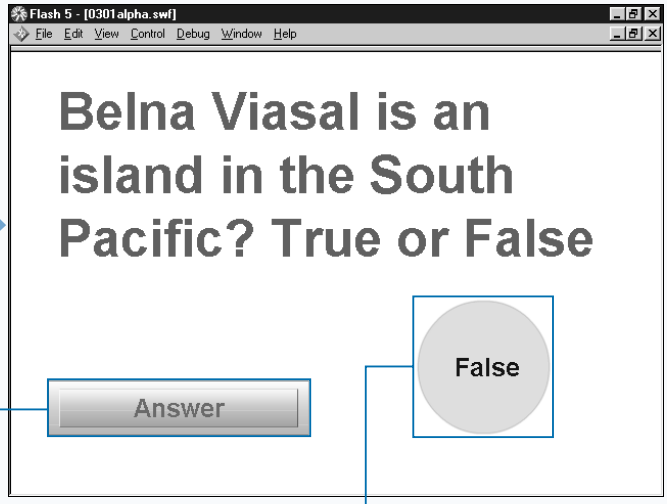
6 Click ☒ to close the Actions panel.

Extra

You can change the `_alpha` property of symbols and instances of symbols using the Effect panel. To open the Effect panel, click Window ➪ Panel ➪ Effect on the menu. Select Alpha from the drop-down menu. You can adjust the `_alpha` property of a an instance of a symbol to any value from 0 to 100. Changing the `_alpha` property of an instance does not change the symbol or other instances of the symbol.

You can use the `_alpha` property in a tween to create a fading effect. Set the `_alpha` property at the start position. Reset the `_alpha` property at the end position. Flash will interpolate the values in between causing the object to fade in or out. You can also change the `_alpha` property of text. However, you must first covert the text to a symbol.





■ The Flash menu is available to you.

7 Click Control ➪ Test Movie to move to the test environment.

Note: See page 38 for instructions on how to test a movie.

8 Click the button to test a movie.

■ When you click the button, the transparency of the movie clip changes.

MAKE MOVIE CLIPS INVISIBLE

You can use the `_visible` property to make a movie or movie clips visible or invisible to the user. This property is useful when you have a movie clip that you do not want the user to see until the movie reaches a particular frame, or a particular action has occurred.

Before you can change the `_visible` property of a movie clip instance, the instance must have a name. You use the Instance panel to name a movie clip instance. The syntax for the `_visible` property is

```
instanceName._visible = BooleanValue; .
```

The `instanceName` argument is used to specify the instance name for which you want to change the `_visible` property. Use the `BooleanValue` argument to set the `_visible` property to `true` or `false`. If you set the value to `true`, the movie clip is visible. If you set the value to false, the movie clip is not visible.

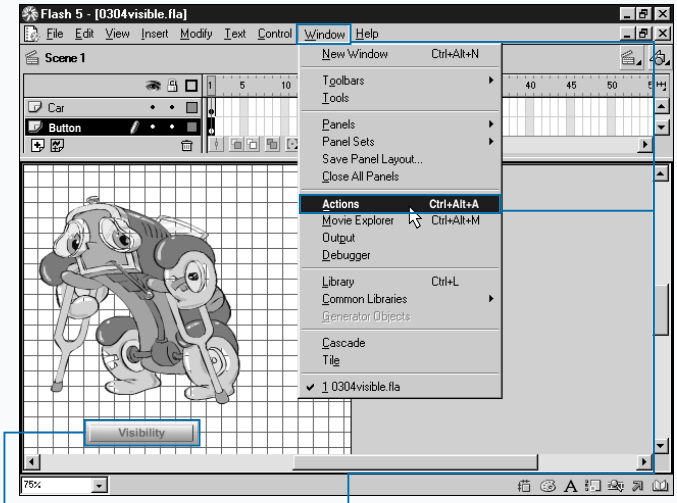
A movie clip instance that is not visible is not active. The user cannot interact with a movie clip that is not active. If you set the visibility of a movie clip associated with a button's up, down, and over state to false, users will not be able to see or click the button.

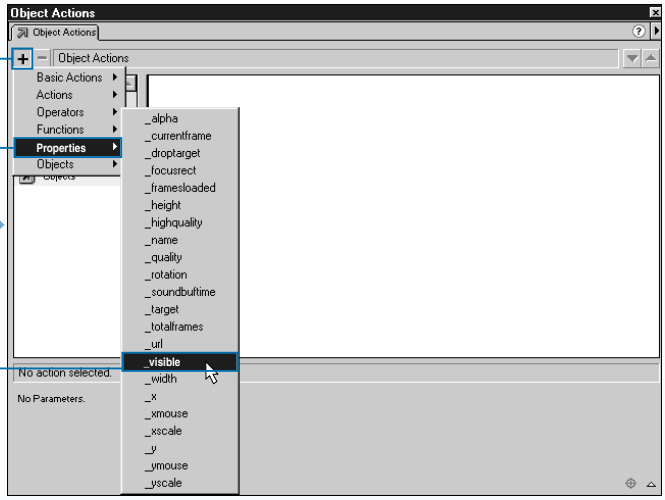
You can retrieve the `_visible` property value. When writing script, you may want to retrieve the `_visible` property value of a movie clip to determine what action ActionScript should perform. For example, you can create a button that toggles the visibility of a movie clip on and off. If the `_visibility` value is `true`, the script will set the visibility to `false`. If the `_visibility` value is `false`, the script will set the `_visibility` to `true`. The syntax for retrieving the `_visible` value is

```
instanceName._visible; .
```

Use the `instanceName` argument to specify the name of the instance for which you want to change the `_visible` property.

MAKE MOVIE CLIPS INVISIBLE





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file visible.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Click ⊞ ➪ Properties ➪ _visible to select the `_visible` property.

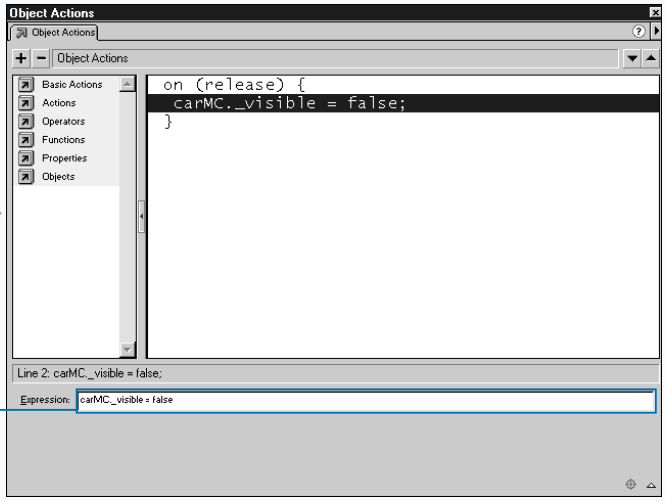
Apply It

You can use the `getProperty` function to retrieve a property. For more information on the `getProperty` function, see page 210. You can use the `setProperty` action to set a property. For more information on the `setProperty` action, see page 64.

The script shown here is associated with a button. When the user releases the button, the script retrieves the `_visible` property for `starMC`. If the `_visible` property is true, the script sets the `_visible` property to false; if the `_visible` property is false, the script sets the `_visible` property to true. In other words, the button toggles `starMC` on and off.

Example:

```
on (release) {
    x = getProperty ( starMC, _visible );
    if (x  == true) {
        setProperty ("StarMC", _visible, false);
    } else {
        setProperty ("StarMC", _visible, true);
    }
}
```



- The default handler `on (release)` appears in the Action list.
- The `_visible` property appears in the Action list.

4 Set the `_visible` property.

- The instance name precedes the property name.
- The value follows the property name.

5 Move to the test environment.

Note: See page 38 for instructions on how to test a movie.

6 Click the button to test your movie.

- When you click the button, the movie clip will no longer be visible.

ROTATE MOVIE CLIPS

You use the `_rotation` property to rotate a movie or movie clip instance. The rotation value is the number of degrees a movie clip has been rotated from the original position at which it was placed on the Stage.

Before you can set the value of the `_rotation` property of a movie clip instance, the movie clip instance must have a name. Use the Instance panel to name your movie clip instance. The syntax for setting the value of the `_rotation` property is

```
instanceName._rotation = value; .
```

The `instanceName` argument is used to specify the Instance name of the movie clip for which you want to set the value of the `_rotation` property. Omit the `instanceName` to set the rotation value for the movie. Use the value argument to set the value of the `_rotation` property.

Note that setting the rotation value to 90 rotates the movie clip instance 90 degrees from its original location, not its current location. If the current rotation value is 90 and you set the rotation value to 90, the movie clip will not rotate. To rotate the movie clip an additional 90 degrees, you must set the rotation value to 180.

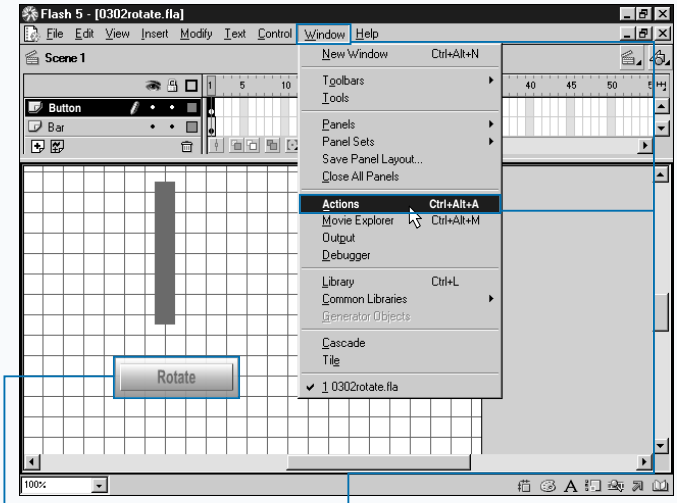
You can retrieve the current `_rotation` value. The syntax for retrieving the rotation value is

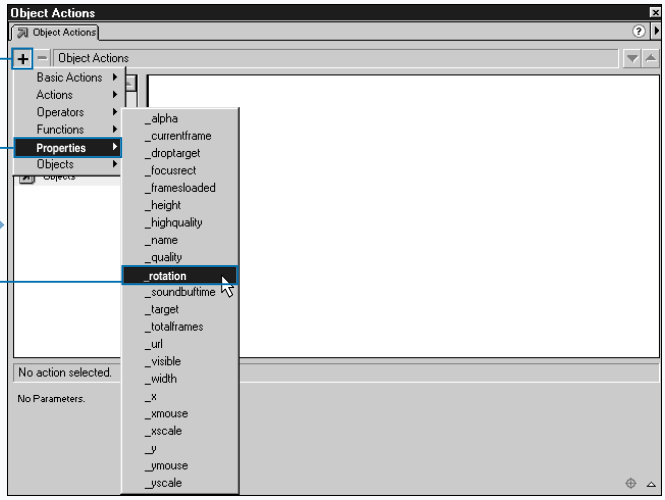
```
instanceName. _rotation; .
```

Use the `instanceName` argument to specify the instance name of the movie clip you want to rotate. Omit the `instanceName` argument to retrieve the rotation value of the movie.

If you want to rotate a movie clip instance a specified number of degrees from its current location, you can retrieve the current value and add the number of degrees you want to rotate the movie clip instance. For example, `sampleMC._rotation = sampleMC._rotation + 90;` rotates a movie clip 90 degrees from its current location.

ROTATE MOVIE CLIPS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file rotate.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

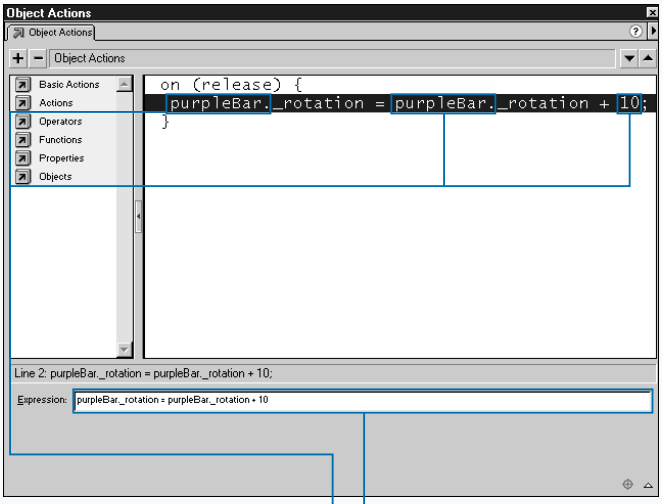
2 Click Window ➪ Actions to open the Actions panel.

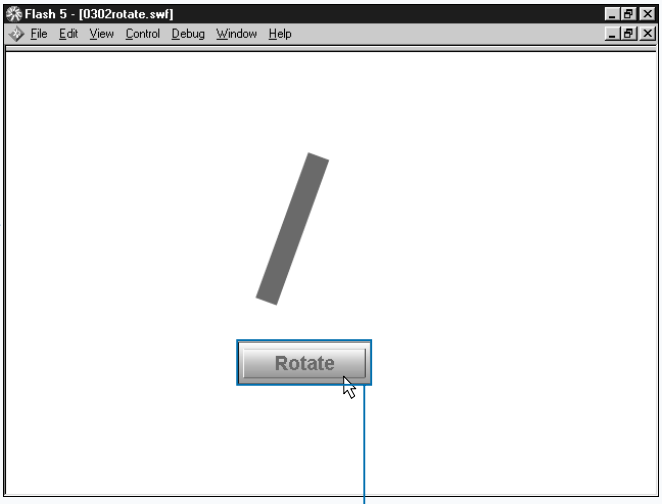
3 Click ⊞ ➪ Properties ➪ _rotation.

Extra

You can use the Flash menu to rotate an object. Select an object, then click Modify ➪ Transform ➪ Rotate. Handles will appear around the object. Use the mouse to grab the handles then drag to rotate. Select an object and click Modify ➪ Transform ➪ Rotate 90 CW to rotate an object 90 degrees clockwise. Select an object and click Modify ➪ Transform ➪ Rotate 90 CCW to rotate an object 90 degrees counterclockwise. You can also use the Arrow tool rotate modifier to rotate objects.

Every symbol and instance of a symbol has a registration point. By default, the registration point is the center of the object. Objects rotate around their registration point. To change the registration point, select the object, click Modify ➪ Transform ➪ Edit Center from the menu, and drag the registration point to a new location.





■ The default handler `on (release)` appears in the Action list.

Note: See page 28 for more information on handlers.

■ The `_rotation` property appears in the Action list.

4 Set the `_rotation` property.

■ The instance name precedes the property name.

■ The value follows the property name.

5 Move to the test environment.

Note: See page 38 for instructions on how to test a movie.

6 Click the button to test your movie.

■ The movie rotates 10 degrees each time you click the button.

CHANGE THE WIDTH OF MOVIE CLIPS

You use the `_width` property to set the distance from the left side of a movie or movie clip to the right side in pixels. You can place an input field on the Stage that allows the user to specify the size of a movie clip. When the user types a number in the input field, the number is assigned to the variable. That variable can be used to determine the width of the movie clip. For more information on variables, see pages 88 to 95.

Before you can change the `_width` property of a movie clip, the movie clip must have a name. You use the Instance panel to name a movie clip. The syntax for the `_width` property is

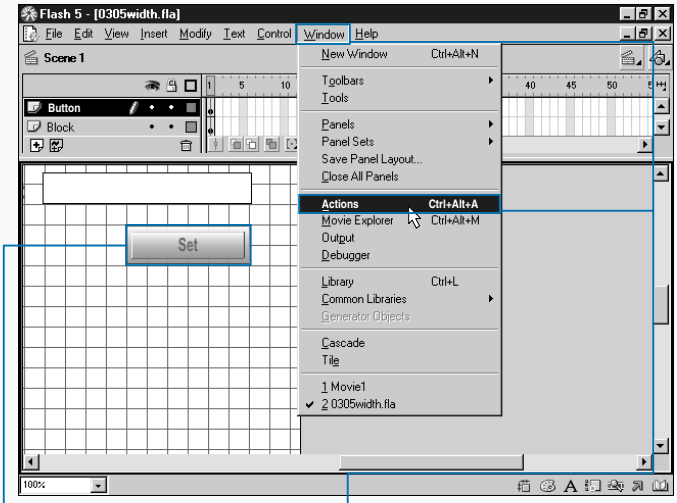
```
instanceName._width = value .
```

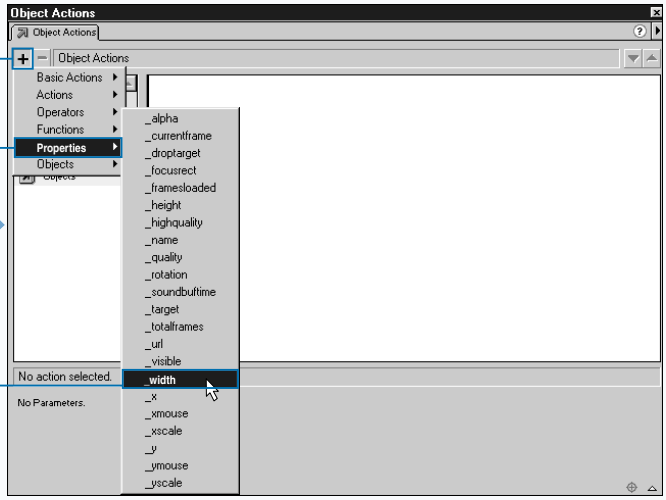
The `instanceName` argument is used to specify the name of the movie clip for which you want to change the `_width` property. Omit the `instanceName` property to set the width of a movie. Use the value argument to set the value of the `_width` property in pixels.

You can retrieve the `_width` property to obtain its current value. The syntax for retrieving the `_width` property is `instanceName._width`. Use the `instanceName` argument to specify the movie clip instance for which you want to retrieve the `_width` property. Omit the Instance name if you want to retrieve the width of a movie clip.

As with all properties, remember, you can use `setProperty` action to set the `_width` property value. You can use the `getProperty` function to retrieve the current property value. For more information on the `setProperty` action, see page 64. For more on the `getProperty` function, see page 210.

CHANGE THE WIDTH OF MOVIE CLIPS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file width.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

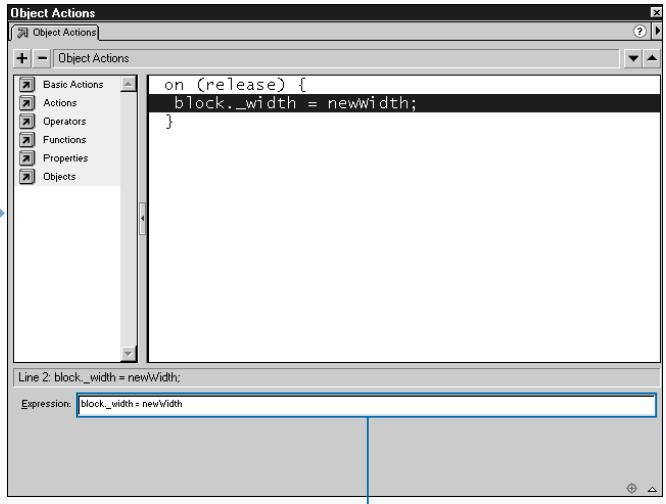
3 Click ⊞ ➪ Properties ➪ _width.

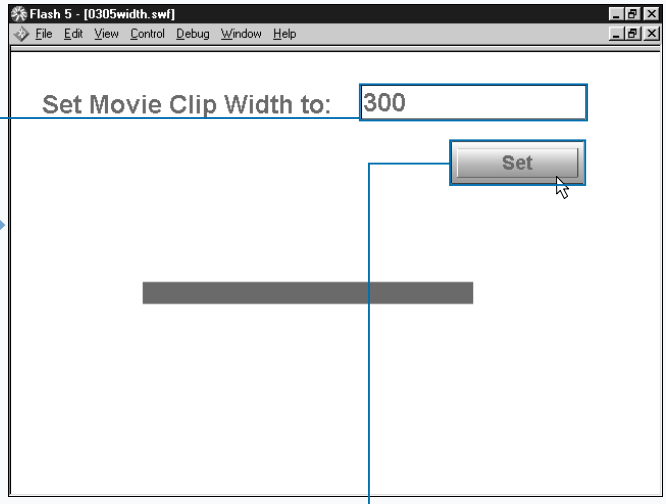
Extra

You can use this script to manipulate the width of a movie clip. This script increases the width of movie clip named block 10 pixels each time the user presses the left arrow. It decreases the width 10 pixels each time the user presses the right arrow. The script is associated with a button.

Example:

```
on (keyPress "<Left>") {
    widthV= getProperty ("block", _width );
    setProperty ("block", _width, widthV + 10);
}
on (keyPress "<Right>") {
    widthV= getProperty ("block", _width );
    setProperty ("block", _width, widthV - 10);
}
```





■ The default handler `on (release)` appears in the Action list.

Note: See page 28 for more information on handlers.

■ The `_width` action appears in the Action list.

4 Set the width of the movie clip instance.

■ If you associate the variable with an input text box, you can obtain data from the user.

Note: See page 22 for more on input text boxes, and pages 88 to 95 for more on variables.

5 Move to the test environment.

Note: See page 38 for instructions on how to test a movie.

6 Type the number of pixels to which you want to set the width of the movie clip.

7 Click the button to test your movie.

■ The width of the movie clip is set to the value entered in the input text box.

CHANGE THE HEIGHT OF MOVIE CLIPS

You can use the `_height` property to set the distance from the bottom of a movie or movie clip to the top in pixels. This property is useful if you need to resize a movie or movie clip as the movie plays. You can increase or decrease the height of a movie or movie clip. A possible use of the `_height` property is to increase or decrease the size of a bar as the user increases or decreases the sound volume. Alternatively, you may want to use the `_height` property to give the user the ability to set the size of objects on the Stage.

Before you can change the `_height` property of a movie clip instance, the movie clip instance must have a name. You use the Instance panel to name a movie clip instance. The syntax for the `_height` property is

```
instanceName._height = value;.
```

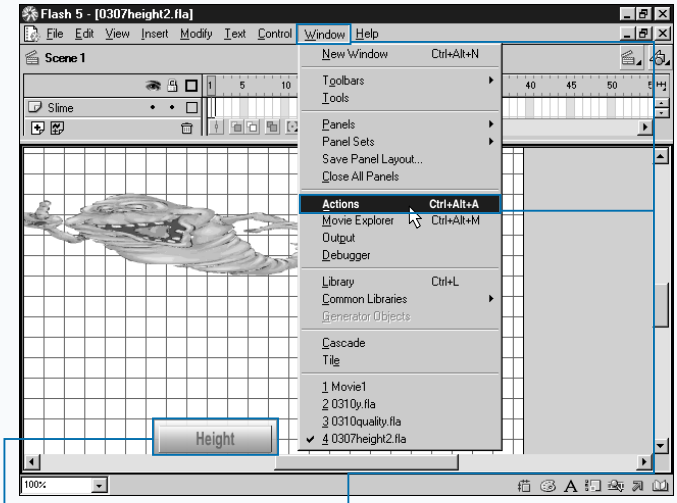
The `instanceName` argument is used to specify the name of the movie clip instance for which you want to change the `_height` property. To change the height of the movie, omit the `InstanceName` argument. Use the value argument to set the value of the `_height` property of the movie or movie clip in pixels.

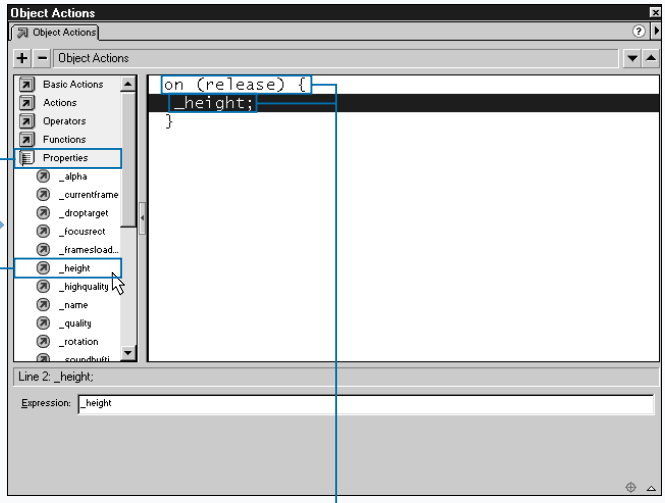
ActionScript also gives you the ability to retrieve the current value of the `_height` property. As with other properties, you can retrieve the `_height` property and use the value to determine what action to perform next. The syntax for retrieving the `_height` property is

```
instanceName. _height;.
```

Use the `instanceName` argument to specify the name if the movie clip for which you want to retrieve the `_height` property. Omit the `instanceName` argument to retrieve the height of the movie.

CHANGE THE HEIGHT OF MOVIE CLIPS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file height2.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Click Properties to open the Properties category.

4 Double-click _height to select the `_height` property.

■ The default handler `on (release)` appears in the Action list.

Note: See page 28 for more information on handlers.

■ The `_height` property appears in the Action list.

Apply It

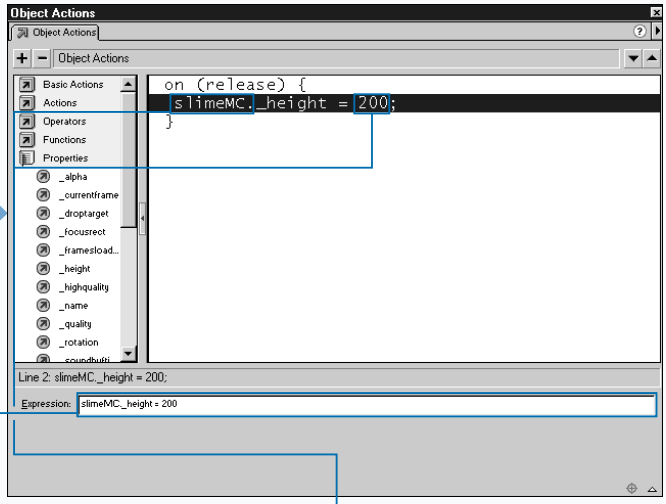
You can use this script to increase the height of a movie clip each time the user clips on a button. The script retrieves the height of a movie clip named block and assigns the value to a variable named `heightV`. It then sets the height of the block movie clip to the value of `heightV plus 10`. You can use the `setProperty` action to set the height of a movie clip. For more information on the `setProperty` action, see page 64,

TYPE THIS:

```
on (release) {
    heightV = block._height;
    setProperty ("block", _height, heightV + 10);
}
```

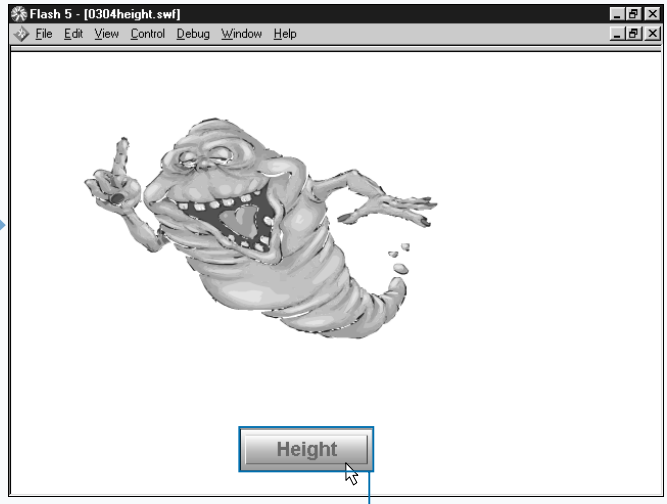
RESULT:

Each time the user clicks a button, the height of the movie clip increases by 10 pixels.



5 Set the `_height` property for a movie clip instance.

- The instance name precedes the property name.
- The value follows the property name.



6 Move to the test environment.

Note: See page 38 for instructions on how to test a movie.

7 Click the button to test your movie.

- The size of the movie clip instance changes to the height you set.

SCALE THE WIDTH OF MOVIE CLIPS

You can use the `_xscale` property to change the width of a movie or movie clip using a percentage. This property is similar to the `_width` property. The `_width` property changes the distance from the left to the right side of a move clip using pixels, while `_xscale` uses a percentage.

Before you can change the `_xscale` property of a movie clip instance, the movie clip instance must have a name. You use the Instance panel to name a movie clip instance. The syntax for the `_xscale` property is

```
instanceName._xscale = value .
```

The `instanceName` argument is used to specify the name of the movie clip instance. Omit the instance name argument to set the `_xscale` property of a movie. Use the value argument to specify the percent by which you want to change the width of the movie clip instance.

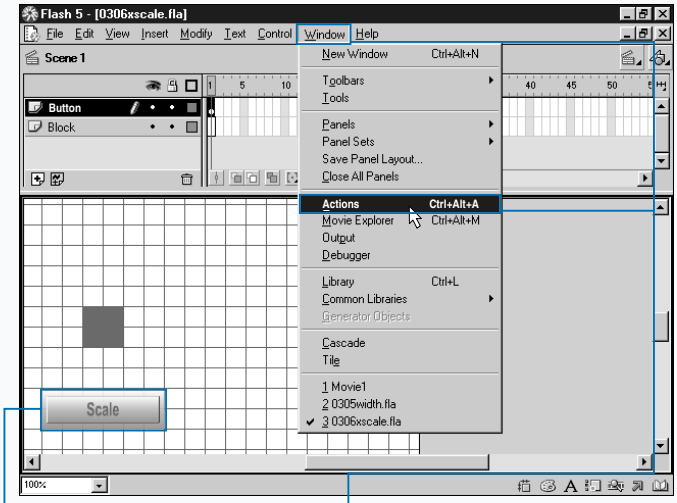
If you want to reduce the width of the movie clip instance to 50 percent of its size before any scaling, assign a value of 50. If you want to increase the width of the movie clip to 100 percent more than its size before any scaling, assign a value of 200. To return a movie clip to its original size, assign a value of 100. Assigning a value of zero will cause your movie or movie clip to disappear.

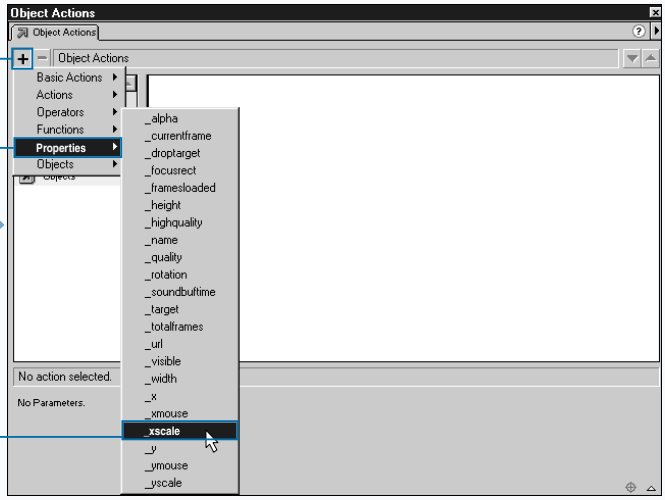
You can also retrieve the current `_xscale` value for a movie or movie clip. The syntax for retrieving the current value is

```
instanceName.xscale .
```

Use the `instanceName` argument to specify the instance name of the movie clip for which you want to obtain the `_xscale`. Omit the `instanceName` argument if you are retrieving the `_xscale` property of a movie.

SCALE THE WIDTH OF MOVIE CLIPS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file xscale.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

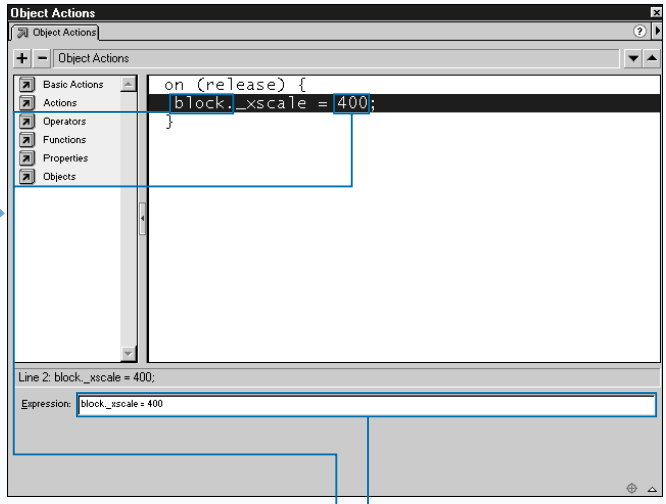
2 Click Window ➪ Actions to open the Actions panel.

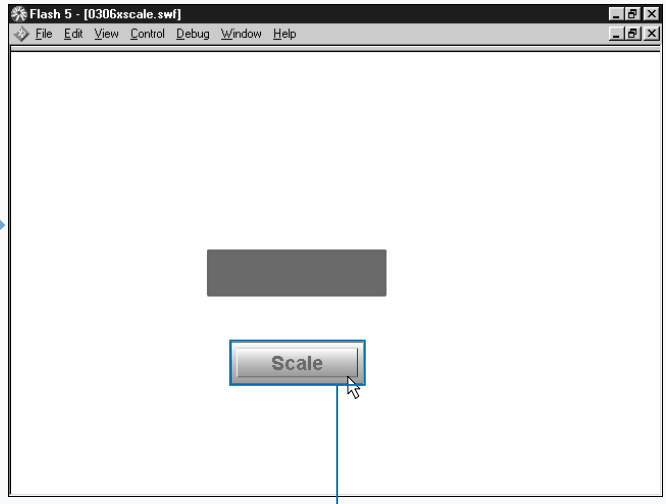
3 Click ⊞ ➪ Properties ➪ _xscale to select the `_xscale` property.

Extra

You can use the Flash menu to scale an object. Select the object and click Modify ➪ Transform ➪ Scale. Handles will appear around the object. Use the mouse to grab the handles then drag to scale. You can use the Flash menu to scale an object by a percentage. Click Modify ➪ Transform ➪ Scale and Rotate and type the scale percentage in the Scale field. You can also use the Arrow tool scale modifier to scale objects.

You can scale an object in a tween to cause the object to appear to grow or shrink over time. Set the size of the object at the start position. Reset the size of the object at the end position. Flash will interpolate the values in between causing the object to shrink or grow.





■ The default handler `on (release)` appears in the Action list.

Note: See page 28 for more information on handlers.

■ The `_xscale` property appears in the Action list.

4 Set the `_xscale` property for the movie clip instance.

■ The instance name precedes the property name.

■ The value follows the property name.

5 Move to the test environment.

Note: See page 38 for instructions on how to test a movie.

6 Click the button to test your movie.

■ The width of the movie clip will increase to the `_xscale` you specified.

SCALE THE HEIGHT OF MOVIE CLIPS

The `_yscale` property is the counter-part of the `_xscale` property. You use the `_yscale` property to change the height of a movie or movie clip by a percentage. You use the `_xscale` property to change the width of a movie or movie clip by a percentage. You can use `_xscale` and `_yscale` in conjunction to change the size of a movie or movie clip.

Before you can change the `_yscale` property of a movie clip instance, the movie clip instance must have a name. You use the Instance panel to name a movie clip. The syntax for the `_yscale` property is

```
instanceName._yscale = value.
```

The `instanceName` argument is used to specify the name of the move clip instance for which you want to set the `_yscale` property. Omit the instance name to change the height of a movie. Use the value argument to specify the percent by which you want to change the height of the movie clip.

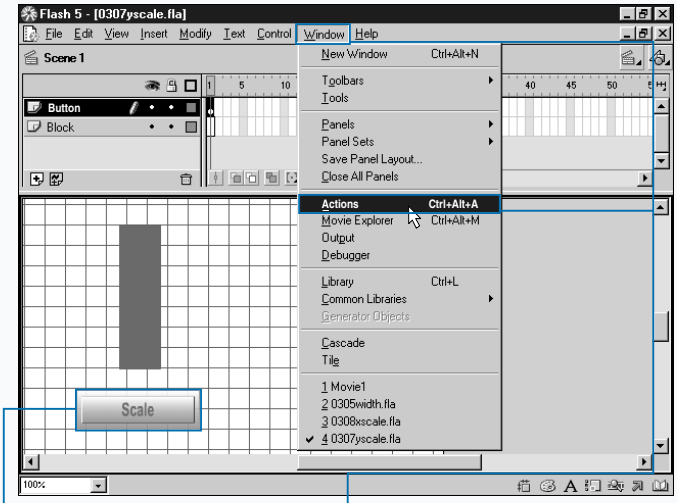
If you want to reduce the height of the movie clip to 50 percent of the size it was before any scaling was applied, assign a value of 50. If you want to increase the height of the movie clip by 100 percent of the size it was before any scaling was applied, assign a value of 200. To return a movie clip to its size before scaling, assign a value of 100. Assigning a value of zero will cause your movie or movie clip to disappear.

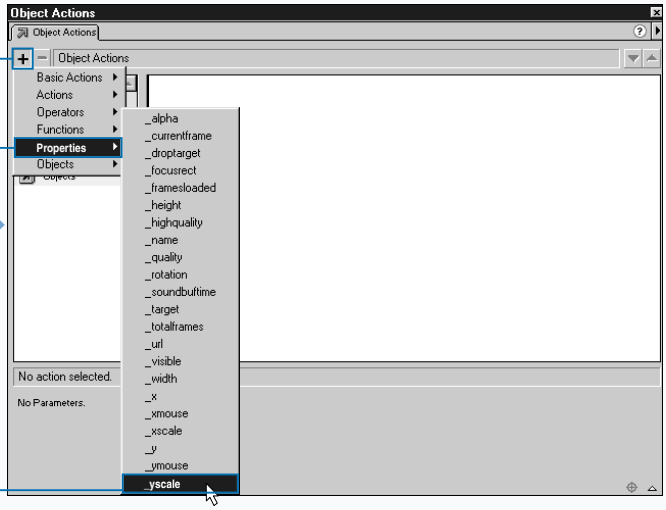
You can also retrieve the current `_yscale` value for a movie or movie clip. The syntax for retrieving the current value is

```
instanceName._yscale.
```

Use the `instanceName` argument to specify the name of the movie clip for which you want to obtain the `_yscale` value. Omit the `instanceName` argument if you are retrieving the `_yscale` property of a movie.

SCALE THE HEIGHT OF MOVIE CLIPS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file yscale.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

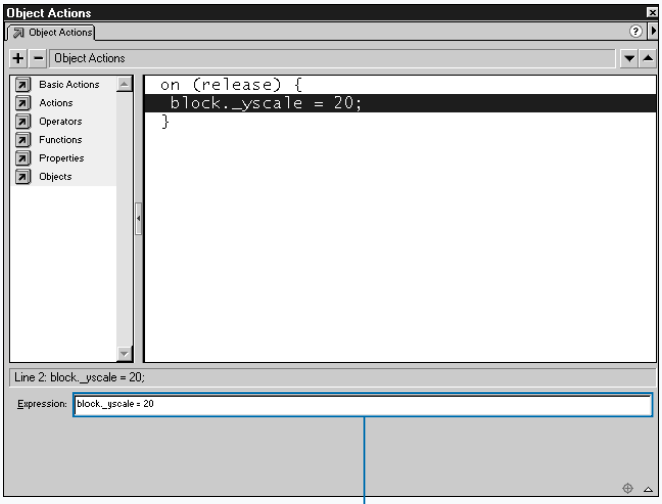
3 Click ⊞ ➪ Properties ➪ _yscale to select the `_yscale` property.

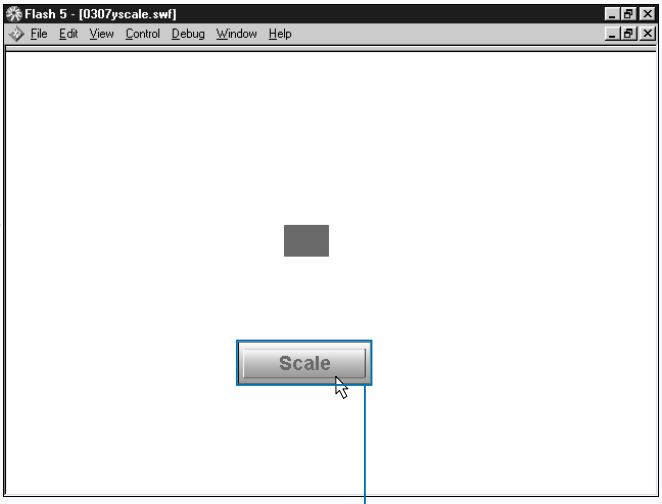
Extra

This script scales a movie clip instance. When the user presses the button, the movie clip named block disappears. When the user releases the button, the movie clip reappears.

Example:

```
on (press) {
    setProperty ("block", _height, 0);
    setProperty ("block", _width, 0);
}
on (release) {
    block._yscale =100;
    block._xscale =100;
}
```





■ The default handler `on (release)` appears in the Action list.

Note: See page 28 for more information on handlers.

■ The `_yscale` property appears in the Action list.

4 Set the `_yscale` property for a movie clip instance.

■ The instance name precedes the property name.

■ The value follows the property name.

5 Move to the test environment.

Note: See page 38 for instructions on how to test a movie.

6 Click the button to test your movie.

■ The movie clip will scale to the size you specified.

MOVE MOVIE CLIPS ACROSS THE STAGE

You can use the _x property to change the location of a movie or movie clip relative to the left side of the Stage. The _x property enables you to move a movie or movie clip back and forth across the Stage.

Before you can change the _x property for a movie clip instance, you must give the movie clip an instance name. Use the Instance panel to name your movie clip. The syntax for the _x property is

```
instanceName._x = value; .
```

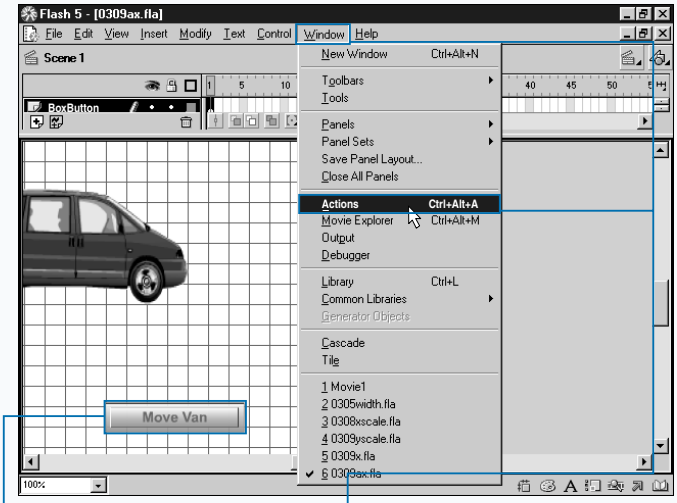
The instanceName argument is used to specify the instance name of the movie clip for which you want to set the _x property. Omit the instance name if you want to set the _x property for the movie. Use the value argument to specify the number of pixels from the left border of the Stage you want to locate the center of the movie or movie clip. The left border of the Stage has a value of 0.

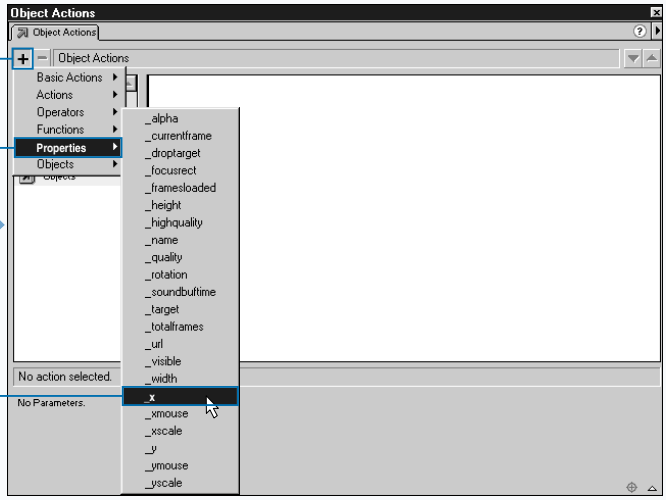
You can retrieve the current value of the _x property. The syntax for retrieving the current value of the _x property is

```
instanceName._x; .
```

Use the `instanceName` argument to specify the name of the instance for which you want to retrieve the _x property. Retrieving the value is useful when you want to reset the value of the _x property relative to its current location. For example, the following moves a movie clip 10 pixels to the right of its current location: `sampleMC._x = sampleMC._x +10;` . You can associate this syntax with a button to move an object 10 pixels each time the user presses a button. You can also associate this syntax with the `onClipEvent (enterFrame)` handler in a single frame movie to create continuous movement.

MOVE MOVIE CLIPS ACROSS THE STAGE





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file x.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Click ⊞ ➪ Properties ➪ _x.

Apply It

You can use the `getProperty` function to retrieve a property. For more information on the `getProperty` function, see page 210. You can use the `setProperty` action to set a property. For more information on the `setProperty` action, see page 64.

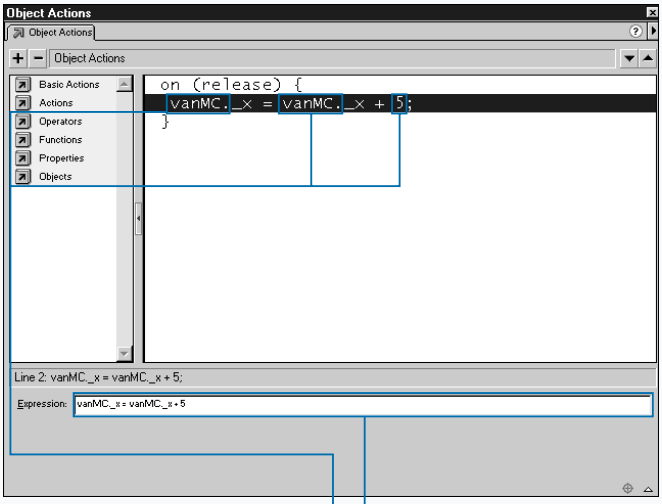
You can use this script to move the movie clip across the Stage. The script is associated with a button. Each time the user clicks the button, a movie clip named car moves 5 pixels to the right.

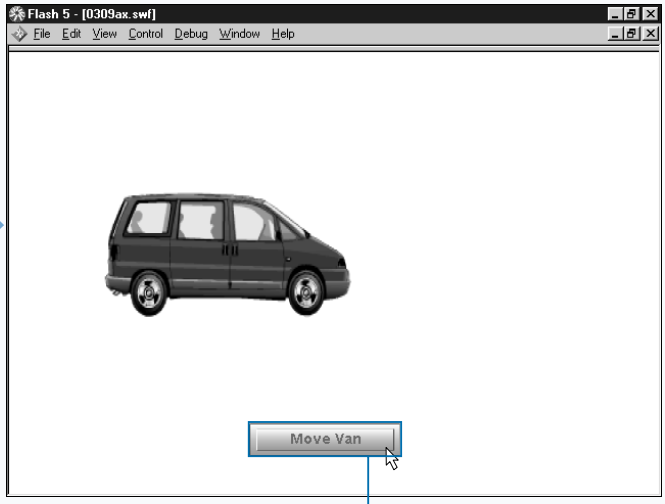
TYPE THIS:

```
on (release) {
    carValue = getProperty ( "car", _x );
    setProperty ("car", _x, carValue + 5);
}
```

RESULT:

Each time the user clicks a button, the movie clip moves 5 pixels to the right.





- The default handler `on (release)` appears in the Action list.

Note: See page 28 for more information on handlers.

- The `_x` property appears in the Action list.

4 Set the `_x` property.

- The instance name precedes the property name.
- The value follows the property name.

5 Move to the test environment.

Note: See page 38 for instructions on how to test a movie.

6 Click the button to test your movie.

- Each time you click the button, the movie clip instance will move the number of pixels you specified.

MOVE MOVIE CLIPS UP AND DOWN

You can use the _y property to change the location of a movie or movie clip relative to the top of the Stage. The _y property enables you to move a movie or movie clip up or down the Stage. Use the _y property in conjunction with the _x property to specify the exact location you want to place the movie or movie clip on the Stage.

Before you can change the _y property of a movie clip instance, you must give the movie clip instance a name. You use the Instance panel to name a movie clip. The syntax for the _y property is

```
instanceName._y = value; .
```

The `instanceName` argument is used to specify the instance name of the movie clip instance for which you want to set the _y property. Omit the `instanceName` argument to set the _y property for the movie. Use the value argument to specify the number of pixels from the top of the Stage you want to place the center point of the movie clip. The top border of the Stage has a value of 0.

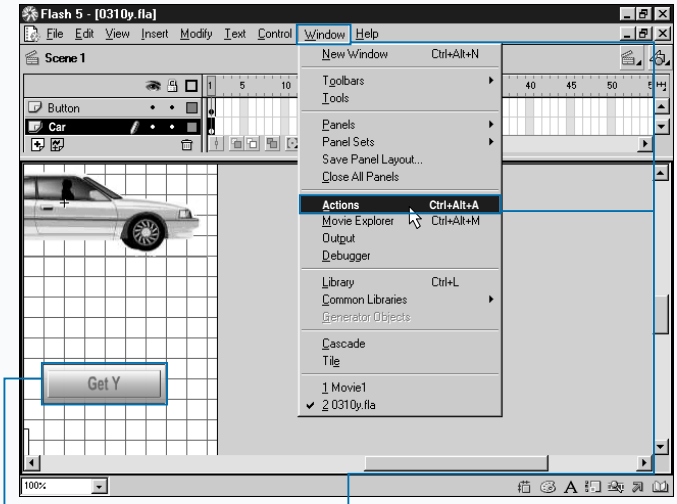
You can retrieve the current value of the _y property. The syntax for retrieving the current value of the _y property is

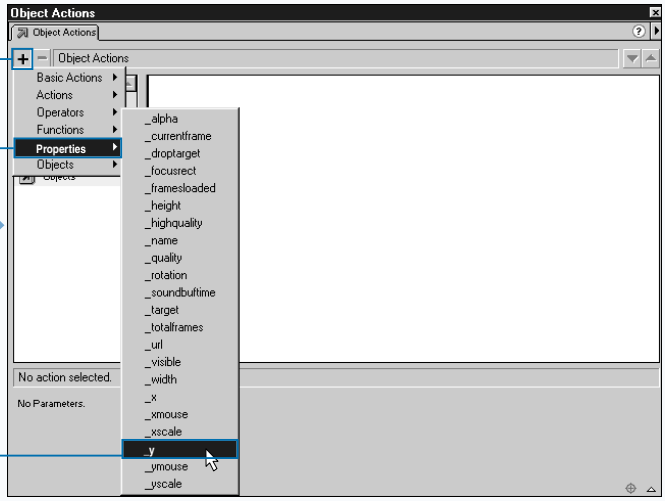
```
instanceName._y .
```

Retrieving the value is useful when you want to reset the value of the _y property relative to its current location.

You will often use the _x and _y properties to move a movie clip around the Stage. You can move a movie anywhere you want by specifying the coordinates. You can use the _x and _y coordinates to create movement.

MOVE MOVIE CLIPS UP AND DOWN





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file y.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

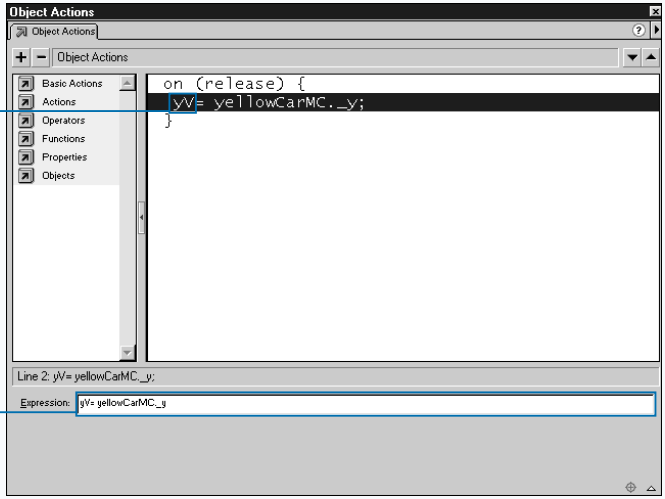
3 Click ⊞ ➪ Properties ➪ _y to select the _y property.

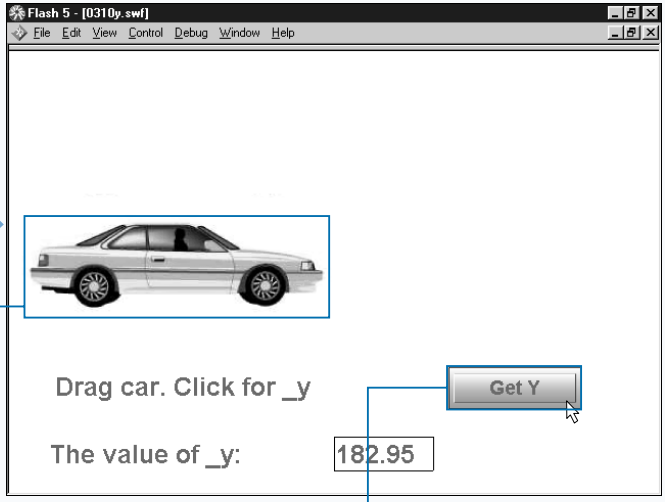
Extra

You can use the script shown here to move a movie clip using key presses. When the user presses the button or the down arrow key, the movie clip named block moves down three pixels. When the user presses the up arrow key, the movie clip named block moves up three pixels.

Example:

```
on (press, keyPress "<Down>") {
    y = getProperty ("block", _y );
    setProperty ("block", _y, y + 3);
}
on (keyPress "<Up>") {
    y = getProperty ("block", _y );
    setProperty ("block", _y, y -3);
}
```





- The default handler `on (release)` appears in the Action list.
- The `_y` property appears in the Action list.

4 Assign the `_y` value to a variable.

- The variable name is `yV`.
- The expression `yellowCarMC._y` retrieves the `_y` value.
- Associate the variable with a dynamic text box. This enables you to display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instructions on how to test a movie.

6 Drag the movie clip.

7 Click the button to test your movie.

- The text box displays the `_y` value associated with the car.

SET MOVIE CLIP QUALITY

Antialiasing is the process of smoothing the edges of images and text. Antialiasing gives Flash the ability to display crisp clear images. Antialiasing also slows down the playback of a movie. You can use the `_quality` property to set the level of antialiasing you want to use in your movie or you can use the `_quality` property to give the user the ability to choose the quality level they want to use as they play back the movie.

The syntax for the `_quality` property is

```
_quality = value; .
```

You use the value argument to specify the quality level. You can choose from LOW, MEDIUM, HIGH, and BEST. You cannot set the quality level for specific elements of a movie. You cannot set the quality for a movie clip. The quality setting is global and applies to the entire movie.

You can also use the `_highquality` property to set the level of antialiasing you want to use in your movie. As with the `_quality` property, you can use the `_highquality` property to give the user the ability to choose the quality level they want to use as they play back the movie.

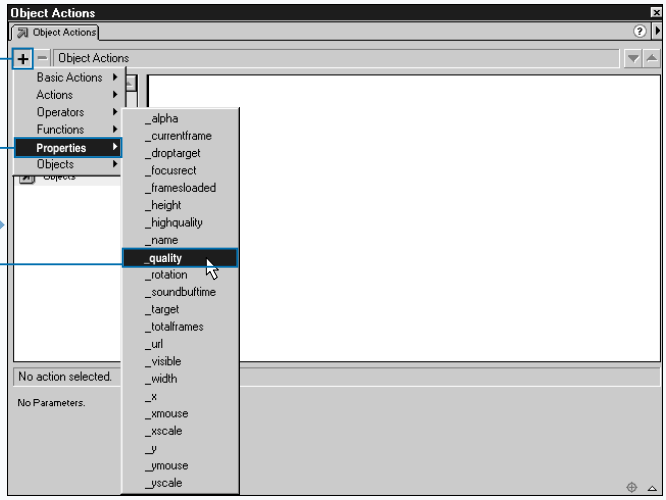
The syntax for the `_highquality` property is

```
_highquality = value; .
```

You can use the value argument to specify the quality to which you want to set your movie clip. Choose from 2 - best, 1 - high quality, or 0 - low quality. Like the `_quality` property, the `_highquality` property is global and applies to all elements of the movie. You cannot use the `_highquality` property to set the quality of a movie clip or other elements of a movie.

Flash 5 deprecated the `_highquality` property; if you are authoring for a Flash 5 environment, use the `_quality` property instead.

SET MOVIE CLIP QUALITY



1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file quality.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

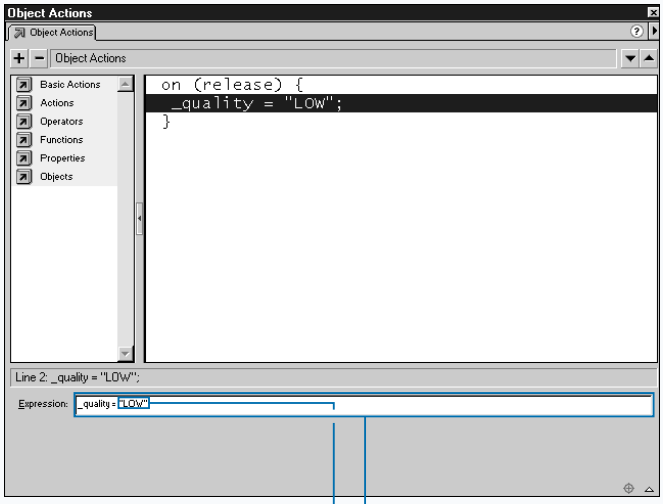
2 Click Window ➪ Actions to open the Actions panel.

3 Click ⊞ ➪ Properties ➪ _quality.

Extra

When using the `_quality` property, you can choose the quality level you want. The table that follows defines each of the quality levels available in Flash.

LEVEL	DEFINITION
LOW	Flash does not antialias graphics or smooth bitmaps.
MEDIUM	Flash antialiases graphics, but does not smooth bitmaps.
HIGH	The default quality level. Flash antialiases graphics and smoothes bitmaps if the movie is static.
BEST	Flash antialiases graphics and always smoothes bitmaps.



■ The default handler `on (release)` appears in the Action list.

■ The `_quality` property appears in the action list.

4 Set the `_quality` value.

■ This is the value.

5 Move to the test environment.

Note: See page 38 for instructions on how to test a movie.

6 Click the button to test your movie.

■ The movie clip quality changes to the value you specified.

INTRODUCTION TO VARIABLES AND DATA TYPES

You use variables to store information for later use. The syntax for creating a variable is `variableName = value;`.

The `variableName` argument represents the name you give to the variable. The equal sign (=) is the assignment operator. The assignment operator tells ActionScript you want to assign something to a variable. Value represents what you want to assign to the variable. Once a value has been assigned, whenever you use the variable name, the assigned value is retrieved. For example, if you make the following assignment:

```
x = 2
```

every time ActionScript sees the variable `x`, it interprets it to mean 2. You can change the value assigned to a variable many times and at any point in your script.

DATA TYPES

In ActionScript, a variable can store the following data types: string, number, Boolean, movie clip, or object. A string is any sequence of characters consisting of any combination of letters, numbers, or punctuation marks. A number is a value on which you can perform mathematical operations such as addition (+), subtraction (–), multiplication (*), division (/), modulo (%), increment (++), or decrement (--). A Boolean is a value that is either true or false. A movie clip is a symbol that can play Flash animation. Objects are used to manipulate data, sounds, and movie clips.

You do not have to define the variable type before assigning a value to a variable. ActionScript examines the expression and determines whether the variable is a number, string, Boolean, object, or movie clip. If you change the value assigned to a variable, if necessary, ActionScript will automatically change the variable type. For example, when you assign `x = 2;`, ActionScript evaluates the expression and determines that `x` is a number. If you later change the assignment to `x = "George";`, ActionScript reevaluates the expression and determines that `x` is a string.

SCOPE OF VARIABLES

ActionScript variables can be either global or local. A global variable can be referenced by any movie clip on any timeline. Global variables are said to have an unlimited scope. A local variable can be referenced only within the curly braces that enclose its block of script. Local variables have a limited scope. Each variable must have a unique name within its scope. Each global variable must be unique within the movie. Each local variable must be unique within the curly braces that enclose it.

Declaring a variable lets ActionScript know that a variable exists even if no value has been assigned to it. You use the `var` statement to declare a local variable. In the example shown here, the variable `x` is declared as a local variable.

Example:

```
var x
```

You use the assignment operator (=) to assign a value to a local variable. In the example shown here, the value `25` is assigned to the local variable `x`.

Example:

```
var x = 25
```

You use the `setVariables` action or the assignment operator (=) to declare global variables. The following two examples set global variables and are equivalent.

Examples:

```
x = 25
set(x,25);
```

Global variables are often initialized in the first frame of a movie. Initializing a variable consists of assigning the initial value of the variable.

NAME VARIABLES

You are free to name your variables anything you like; however, when naming your variables, you must follow these rules:

* The first character of your variable name must be a letter, underscore (_), or dollar sign ($).
* Each subsequent character must be a number, letter, underscore (_), or dollar sign ($).
* Your variable name cannot be a keyword.
* Your variable name cannot be a Boolean literal.
* Your variable name must be unique within its scope. Global variables must be unique within the movie. Local variables must be unique within the curly braces that enclose them.
* Your variable name does not have to be in the format of first word beginning with a lowercase letter and subsequent words beginning with capital letters; however, that is the convention in this book. If you develop a convention and use it consistently, you will have an easier time debugging your code.

DYNAMICALLY DISPLAY TEXT

You can use variables to dynamically display text or to create forms. If you want to dynamically display text, you start by using the Text tool to create a text box. After you have created the text box, click Window ➪ Panels ➪ Text Options on the menu. The Text Options panel opens. Select Dynamic Text as the text type and enter the name you want to assign to your variable in the Variable field. If you would like a border around your field, select Border/Bg. Any information you assign to the variable displays on the screen.

CAPTURE USER INPUT

If you want to create a form to capture user input, you start by using the Text tool to create a text box. The box will capture the user's entry. After you have created the text box, click Window ➪ Panels ➪ Text Options on the menu. The Text Options panel opens. Select Input Text as the text type and enter the name of your variable in the Variable field. Any information the user types in the field is stored in the variable you assigned. You reference Input Text variables and Dynamic Text variables the same way that you reference any other variable.

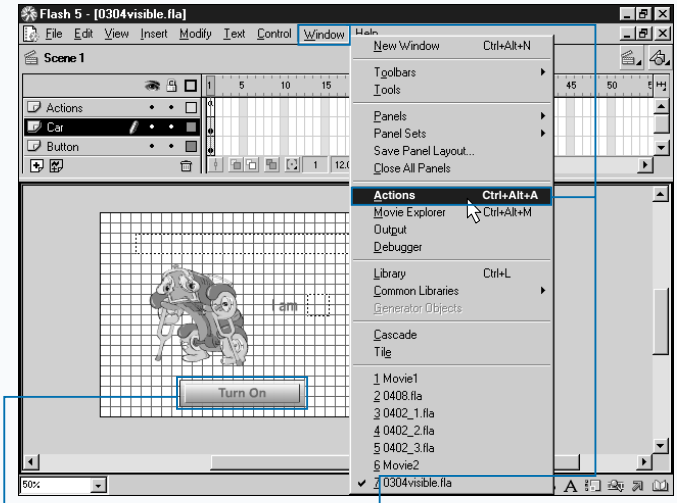
ASSIGN A VALUE TO A VARIABLE

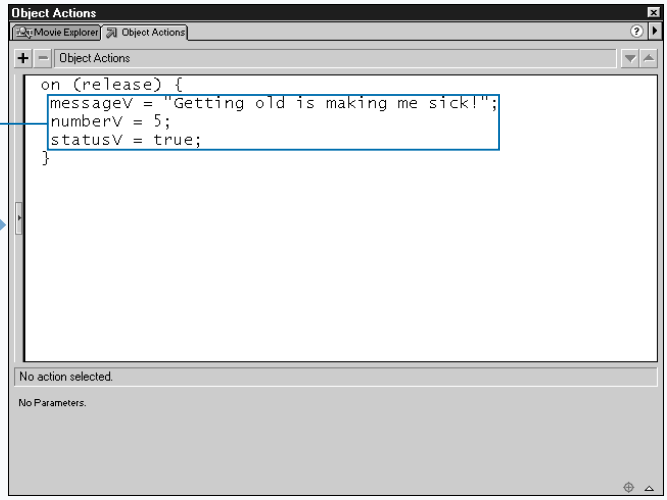
You can assign a string to a variable. A string is any sequence of characters consisting of any combination of letters, numbers, or punctuation marks. You use a string variable to store string values such as text or a URL for later use. You cannot perform mathematical operations on a string even if the string is a number. When assigning a string to a variable, you must enclose the string in single or double quotation marks. This example, `userName = "John Smith";`, stores a string to the variable `userName`.

You can assign a number to a variable and use the number as a counter, in a mathematical calculation, or to set a property. A number assigned to a variable can be the result of a mathematical calculation. These examples all assign numbers to a variable: `a = 5;`, `x = a + 25;`, and `x = 5 + 7;`. Any expression that returns a number that is assigned to a variable, assigns a number to the variable. ActionScript stores numbers as floating-point numbers. A floating-point number is a number with no fixed number of digits before or after the decimal point.

You use a Boolean when you want to set a condition to true or false, evaluate whether a condition is true or false, or compare values. Booleans are often used with logical operators. When assigning a Boolean, the value is always either the word `true` or the word `false` or an expression that evaluates to `true` or `false`. This example, `x = false;`, assigns a Boolean to a variable.

ASSIGN A VALUE TO A VARIABLE





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file variable.fla, which you can find on the CD-ROM that accompanies this book.

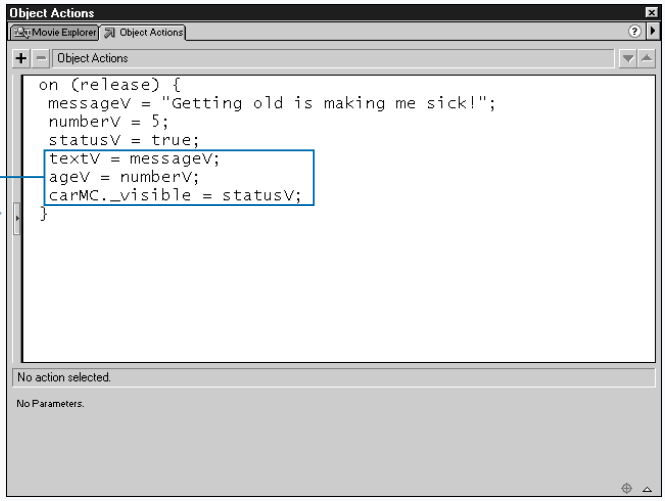
2 Click Window ➪ Actions to open the Actions panel.

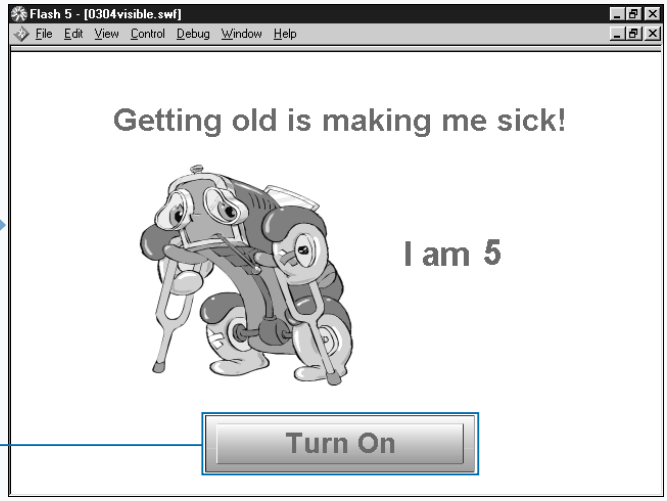
3 Assign values to variables.

Extra

Several characters cannot be included in a string unless they are preceded with a backslash. The table shown here lists these characters.

ESCAPE SEQUENCE	CHARACTER
\"	Double quotation mark
\'	Single quotation mark
\\	Backslash
\b	Backspace character (ASCII 8)
\f	Form feed character (ASCII 12)
\n	Line feed character (ASCII 10)
\r	Carriage return character (ASCII 13)
\t	Tab character (ASCII 9)
\000 - \377	A byte specified in octal
\x00 - \xFF	A byte specified in hexadecimal
\u0000 - \uFFFF	A 16-bit Unicode character specified in hexadecimal





4 Use the variables in an expression.

■ If you associate the variables with dynamic text boxes, you can display the values the variables return to the user.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button to test your movie.

■ The script assigned the values needed to execute the script to variables. Your movie will perform the actions the same way it would have if you had used literal values.

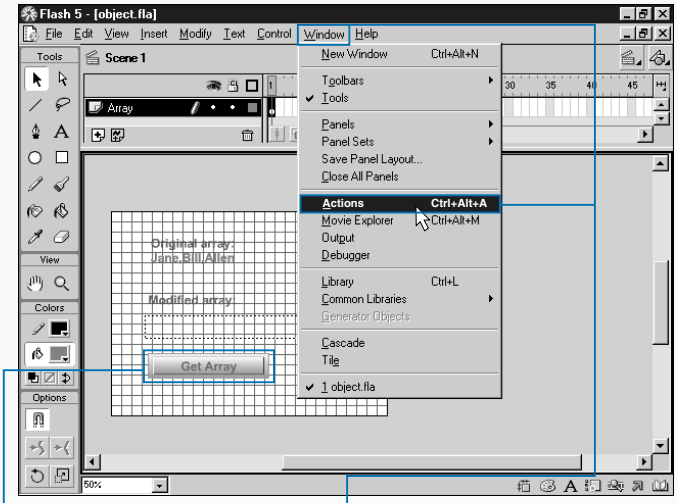
ASSIGN AN OBJECT OR MOVIE CLIP TO A VARIABLE

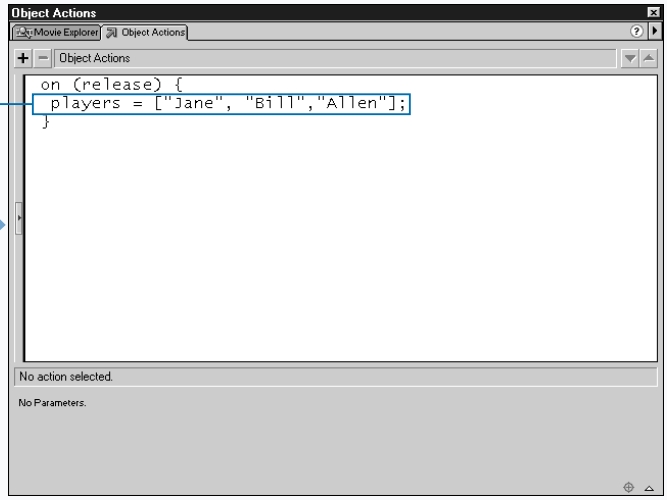
ActionScript divides data types into two categories: primitive and reference. Strings, numbers, and Booleans are primitive data types. A variable that contains a primitive data type stores the value that has been assigned to it. Primitive data types do not update when changes are made to the assigning value. Each time ActionScript encounters the variable it retrieves the value last assigned to it regardless of any subsequent updates, for example, `a = 10;,b = a;, a = 30;`. In this example, `b` is equal to `10`. The value of `b` remains `10` even though the value of `a` has changed to `30`. This is because `b` contains the value of `a` at the time of assignment. Changes to `a` do not automatically update `b`.

Objects and movie clips are reference data types. When you retrieve an object or movie clip from a variable, any changes made subsequent to the time you originally assigned the object to the variable are reflected. Variables assigned objects or movie clips do not store the object or movie clip assigned to them, but, instead, store a reference to the object or movie clip. Each time the variable is called, the object is retrieved. You can assign objects and movie clips to variables and use those variables to reference the object or movie clip.

An array is a list of values separated by commas. The following assigns rose, lily, and daisy to the array `flowers`: `flowers = ["rose", "lily", "daisy"];`. The syntax `flowers[0] = ["pansy"];` changes rose to pansy. Since you have changed the value of rose to pansy, `flowers` now returns pansy, lily, daisy. This is because changes made to a reference data automatically update.

ASSIGN AN OBJECT OR MOVIE CLIP TO A VARIABLE





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file object.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

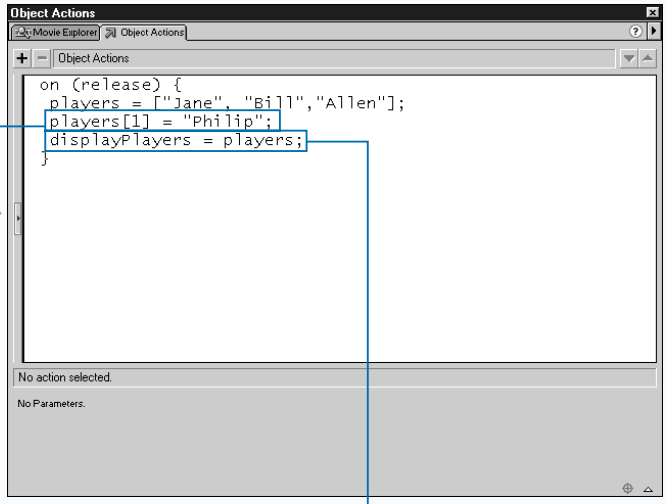
3 Create an array.

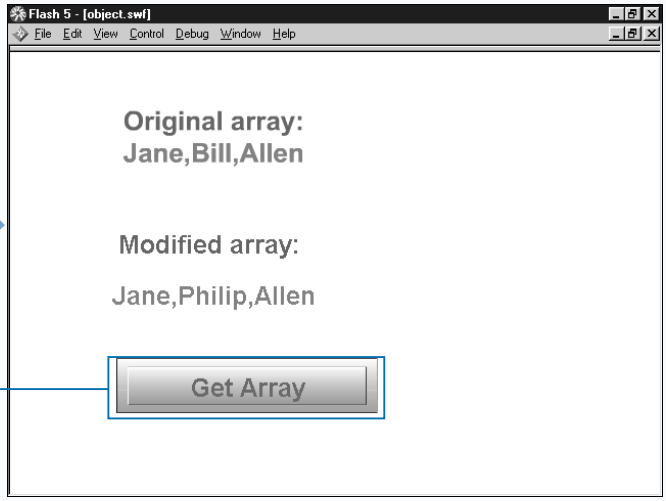
Note: See pages 164 to 175 for information on working with arrays.

Extra

Understanding the difference between primitive data types and reference data types can be confusing. The table that follows illustrates the difference by comparing values assigned to variables with values assigned to an array. For more information on arrays, see Chapter 7.

DATA TYPE: ARRAY	
`v1 = "a,"` `v2 = "b,"` `v3 = c`	
`y = v1+ v2 + v3`	`y = ["a","b","c"];`
`y returns: a,b,c`	`y returns: a,b,c`
`v1 = "z,";`	`y[0] = "z";`
`output = y`	`output = y`
`Output returns: a, b, c`	`output returns: z, b, c`





4 Change the array.

- This example changes one of the elements in the array.
- If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

5 Assign the array to a variable.

6 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

7 Click the button to test your movie.

- Because an array is a reference data type, the change you made to the array is reflected in the output.

LOAD VARIABLES

The `loadVariables` action reads data from an external file. This is useful when you need to read data from a text file or you are working with data from a CGI, Active Server Page (ASP), or Personal Home Page (PHP) script. Your Flash movie and the data must be located in the same subdomain if you are accessing the data using a Web browser.

The syntax for load variable is `loadVariables (url, location, variables);`. The `url` argument is used to specify the absolute or relative URL of the file containing the variables you want to load. The `location` argument is used to specify the level or target path that will receive the variables. For more information on levels and target paths, see page 224. The variables argument is used to specify the method you want to use to send variables. Choose from `get` and `post`.

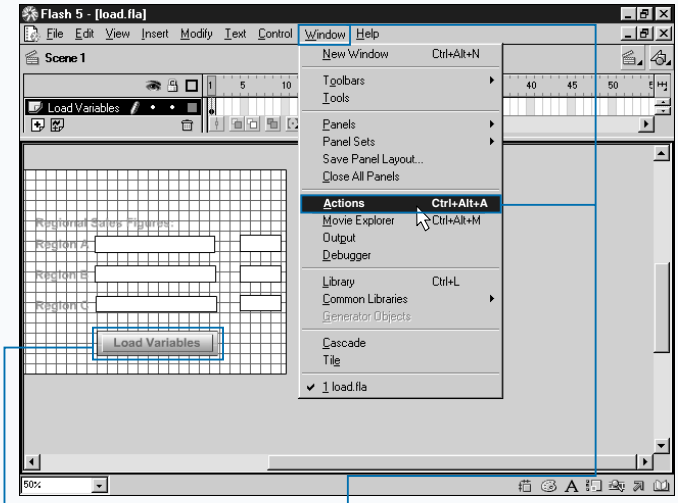
Set up the data you want to load by entering it in the following format:
`variableName=value&variableName=value&variableName=value...`

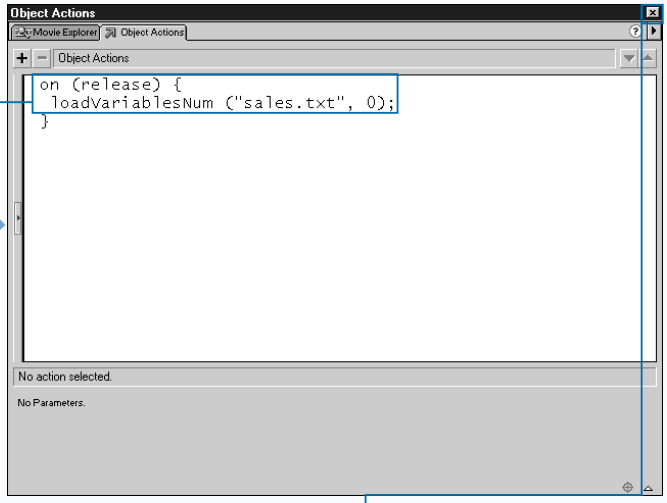
Use `variableName` to specify the name of the variable. Use `value` to specify the value you are assigning to the variable. Separate each variable/value pair with an `&`. The variable names in your text file must match the variable names in your movie.

Certain characters cannot be read directly from a text file. You must use URL encoding for these characters. For example, if your text file includes `temperature=+15`, ActionScript will read it as temperature=15, dropping the plus. The URL code for the + is `%2b`. To have ActionScript read the entry correctly, type `temperature=%2b15`. See Chapter 14 for the URL encoding table.

ActionScript sometimes needs to substitute `loadVariable` with `loadVariableNum`. If you are entering your script using the Normal mode, this substitution is done automatically. See the appendix for more information on `loadVariable` and `loadVariableNum`.

LOAD VARIABLES





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses files load.fla and sales.txt, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Load the variables.

■ If you associate the variables with dynamic text boxes, you can display the values the variables return to the user.

4 Click ☒ to close the Actions panel.

Apply It

You can use the `escape` function to convert an expression to URL-encoded format. The syntax for the `escape` function is `escape(expression)`. Use the expression argument to specify the expression you want to convert to URL-encoded format.

TYPE THIS:

```
escape("temperature=+15");
```

RESULT:

temperature%3D%2B15

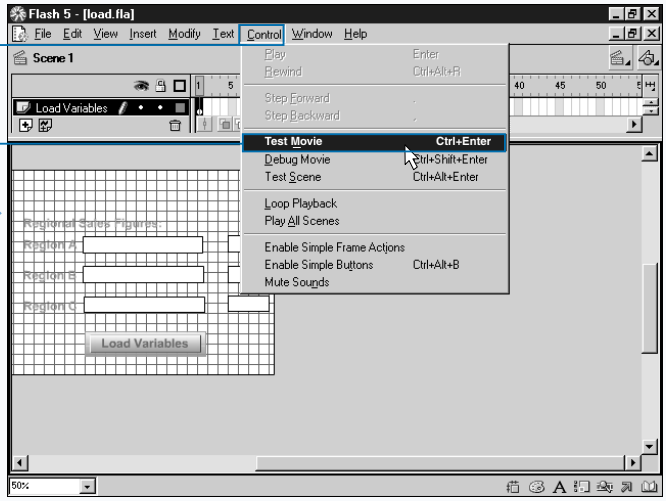
You can use the `unescape` function to convert an expression from URL-encoded format to ASCII characters. The syntax for the `unescape` function is `unescape(expression)`. Use the `expression` argument to specify the expression you want to convert from URL-encoded format to ASCII.

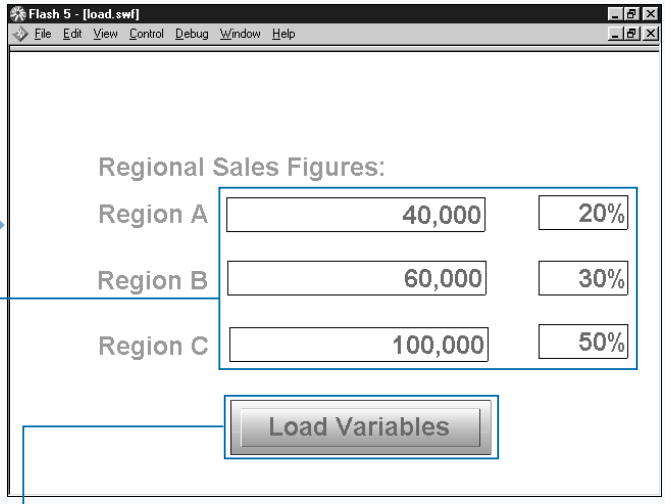
TYPE THIS:

```
unescape("temperature%3D%2B15")
```

RESULT:

temperature=+15





5 Click Control ➪ Test Movie to move to the text environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button to test your movie.

■ Each of these boxes is a dynamic text box.

■ When you click the button, the variables load.

EXTRACT A CHARACTER FROM A STRING

The `string.charAt` method enables you to extract the character at a specified location. Use this method to validate entries. For example, suppose the fourth character in an entry should be a dash. Use the `string.charAt` method to extract that character for evaluation. If the entry is not a dash, send an error message to the user.

When using the `string.charAt` method, precede the word `charAt` with the string or a variable that contains the string from which you want to retrieve a character followed by a period. Place the index position of the character you want enclosed in parentheses after the word `charAt`. Here is how you determine the index position. Starting from the left, the first character in a string has an index value of 0, the second character has an index value of 1, the third character has an index position of 2, and so on. In the example shown here, `x` is equal to `c`.

```
x = "abcdefg".charAt(2)
```

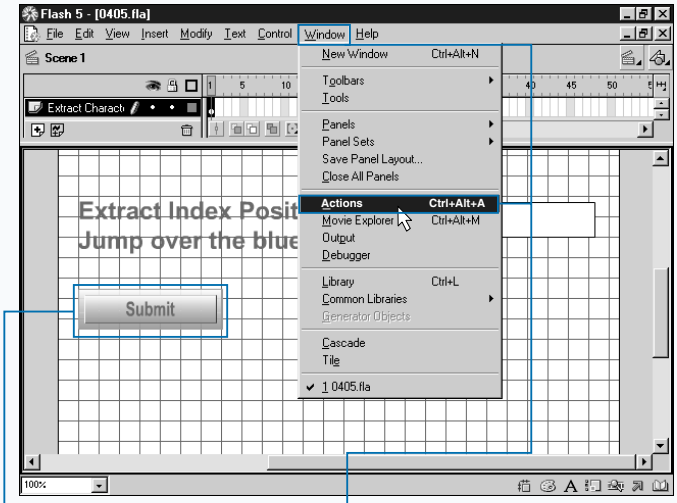
The letter `a` has an index of 0, the letter `b` has an index of 1, and the letter `c` has an index of 2.

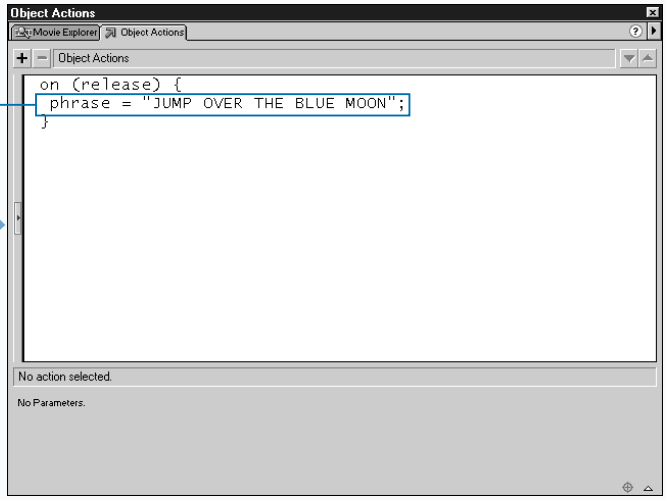
When using `string.charAt`, if you enter an index position that is not a number from 0 to the last index position in the string, ActionScript will return an empty string.

You use the `string.charCodeAt` method to retrieve the code for the character at a specified location in a string. The value returned is an integer from 0 to 65535. The `string.charAt` method and the `string.charCodeAt` method are the same, except the `string.charCodeAt` method retrieves a code instead of the character.

The syntax for `string.charCodeAt` is `stringValue.charCodeAt(index);`. `StringValue` represents the string from which you want to retrieve a numeric code. `Index` represents the index position of the value for which you want to retrieve a code.

EXTRACT A CHARACTER FROM A STRING





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file extract.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Assign a string to a variable.

Extra

This script illustrates using `charCodeAt`. It retrieves the ASCII code of a character in the variable `phrase1`. The variable `sampleText` is a dynamic text box.

Example:

```
on (release) {
    phrase = "JUMP OVER THE BLUE MOON";
    sampleText =phrase.charCodeAt(6);
}
```

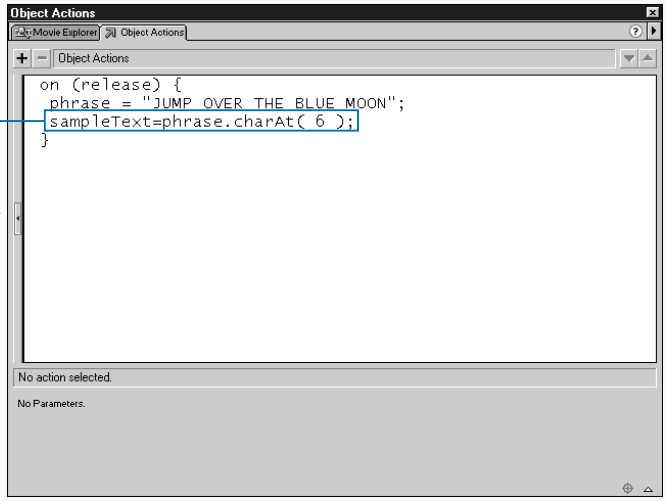
You use the `string.fromCharCode` method to obtain the character assigned to the ASCII code. The `string.fromCharCode` method uses the following syntax:

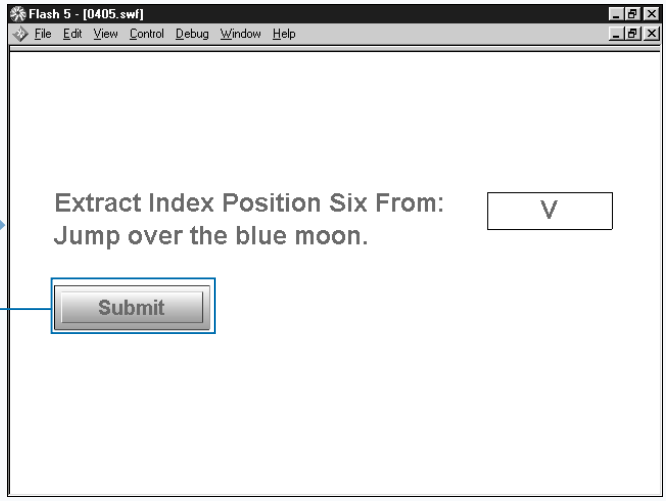
```
string.fromCharCode(code1,..., codeN);
```

Code1,..., CodeN represents the character code. In this script, the user enters a code and the character the code represents is retrieved. The variable `userEntry` is an Input text box. The variable `character` is a dynamic text box.

Example:

```
on (release) {
    character = string.fromCharCode(userEntry);
}
```





4 Extract a character.

- This example extracts the character at index position 6.
- If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button to test your movie.

- ActionScript extracts the character at index position 6.

EXTRACT A SUBSTRING FROM A STRING

You can use the `string.substring` method to extract a string that is embedded in another string. This is useful when you only need part of a string or you want to validate a user's entry. For example, if the fifth through ninth characters of a user's entry should always be numbers, you can use the `string.substring` method to extract those characters for evaluation. After evaluation, if the characters are not numbers, you can send an error message to the user. Suppose you had the data shown here: `userEntry = "PART12345-APL";`.

If you want the characters in the fifth through ninth position, use the `string.substring` method to extract them. When using the `string.substring` method, precede the word `substring` with the string or variable from which you want to extract data followed by a period. Follow the word `substring` with `from` and `to` arguments enclosed in parentheses and separated by a comma.

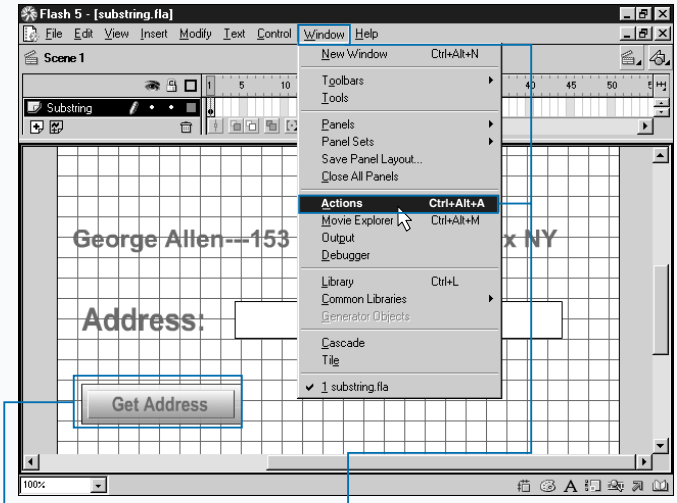
The `from` argument represents the index position of the first character you want. The `to` argument represents the index position plus 1 of the last character you want. If you do not specify a `to` value, the start position to the end of the string will be extracted. For an explanation of index position, see page 96. The expression shown here extracts 12345 from `userEntry`:

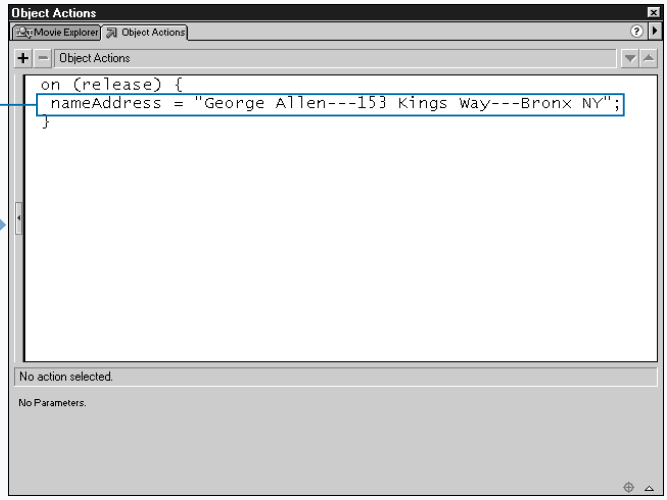
```
part = userEntry.substring(4,9);
```

When using the `substring` method, if the `from` position is greater than the `to` position, ActionScript will automatically swap the arguments. If the `from` position is equal to the `to` position, ActionScript will return an empty string.

ActionScript provides you with several methods that you can use to obtain a `substring`. The `substr` method uses the syntax shown here:
`stringValue.substr(start, length);`.
The `start` argument represents the index position of the first character of the substring to be extracted. The `length` argument represents the number of characters to be extracted.

EXTRACT A SUBSTRING FROM A STRING





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file substring.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Assign a string to a variable.

Apply It

You can also use the `slice` method to obtain a substring. The `slice` method uses this syntax:

```
stringValue.slice(start, end);
```

The `start` argument represents the index position of the first character of the substring to be extracted. If `start` is a negative number, the count begins at the end of the string. The `end` argument represents the index position for the last character to be extracted. If `end` is a negative number, the count begins at the end of the string.

TYPE THIS:

```
on (release) {
    nameAddress = "George Allen---153 Kings Way---Bronx NY";
    sampleText = nameAddress.slice( -24,-11);
}
```

RESULT:

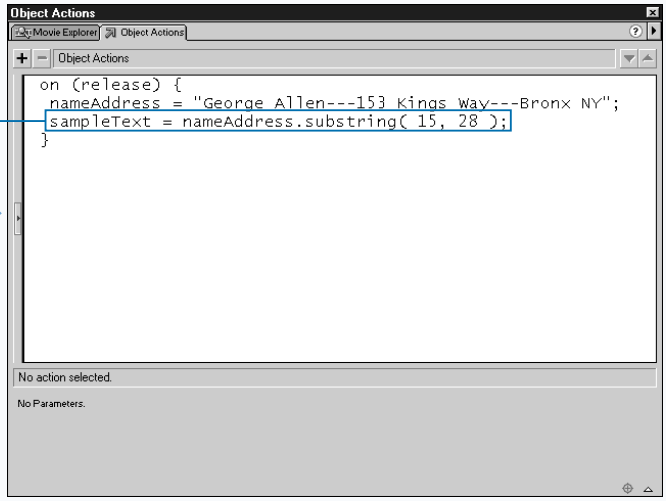
153 Kings Way

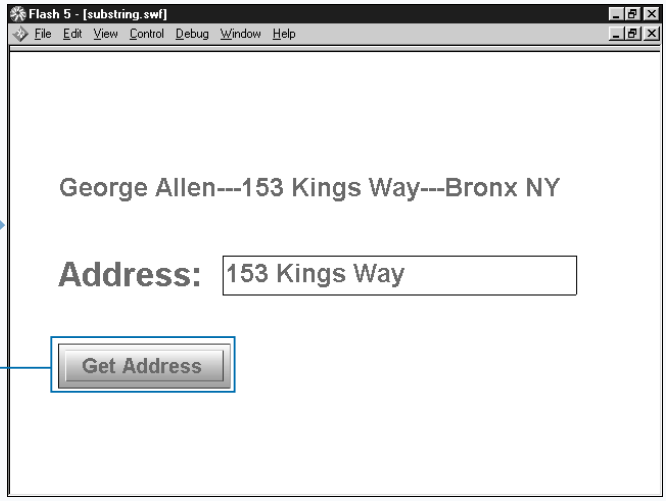
TYPE THIS:

```
on (release) {
    nameAddress = "George Allen---153 Kings Way---Bronx NY";
    sampleText = nameAddress.slice( 15,28);
}
```

RESULT:

153 Kings Way





4 Extract a substring.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button to test your movie.

■ ActionScript extracts the substring and displays it on screen.

CONCATENATE STRINGS

You can use the `string.concat` method to join several strings together to form a single string. This method is useful when you want to include data entered by the user in your output.

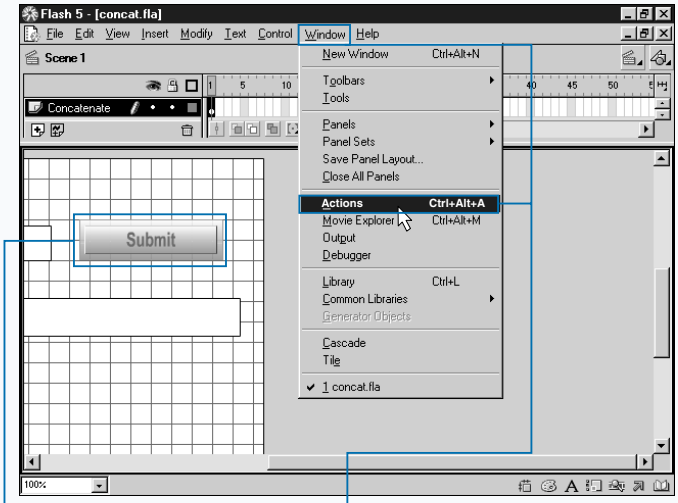
When using the `string.concat` method, precede the word `concat` with the string or variable to which you want to concatenate, followed by a period. After the word `concat`, list the values you want to concatenate, separated by commas and enclosed in parentheses. Any spaces at the beginning or end of the string are considered part of the string.

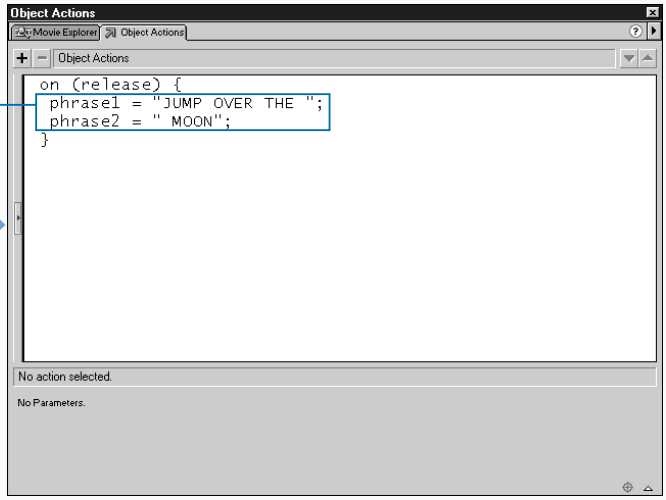
You can also concatenate or join two strings using the addition (+) operator. If one operand in an expression is a string, the addition operator will concatenate. If all operands in an expression are numbers, the addition operator will add them. Here also, any spaces at the beginning or end of the string are considered part of the string.

In Flash 4, the + operator was only used as a numeric operator. It did not concatenate. When Flash 4 movies are imported into Flash 5, operands that use the addition operator are converted to numbers.

In Flash 4, you used either the `add` operator or the `&` operator to concatenate strings. The `add` operator has been deprecated in Flash 5. You should use + to concatenate if you are creating movies for Flash 5. In Flash 5, the `&` operator is used as the `bitwise AND`. When Flash 4 files that use `&` are brought into Flash 5, the `&` is automatically converted to `add`.

CONCATENATE STRINGS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file concat.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Assign strings to variables.

Apply It

You can use the plus operator to concatenate strings. The script that follows provides an example. The variable `sampleText` is a dynamic text box. The variable `userEntry` is an input text box. The user enters a color in the `userEntry` field. When the user releases the mouse after clicking a button, the color they entered is concatenated with the variables phrase1 and phrase2.

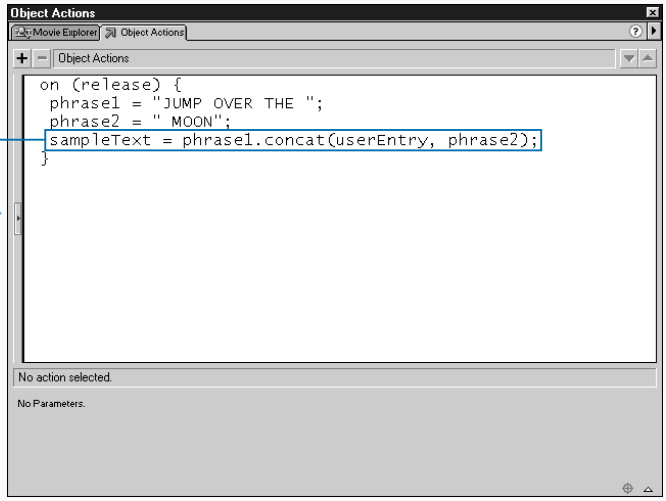
TYPE THIS:

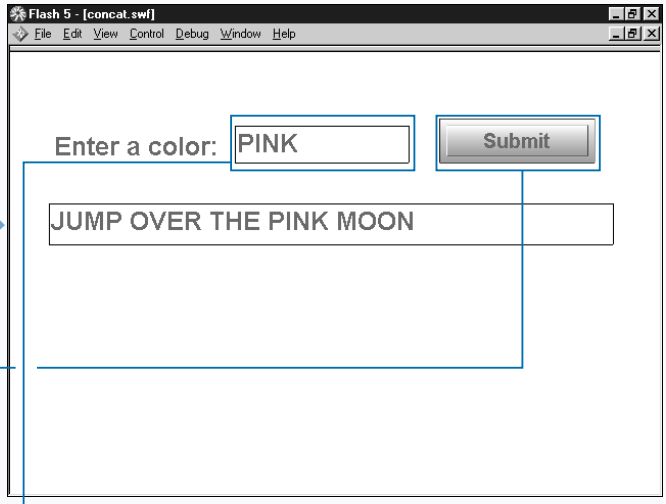
```
on (release) {
    phrase1 = "JUMP OVER THE ";
    phrase2 = " MOON";
    sampleText = phrase1 + userEntry + phrase2;
}
```

RESULT:

If the user enters PINK, here is the result.

JUMP OVER THE PINK MOON





4 Concatenate the strings.

■ If you associate the variables with dynamic text boxes, you can display the values the variables return to the user.

■ If you associate the variables with input text boxes, you can obtain input from the user.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Type a word or phrase.

7 Click the button.

■ ActionScript concatenates the strings.

OBTAIN THE LENGTH OF A STRING

You can use the `length` property to count the number of characters in a string. This property is useful when validating user input. For example, if the user enters a value that is required to be exactly 10 characters long, you can evaluate the entry and send an error message back to the user if the entry is more or less than 10 characters.

To use the `length` property, you simply precede the word length with the string or string variable for which you want to obtain the length.

In the example below, the string `"S2498"` is assigned to the variable `studentID`. Then, the length property is used to obtain the length of the value in the variable `studentID`. The variable `enteredLength` returns `5`.

```
studentID = S249876
enteredLength = studendID.length;
```

When calculating string length, spaces are included in the count. In the following example, `"Hello"` would have a length of 6.

Example:

```
"Hello".length
```

In Flash 4, the `length` function is used to obtain the length of a string. The syntax for the length function is `length(expression);`. Use the `expression` argument to specify the expression or variable for which you want to obtain the length.

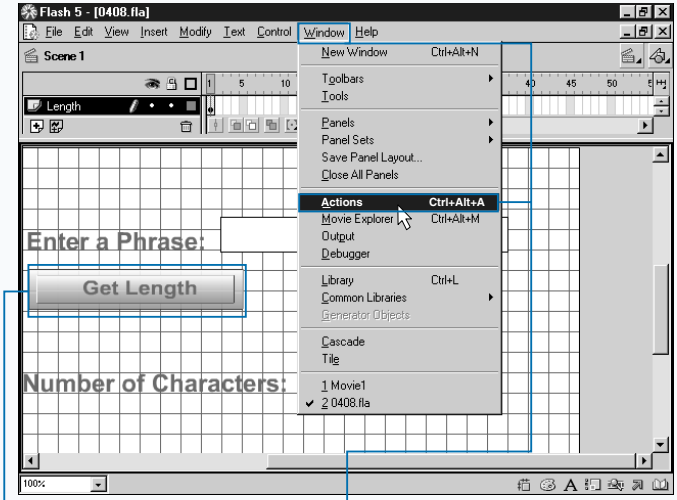
Example:

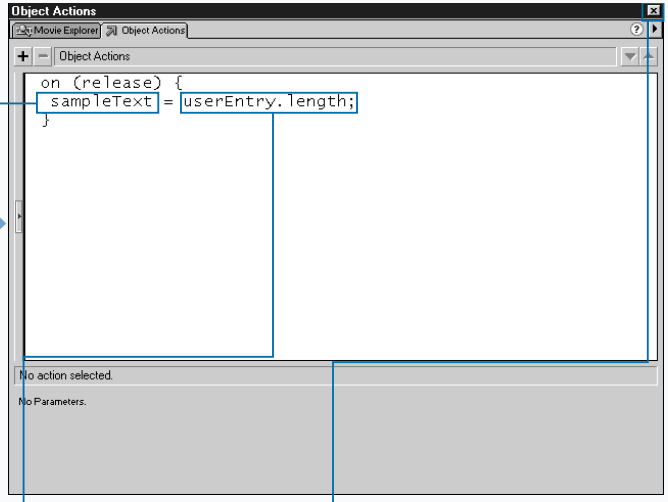
```
studentID = S249876
enteredLength = length(studentID)
```

The variable `enteredlength` returns `7`.

The `length` function has been deprecated in Flash 5. In you are authoring for a Flash 5 environment, you should use the `length` property instead.

OBTAIN THE LENGTH OF A STRING





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file length.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Obtain the length of a string.

■ If you associate the variable with an input text box, you can obtain input from the user.

4 Assign the results to a variable.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

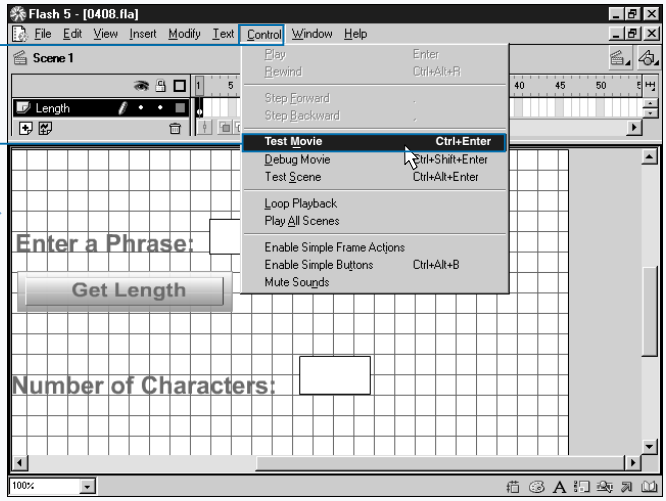
5 Click ☒ to close the Actions panel.

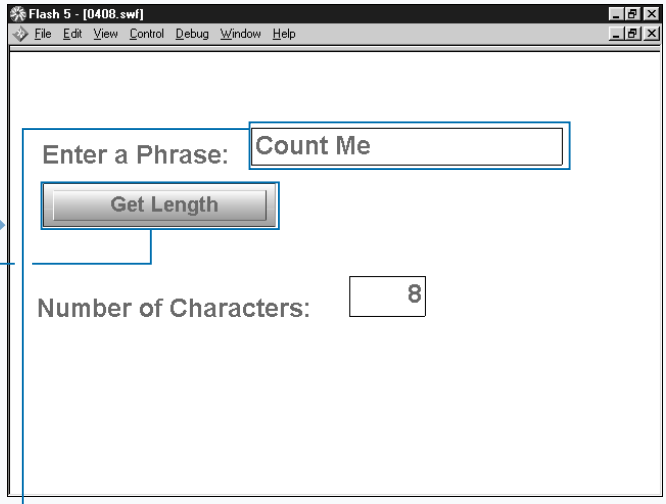
Extra

You can use ActionScript to check the length of an entry. The script shown here demonstrates. In this movie, the user enters an identification number. If the identification number is not five characters long, the user gets an error message. The script is attached to a button. The variable `studentId` is an Input Text field. The variable `errorMessage` is a Dynamic Text field.

Example

```
/* Executes script when the user releases the button. */
on (release) {
/* Checks the length of the input text box studentID. */
    if (studentID.length == 5 ) {
/* If the length of StudentID is fives character long, no
message is displayed in the dynamic text box errorMessage. */
        errorMessage = " ";
/*  If the length of StudentId is not equal to five, a
message is displayed in the field errorMessage. */
    } else {
        errorMessage = "Your ID must be 5 characters long.";
    }
}
```





6 Click Control ➪ Test Movie to move to the test environment.

Note: See page 38 for instructions on how to test your movie.

7 Type a word or phrase.

8 Click the button to test your movie.

■ ActionScript returns the number of characters in the text you typed.

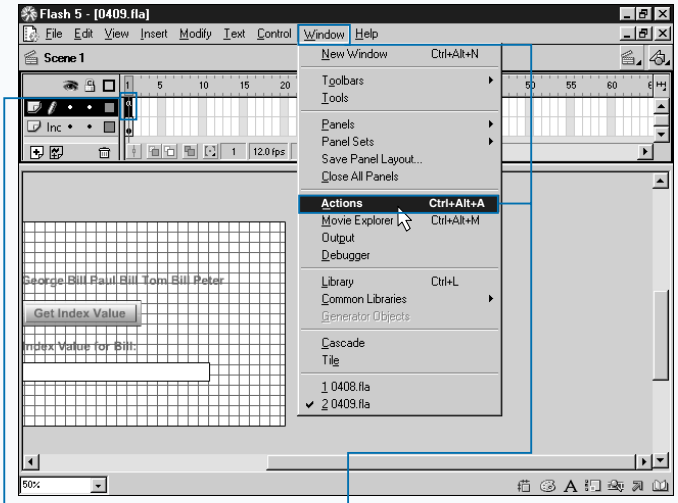
SEARCH A STRING

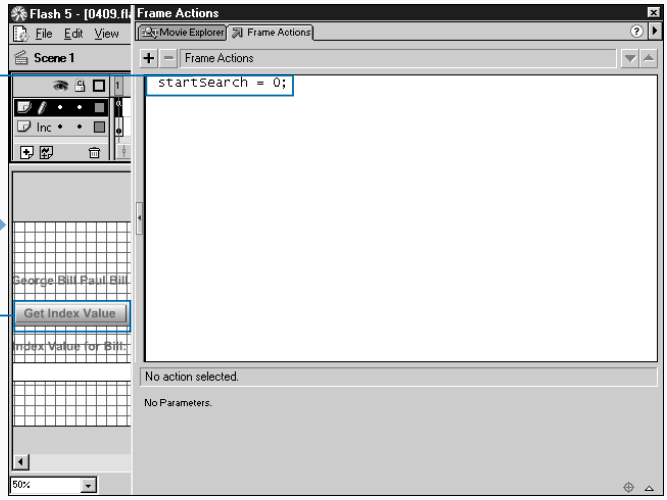
Using the `string.indexOf` method, you can locate a substring within a string. Use this method if you need to search a string for a word or phrase. The `string.indexOf` method returns the index value of the first occurrence of the string. If the value is not found, `string.indexOf` returns a -1.

To use the `string.indexOf` method, precede the word `indexOf` with the string or string variable you want to search, followed by a period. After the word `indexOf`, place the value and the start arguments separated by a comma and enclosed in parentheses. The `value` argument is used to specify the string or a variable containing the string for which you want to search. The `start` argument is used to specify the index position of the character at which you want to begin your search. For an explanation of index position, see on page 96. If you do not specify a start, ActionScript will begin the search at the beginning of the string.

If you would like to do multiple searches of a string, you need to initialize a variable to 0 in the first frame of the movie and use that variable as the start. For the purpose of this example, call the variable `startSearch`. After the script finds the substring, have the script assign the index value of the substring to a variable. For the purpose of this example, call that variable `indexValue`. Assign `indexValue +1` to `startSearch`. The next search will start at the index position to the right of the last value found. When no value is found, ActionScript will return a -1.

SEARCH A STRING





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file search.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a frame.

2 Click Window ➪ Actions to open the Actions panel.

3 Initialize the variable.

4 Select the frame, button, or movie clip to which you want to add ActionScript.

■ This part of this example uses a button.

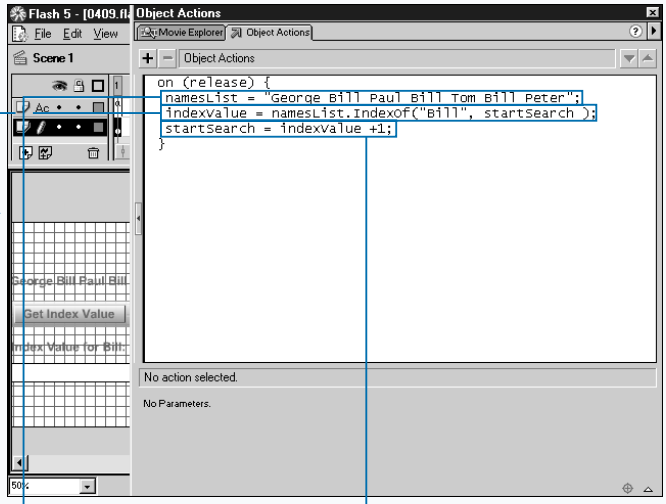
Extra

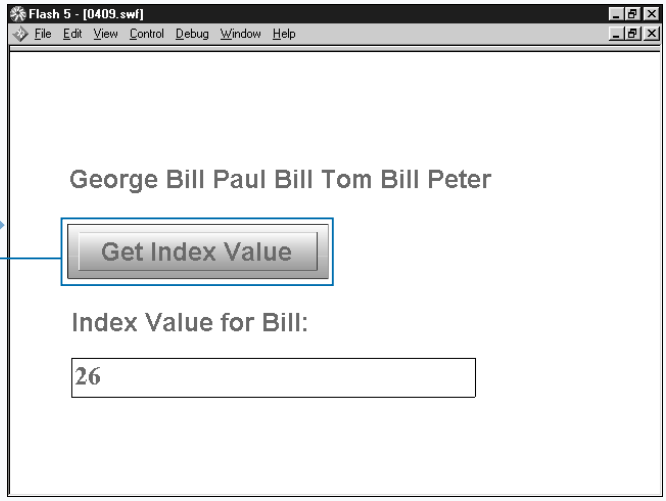
The `string.lastIndexOf` method works exactly like the `string.indexOf` method, except it returns the index value of the last occurrence of a substring within a string. The `string.lastIndexOf` method uses the following syntax.

```
stringValue.lastIndexOf(value, start);
```

The script shown here illustrates using the `string.lastIndexOf` variable. The variable `startSearch` begins the search at index position 36. The variable `indexValue` is a Dynamic Text box. It displays the results of the search. Note that because `string.lastIndexOf` searches from the end of the string to the beginning, after each search the variable `startSearch` is assigned the value `indexValue-1`.

```
Frame 1:
startSearch = 36;
Button:
on (release) {
    namesList = "George Bill Paul Bill Tom Bill Peter";
    indexValue = namesList.lastIndexOf("Bill", startSearch );
    startSearch = indexValue -1;
}
```





5 Assign a string to a variable.

6 Find the index of a value.

■ Associate the variable with a dynamic text box. This enables you to display the value the variable returns to the user.

7 Reset the variable you initialized in frame 1.

8 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

9 Click the button to test your movie.

■ Each time you click the button, ActionScript returns the next index value for the string for which you are searching.

CHANGE THE CASE OF A STRING

You can use the `string.toLowerCase` and `string.toUpperCase` methods to change the case of your text. This is useful when you are formatting output or when you want to compare strings without regard to case. The `string.toLowerCase` method turns all characters that are not already lowercase to lowercase. The `string.toUpperCase` method turns all characters that are not already uppercase to uppercase.

To use the `string.toLowerCase` method, simply precede the word `toLowerCase` with the string or string variable you want to convert to lowercase followed by a period. Follow the word `toLowerCase` with parentheses.

Example:

```
userEntry = "HELLO";
sampleText = userEntry.toLowerCase();
```

Result:

hello

The syntax for the `string.toUpperCase` method is similar to the syntax for the `string.toLowerCase` method. To use the `string.toUpperCase` method, simply precede the word `toUpperCase` with the string or string variable you want to convert to uppercase followed by a period. Then follow the word `toUpperCase` with parentheses.

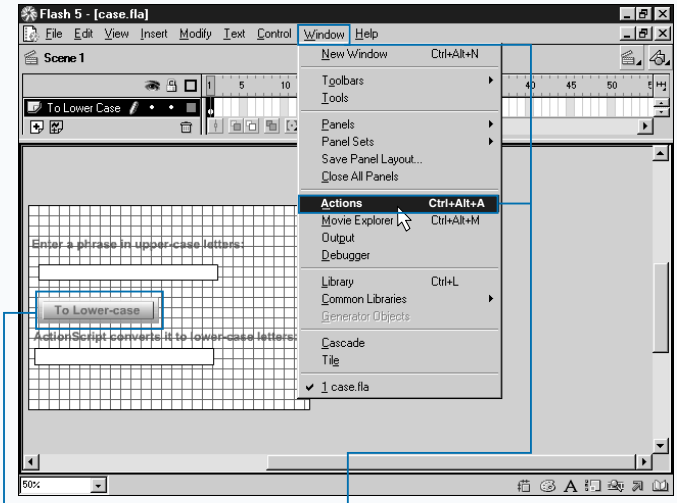
Example:

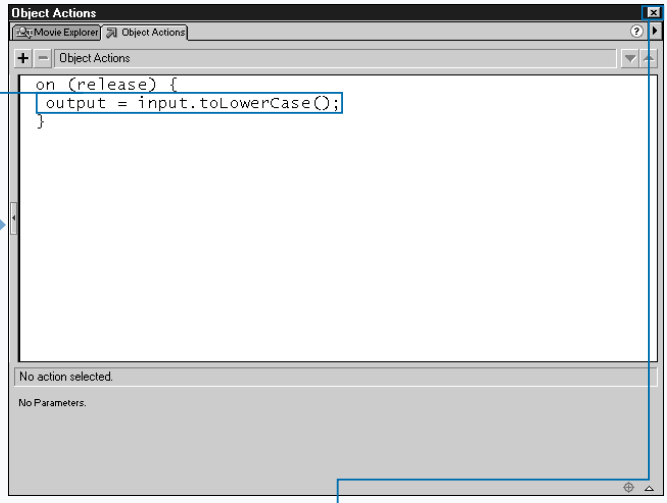
```
userEntry = "hello";
sampleText = userEntry.toUpperCase();
```

Result:

HELLO

CHANGE THE CASE OF A STRING





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file case.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Convert a string to lowercase.

■ Associate the variable with a dynamic text box. This enables you to display the value the variable returns to the user.

■ Associate the variable with an input text box. This enables you to obtain input from the user.

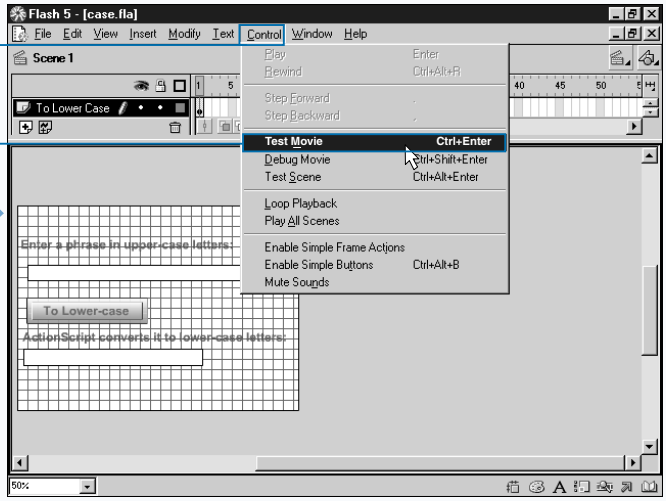
4 Click ☒ to close the Actions panel.

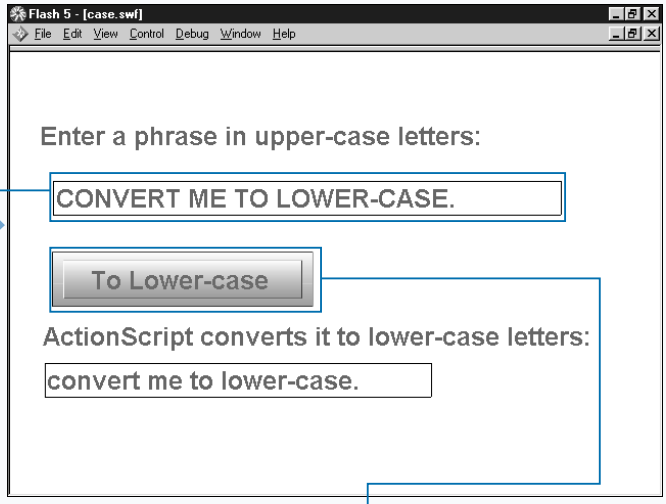
Extra

This example concatenates the user's entry with two phrases generated by the script. If the user's entry is in lowercase, the output is inconsistent. This script converts the user's entry to uppercase. For more information on concatenating, see page 100.

Example:

```
on (release) {
    phrase1 = "JUMP OVER THE";
    phrase2 = "MOON";
    sampleText = phrase1.concat(userEntry.toUpperCase(), phrase2);
}
```





5 Click Control ➪ Test Movie to move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Type a phrase in uppercase letters.

7 Click the button to test your movie.

■ ActionScript converts the uppercase letters to lowercase letters.

CONVERT STRINGS, NUMBERS, AND BOOLEANS

CONVERT TO A STRING

The `String` function converts a Boolean, number, object, or expression to a string. This is useful when you need the string equivalent of a Boolean, number, object, or expression. The string function uses the syntax shown here:

`String(expression)`

`Expression` represents the number, Boolean, variable, or object you want to convert to a string.

The following table illustrates the `String` function.

CONDITION	EXPRESSION	RETURN
If the expression is omitted, the function returns an error message.	`string()`	`Wrong number of parameters.`
If the expression is a Boolean literal, the function returns a string representation of the Boolean value.	`a= false; string(a)`	`false`
	`b = true; string(b)`	`true`
If the expression is a numeric, the function returns the decimal representation of the number.	`string(1/2)`	`.5`
If the expression is a string, the function returns the expression.	`a = "hello" string(a)`	`hello`
If the expression is an object, the function returns the string representation of the object.	`string(Date)`	`[type Function]`
If the expression is a movie clip, the function returns the movie clip in dot notation.	`a =(_root.sampleMC) string(a)`	`_level0.sampleMC`

CONVERT TO A BOOLEAN

The Boolean function converts a number, string, movie clip, object, or expression to a Boolean. This function is useful when you need to evaluate a condition. The Boolean function uses the syntax shown here:

`Boolean(expression)`

`Expression` represents the number, string, movie clip, object, or expression you wish to convert.

The following table illustrates the Boolean function.

CONDITION	EXPRESSION	RETURN
If the expression is omitted, the function returns the value false.	`Boolean()`	`false`
If the expression is any number other than 0, the function returns the value true.	`Boolean(-234.67)`	`true`
	`Boolean(10)`	`true`
If the expression is 0, the function returns the value `false`.	`x = 0; Boolean(x)`	`false`
If the expression is a Boolean value, the function returns the value of the expression.	`a = false; Boolean(a)`	`false`
	`b =true; Boolean(b)`	`true`

Continued

CONVERT TO A BOOLEAN (CONTINUED)

CONDITION	EXPRESSION	RETURN

If the expression is a movie clip or an object, the function returns true if the movie clip or object exists, otherwise it returns false.	`a =(_root.sampleMC);`
	`Boolean(a)`
		`true`
		If the movie clip exists.
		`false`
		If the movie clip does not exist.

If the expression is the string representation of any number other than 0, the function returns true.	`Boolean("123")`	`true`
	`Boolean("12.34")`	`true`
	`Boolean("-123")`	`true`
If the expression is not the string representation of any number other than 0, the function returns false.	`Boolean("0")`	`false`
	`Boolean("abc")`	`false`

CONVERT TO A NUMBER

The Number function converts a string, Boolean, or expression to a number. This is useful when you need the numeric equivalent of a string, Boolean, or expression. The number function uses the following syntax:

`Number(expression)`

`Expression` represents the Boolean, string, or object you wish to convert.

The following table illustrates the Number function.

CONDITION	EXPRESSION	RETURN
If the expression is omitted, the function returns an error.	`Number()`	`Wrong number of parameters.`
If the expression is a number, the function returns the value of the expression.	`x = 10;` `Number(x)`	`10`
If the expression is the Boolean value true, the function returns 1.	`a = true;` `Number(a)`	`1`
If the expression is the Boolean value false, the function returns 0.	`a = false` `Number(a)`	`0`
If the expression is a number represented as a string, the function returns the number.	`a= "10";` `Number(a)`	`10`
If the expression is a string not enclosed in quotes, the function returns 0.	`Number(abc)`	`0`

If the expression is a string variable or a string enclosed in quotes, the function returns `NaN`. `NaN` stands for "not a number."	`Number("abc")`
	
	`a=abc`
	`Number(a)`
		`NaN`
		
		`NaN`

UNDERSTAND PRECEDENCE

Operators are characters that you can use to combine, compare, or modify values. When using operators, you must be aware of precedence. Precedence determines the order in which ActionScript performs calculations, for example, `x = 1 + 2 * 3;` .

ActionScript performs multiplication and division before addition and subtraction, in accordance with its rules of precedence. In the example, x is equal to 7. ActionScript multiplies 2 times 3 to get 6 and adds the result to 1. You can change the order of calculation by using parentheses. Parentheses override the normal order of precedence causing ActionScript to calculate the items enclosed in parentheses first, for example, `y = (1+2) * 3;` .

In this example, `x` is equal to `9`. ActionScript adds `1` plus `2` to get `3` and multiplies the result by `3`. If you nest parentheses, ActionScript evaluates the innermost parentheses first. If two or more operators have the same precedence, ActionScript uses associativity to determine the order in which to perform the calculation. An operator can have an associativity of either left to right or right to left. If an operator has an associativity of left to right, ActionScript performs the calculation from left to right, or vice versa for right to left.

TYPES OF OPERATORS

Operators are characters that you use to combine, compare, or modify values. ActionScript calls the elements on which operators perform operands. There are three types of operators: unary, binary, and ternary.

Unary operators accept one operand. Increment and decrement are unary operators. Most unary operators prefix the operand. The increment and decrement operators, however, can either prefix or suffix the operand, depending on the operation you want to perform. For more about increment and decrement, see pages 122 to 125.

Binary operators are the most common type of operator. Binary operators accept two operands. You place binary operators between the operands. Plus (`+`) and minus (`-`) are binary operators.

Ternary operators accept three operands. The conditional operator (`?:`) is a ternary operator. ActionScript evaluates `expression1`. If `expression1` is `true`, ActionScript returns `expression2`; if not `true`, ActionScript returns `expression3`. For more on the conditional operator, see Chapter 6.

OPERATOR PRECEDENCE AND ASSOCIATIVITY

OPERATOR	DESCRIPTION	ASSOCIATIVITY
HIGHEST PRECEDENCE		
+	Unary plus	Right to left
–	Unary minus	Right to left
~	Bitwise one's complement	Right to left
!	Logical NOT	Right to left
not	Logical NOT (Flash 4 style)	Right to left
++	Post-increment	Left to right
--	Post-decrement	Left to right
()	Function call	Left to right
[ ]	Array element	Left to right
.	Structure member	Left to right
++	Pre-increment	Left to right
--	Pre-decrement	Right to left
new	Allocate object	Right to left
delete	Deallocate object	Right to left
typeof	Type of object	Right to left
void	Returns undefined value	Right to left
*	Multiply	Left to right
/	Divide	Left to right
%	Modulo	Left to right
+	Add	Left to right
add	String concatenation (formerly &)	Left to right
-	Subtraction	Left to right
<<	Bitwise left shift	Left to right
>	Bitwise right shift	Left to right
>>	Rightwise right shift (unsigned)	Left to right

OPERATOR	DESCRIPTION	ASSOCIATIVITY
HIGHEST PRECEDENCE		
<	Less than	Left to right
<=	Less than or equal to	Left to right
>	Greater than	Left to right
>=	Greater than or equal to	Left to right
lt	Less than (string version)	Left to right
le	Less than or equal to (string version)	Left to right
gt	Greater than (string version)	Left to right
ge	Greater than or equal to (string version)	Left to right
==	Equal	Left to right
!=	Not equal	Left to right
eq	Equal (string version)	Left to right
ne	Not equal (string version)	Left to right
&	Bitwise AND	Left to right
^	Bitwise XOR	Left to right
|	Bitwise OR	Left to right
&&	Logical AND	Left to right
and	Logical AND (Flash 4)	Left to right
||	Logical OR	Left to right
or	Logical OR (Flash 4)	Left to right
?:	Conditional operator	Right to left
=	Assignment	Right to left
*=,/=,%=, +=,-=,&=, \=,^=,<<=, >=,>>=	Compound assignment	Right to left
,	Multiple evaluation	Left to right

ADD NUMERIC VALUES

You can use the plus (+) operator to add numeric values. This operator is essential when you are performing mathematical operations. The + operator enables you to take one numeric value, add it to another, and obtain the sum. To use the plus operator, you place the plus sign + between two numeric `expressions`; ActionScript adds them. A numeric `expression` is any `expression` that ActionScript can evaluate and return a numeric value. The syntax for the plus operator action is

```
numericExpression1+numericExpression2
```

You can use the plus + operator to add numbers, numeric variables and numbers, or numeric variables, for example:

```
z=x+2;
z=x+y;
```

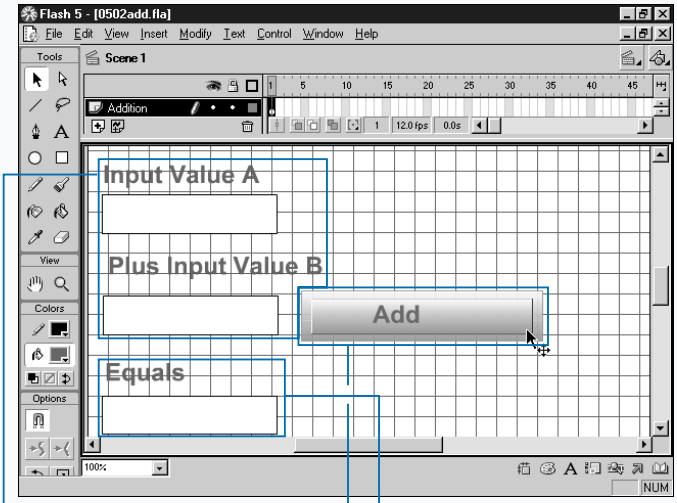
Concatenating joins two or more strings together to form a single string. You can use the plus + operator to concatenate strings. If one operand in an expression is a string, the plus operator concatenates. If all operands in an expression are numbers, the plus operator adds. For more about concatenating strings, see Chapter 4.

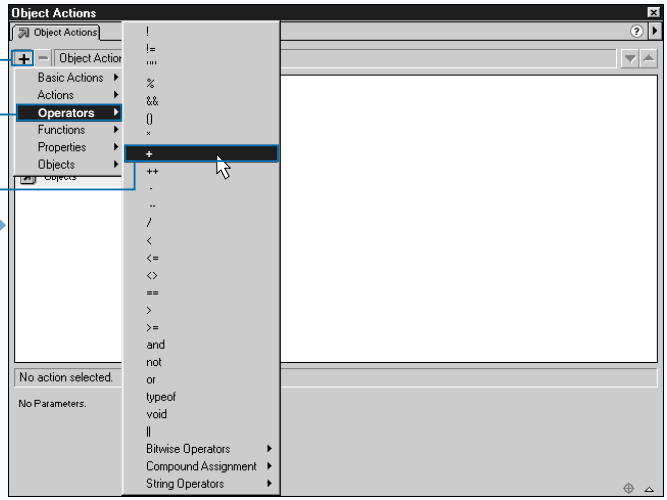
If a number is a string and you want to use the number to add, use the `Number` function to convert the string to a number, for example:

```
a="10";
x=20;
y=Number(a)+x;
```

You can use the plus operator with other numeric operators to perform complex mathematical calculations. You can add, subtract, multiply, and divide in the same expression, but you must be careful of precedence. Precedence controls the order of calculation. For example, ActionScript performs multiplication and division before addition and subtraction. For an explanation of precedence, see page 110.

ADD NUMERIC VALUES





1 Open file add.fla on the CD-ROM.

2 Create input text boxes.

3 Create a dynamic text box.

4 Create a button.

5 Click to select the button.

6 Open the Object Actions panel.

7 Click ⊞ ➪ Operators ➪ +.

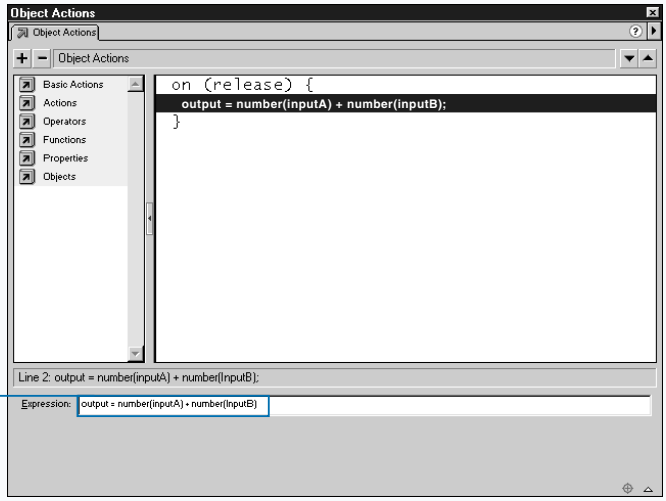
■ The Parameters panel opens.

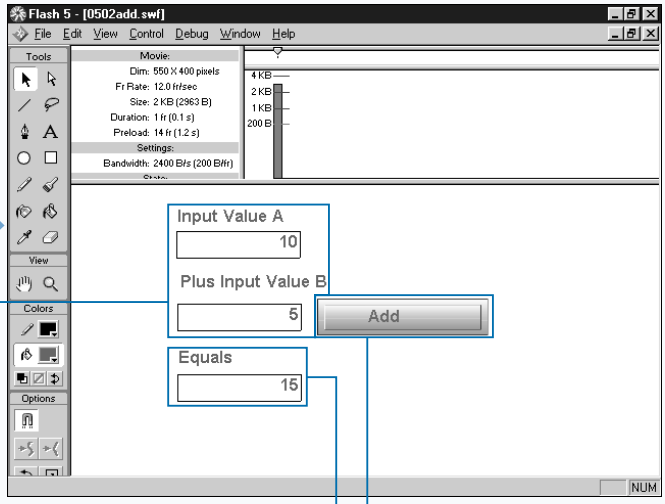
Apply it

This script illustrates calculating a subtotal. The user enters a value and clicks the Calculate button to add this entry to previous entries.

Example:

```
/* on (release) - tells Flash to start the
action when the user releases the mouse.*/
on (release) {
    /* output - a dynamic text box.  subtotal -
a dynamic text box.  input - an input text box.
Number(input) - converts the value of input to
a number. */
    /* The value of number(input) plus the
value of subtotal are added together and
assigned to output.*/
    output = Number(input) + subtotal;
    /* subotal - a dynamic text box. The
current value of Number(input) + the current
value of  subtotal are added together and
assigned to subtotal. */
    subtotal = Number(input) + subtotal;
    /* Resets the value of input to 0.*/
    input = 0;
}
```





8 Type the mathematical expression.

■ This example adds inputA to inputB.

9 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

10 Type in your values.

11 Click the button to calculate.

■ ActionScript adds the two input text boxes and displays the results in the dynamic text box.

SUBTRACT NUMERIC VALUES

You can use the minus (-) operator to subtract numeric values or to negate a number or `expression`. This operator is essential when performing mathematical calculations. To use the minus operator to subtract, you place the minus sign between two numeric `expressions`. A numeric `expression` is any statement that ActionScript can evaluate and return a numeric value. The syntax for the minus operator is

```
numericExpression1-numericExpression2
```

You can use the minus operator to subtract numbers, numeric variables and numbers, or numeric variables. If `x=4` and `y=2`, all of the following examples are valid:

```
z=2-1;
z=x-3-2;
z=x-y;
```

To use the minus operator to negate, you place the minus operator in front of a numeric `expression`. Negating reverses the sign of the `expression`, for example:

```
-2,
-(x+2)
```

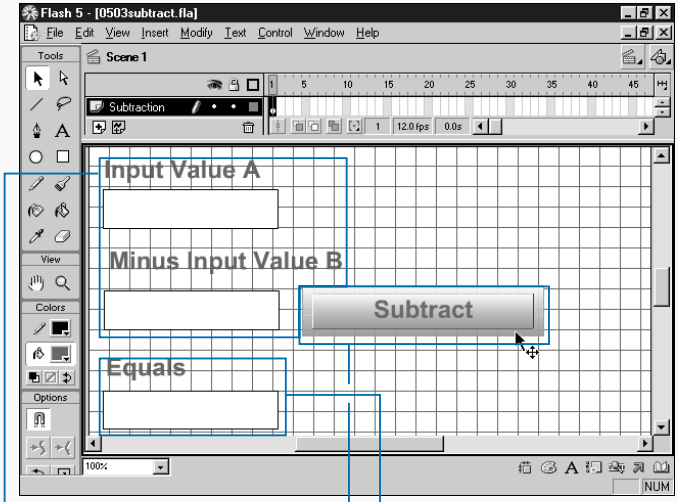
You can use the minus operator with other numeric operators to perform complex mathematical calculations. You can add, subtract, multiply, and divide in the same expression, but you must be careful of precedence. Precedence controls the order of calculation. For example, ActionScript performs multiplication and division before addition and subtraction. For an explanation of precedence, see page 110.

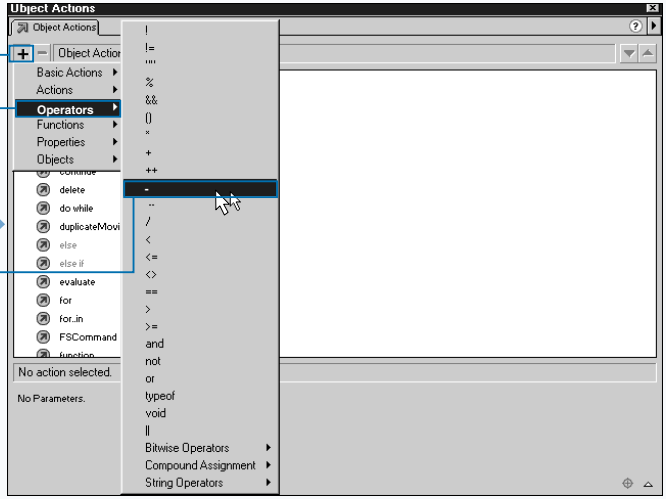
If you use a string when subtracting, ActionScript attempts to convert the string to a number. If ActionScript is unable to convert the string to a number, it returns `NaN`, which means not a number. You can use the `Number` function to convert the string to a number. To use the `Number` function, you type the word `Number`, followed by the `expression` you want to convert, enclosed in parentheses. For more on the `Number` function, see Chapter 4.

Example:

```
a="10";
x=5;
y=Number(10)-5;
```

SUBTRACT NUMERIC VALUES





1 Open file subtract.fla on the CD-ROM.

2 Create input text boxes.

3 Create a dynamic text box.

4 Create a button.

5 Click to select the button.

6 Open the Object Actions panel.

7 Click ⊞ ➪ Operators ➪ –.

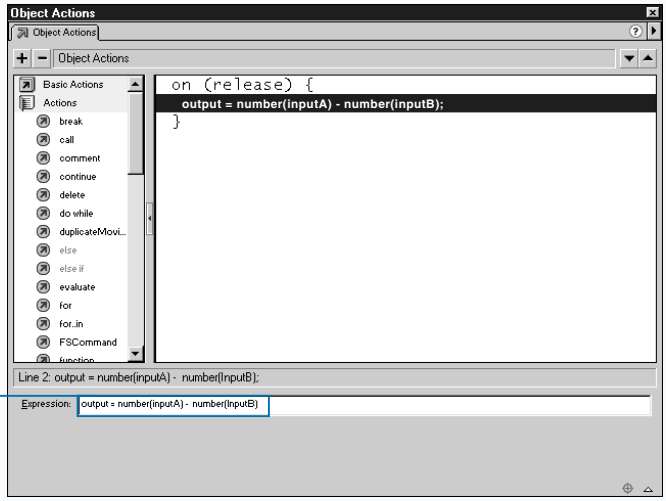
■ The Parameters panel opens.

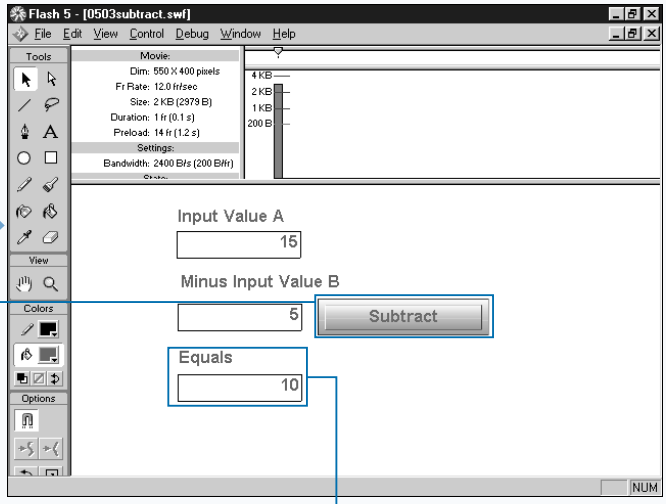
Apply It

In this script, the user types the cost of items and clicks the calculate button to calculate a subtotal. After the user has entered all the items, the user enters a discount amount in dollars. ActionScript subtracts the discount amount from the subtotal to calculate a final price.

Example:

```
/* on (release) - tells Flash to start the action when the user
releases the mouse button. */
on (release) {
    /* output - a dynamic text box.  subtotal - a dynamic text box.
input - an input text box. Number(input) - converts the value of
input to a number. */
    /* The value of number(input) plus the value of subtotal are
added together and assigned to output. */
    output = Number(input ) + subtotal;
    /* subtotal - a dynamic text box. The current value of
Number(input) + the current value of  subtotal are added together and
assigned to subtotal. */
    subtotal = Number(input) + subtotal;
    /* finalPrice - a dynamic text box.  discount - a dynamic text
box. number(discount) - converts the value of discount to a number.*/
    /* The value of number(discount) is subtracted from subtotal  and
the result is assigned to finalPrice. */
    finalPrice = subtotal - number(discount);
    /* Resets the value of input to 0.*/
    input = 0;
}
```





8 Type the mathematical expression.

■ This example subtracts inputB from inputA.

9 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

10 Click to calculate.

■ ActionScript subtracts using the entries in the input text boxes and displays the results in the dynamic text box.

MULTIPLY NUMERIC VALUES

You can use the multiplication (`*`) operator to multiply numeric values. This operator is essential when you are performing mathematical operations. To use the multiplication operator, you place an asterisk * between the numeric `expressions` you want to multiply. A numeric `expression` is any statement that ActionScript can evaluate and return a numeric value. The syntax for the multiplication operator is

```
numericExpression1*numericExpression2
```

ActionScript multiplies `numericExpression1` by `numericExpression2`. If your `expression` contains a series of calculations, they are performed from left to right. For example: `x = 3*2*8;`. The example multiplies `3` times `2`, yielding `6`, and then multiplies `6` times `8`, yielding `48`.

You can use the multiplication operator to multiply numbers, numeric variables and numbers, or numeric variables. If `x=4` and `y=2`, all of the following examples are valid.

```
z=2*3;
z=x*3;
z=x*y;
```

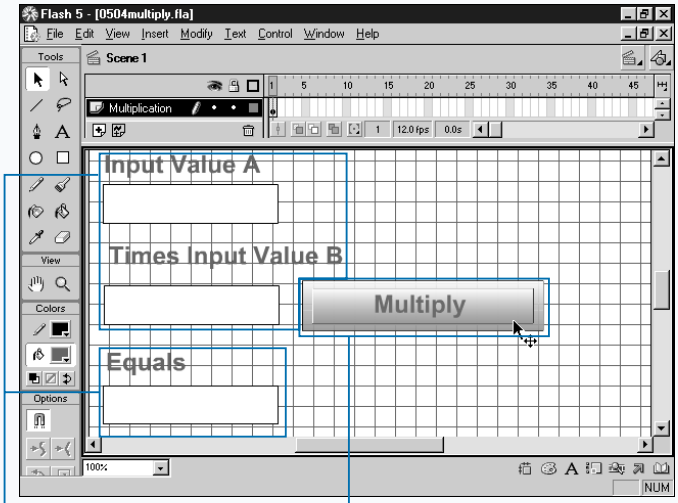
You can use the multiplication operator with other numeric operators to perform complex mathematical calculations. You can add, subtract, multiply, and divide in the same `expression`, but you must be careful of precedence. Precedence controls the order of calculation. For example, ActionScript performs multiplication and division before addition and subtraction. For an explanation of precedence, see page 110.

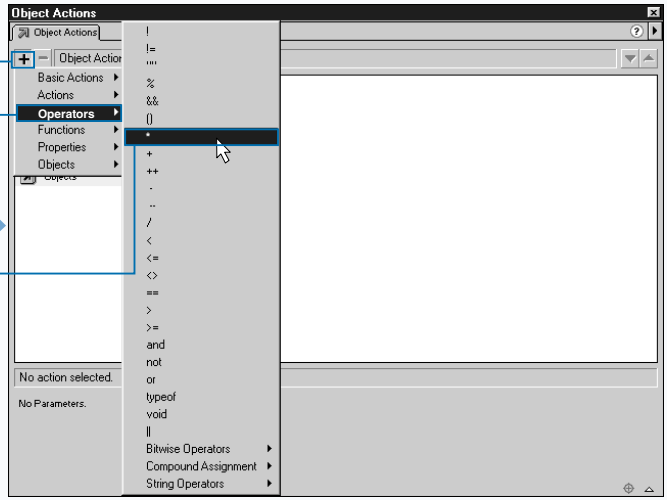
If you use a string in your calculation, ActionScript attempts to convert it to a number. If ActionScript is unable to convert it to a number, it returns `NaN`, which means not a number. To convert a string to a number, use the `Number` function. To use the `Number` function, type the word `Number`, followed by the `expression` you want to convert, enclosed in parentheses. For more about the `Number` function, see Chapter 4.

Example:

```
a="10";
x=5;
y=Number(10)*5;
```

MULTIPLY NUMERIC VALUES





1 Open file multiply.fla on the CD-ROM.

2 Create input text boxes.

3 Create a dynamic text box.

4 Create a button.

5 Click to select the button.

6 Open the Object Actions panel.

7 Click ⊞ ➪ Operators ➪ *.

■ The Parameters panel opens.

Extra

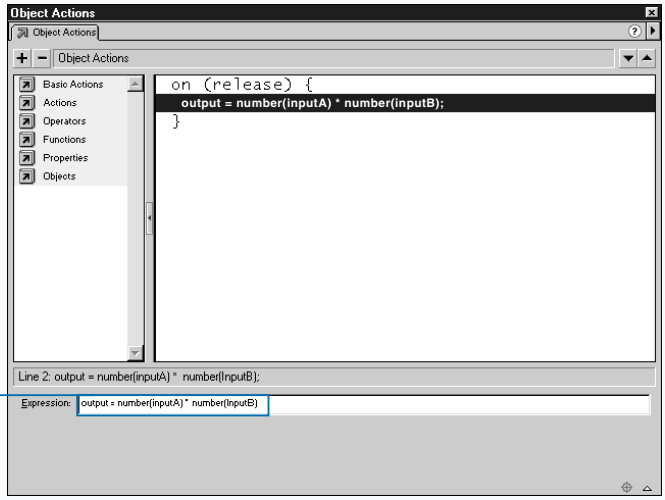
Your mathematical calculations can be as complex as necessary, combining numbers and variables. If you nest parentheses, ActionScript calculates the innermost `expressions` first.

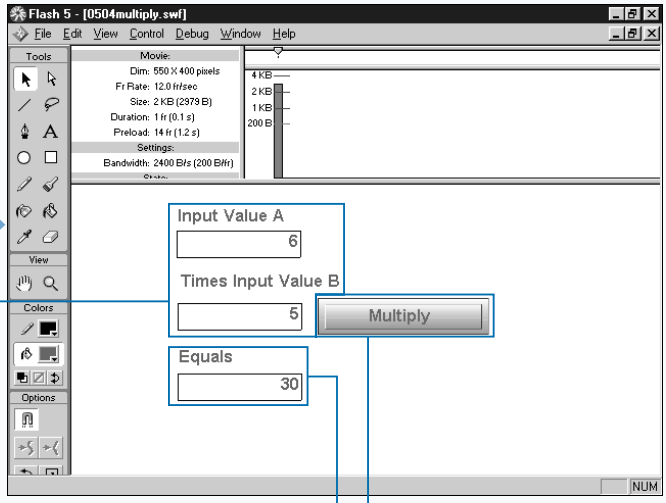
TYPE THIS:

```
x=25.356;
y=4 *x +10;
z=(2+4)-(30*(-5*x));
output=z+y*z+(2+4);
```

RESULT:

```
428273.9856
```





8 Type the mathematical expression.

■ This example multiplies inputA by InputB.

9 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

10 Type in values.

11 Click to calculate.

■ ActionScript multiplies using the entries in the input text boxes and displays the results in the dynamic text box.

DIVIDE NUMERIC VALUES

You can use the division (/) operator to divide numeric values. The division operator is essential when you are performing mathematical calculations. To use the division operator, you place the / operator between two numeric expressions. A numeric expression is any statement that ActionScript can evaluate and return a numeric value. The syntax for the division operator is

```
numericExpression1/numericExpression2
```

ActionScript divides numericExpression1 by numericExpression2. If your expression contains a series of calculations, they are performed from left to right, for example, 18/3/2. The example above divides 18 by 3, yielding 6, and then divides 6 by 2, yielding 3.

You can use the division operator to divide numbers, numeric variables and numbers, or numeric variables. If x=4 and y=2, all of the following examples are valid.

```
6/3
x/4
x/y
```

You can use the division operator with other numeric operators to perform complex mathematical calculations. You can add, subtract, multiply, and divide in the same expression, but you must be careful of precedence. Precedence controls the order of calculation. For example, ActionScript performs multiplication and division before addition and subtraction.

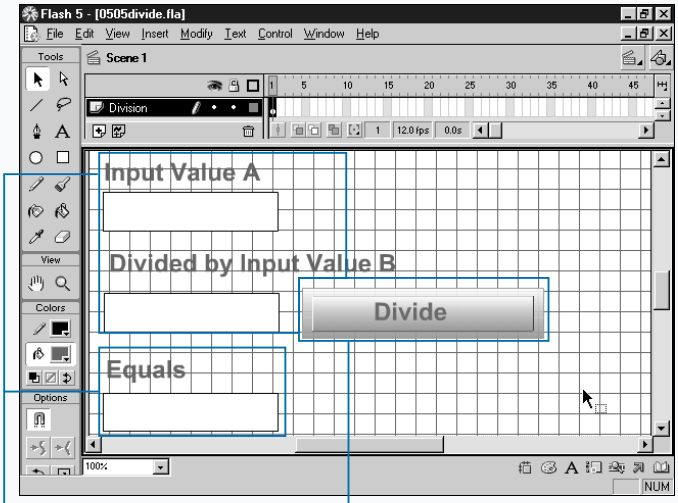
If you use a string in your calculation, ActionScript attempts to convert it to a number. If ActionScript is unable to convert the string to a number, it returns NaN, which means "not a number." You can use the Number function to convert a string to a number. To use the Number function, you type the word Number, followed by the expression you want to convert, enclosed in parentheses.

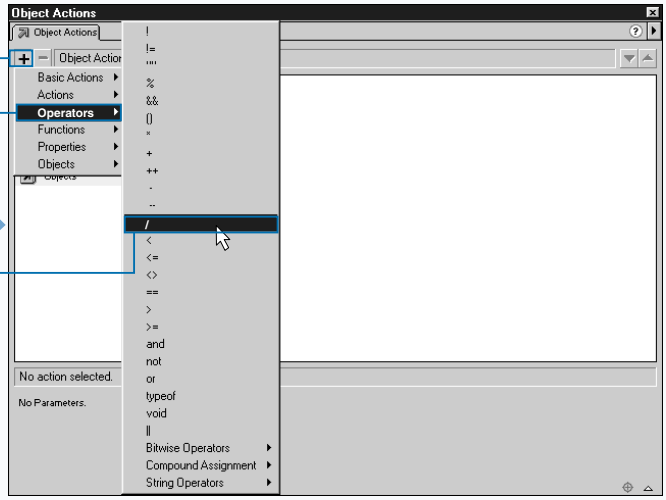
Example:

```
a="10";
x=5;
y=Number(10)/5;
```

If you divide by 0, ActionScript returns infinity.

DIVIDE NUMERIC VALUES





1 Open file divide.fla on the CD-ROM.

2 Create input text boxes.

3 Create a dynamic text box.

4 Create a button.

5 Click to select the button.

6 Open the Object Actions panel.

7 Click ⊞ ➪ Operators ➪ /.

■ The Parameters panel opens.

Extra

You can use the `parseFloat` function to convert a string that begins with a number to a floating-point number. A floating-point number is any number with a decimal point. This function is useful when the number you need is in a string. The syntax for the `parseFloat` function is

```
parseFloat(string);
```

Examples:

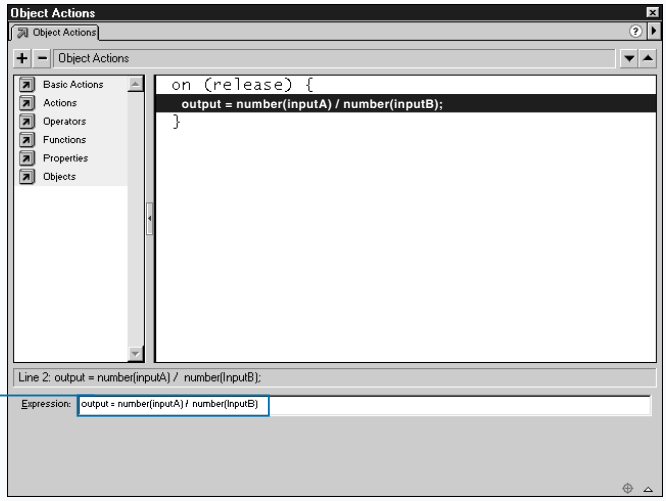
FUNCTION	RETURN
parseFloat("-123")	-123
parseFloat(123.45)	123.45
parseFloat(123abc)	123
parseFloat(abc123)	NaN

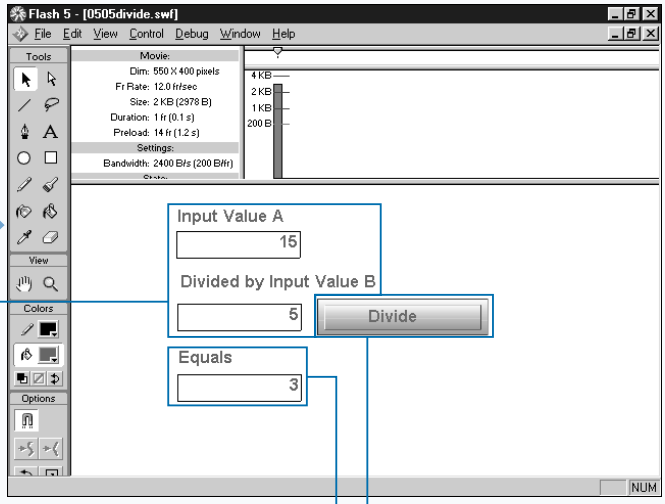
You can use the `parseInt` function to convert a string that begins with a number to an integer. An integer is a number that does not contain a decimal point. This function is useful when the number you need is in a string. The syntax for the `parseFloat` function is

```
parseInt(string);
```

Examples:

FUNCTION	RETURN
parseInt("-123")	-123
parseInt(123.45)	123
parseInt(123abc)	123
parseInt(abc123)	NaN





8 Type the mathematical expression.

■ This example divides inputA by inputB.

9 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

10 Type in values.

11 Click to calculate.

■ ActionScript divides using the entries in the input text boxes and displays the results in the dynamic text box.

FIND THE MODULO

The remainder of one `expression` divided by another `expression` is called the modulo. In ActionScript, you can use the modulo operator (%) to find the remainder. This operator is useful when you are working with numbers or writing scripts. When scripting, you might want a particular action to occur only if a number is evenly divisible by another. The modulo operator is perfect for this. To use the modulo operator, you place the % between two numeric `expressions`. The syntax for the modulo operator is

```
numericExpression1%numericExpression2
```

ActionScript divides `numericExpression1` by `numericExpression2` and returns the remainder, for example: ActionScript divides `13` by `5` and assigns `3`, the remainder, to `x`.

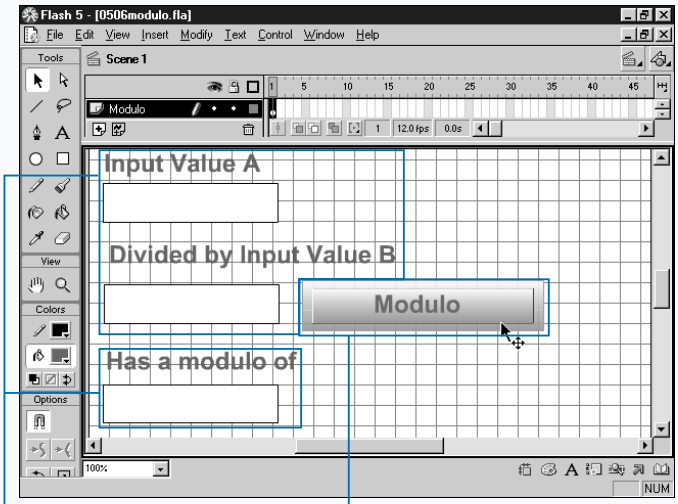
You can use the modulo operator with numbers, variables and numbers, and variables. The `numericExpression` argument represents any number, integer, floating-point number, or string that converts to a number.

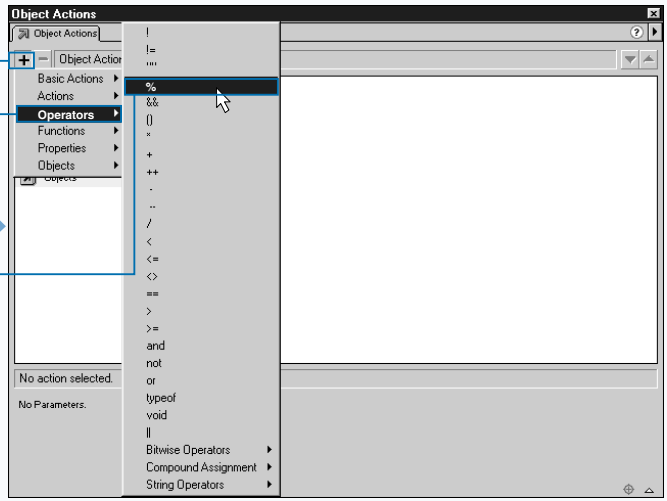
If the `numericExpression` is not a number, ActionScript attempts to convert it to a number.

In general, a leap year occurs in years that are evenly divided by four. You can use the modulo to help you determine if a year is a leap year. You just divide the year by four and if the remainder is 0 the year is a leap year.

You can also use the modulo when you want to execute a statement every nth time. For example, if you want to execute a statement every other time the user presses a button, increment a number — let us say x — each time the user presses the button. Then execute your statement only if the modulo of x divided by 2 is equal to 0.

FIND THE MODULO





1 Open file modulo.fla on the CD-ROM.

2 Create input text boxes.

3 Create a dynamic text box.

4 Create a button.

5 Click to select the button.

6 Open the Object Actions panel.

7 Click ⊞ ➪ Operators ➪ %.

■ The Parameters panel opens.

Apply It

When you use the subtract, multiply, divide, or modulo operators with a string, ActionScript tries to convert the string to a number. When you use the plus operator with a string, ActionScript concatenates. This can produce interesting results when you are performing mathematical calculations using strings. The following script illustrates:

TYPE THIS:

```
on (release) {
    x = "24";
    y = "5";
    a = "10";
    output = x%y+a;
}
```

RESULT

410

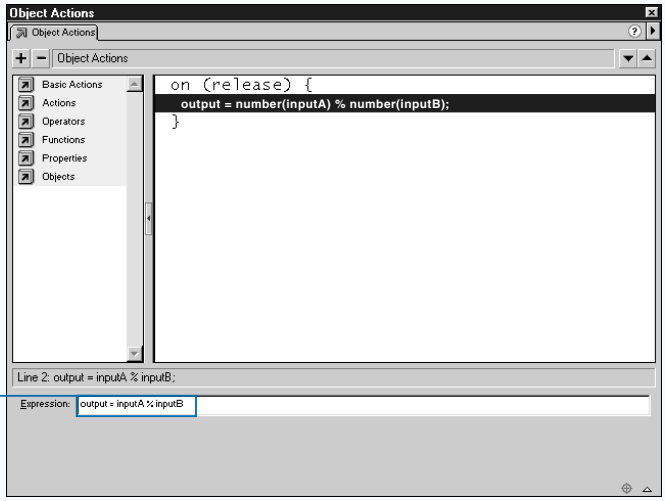
ActionScript finds the modulo of `24` divided by `5`, which is `4`, and then concatenates `10`. To add a, you must convert a to a number. Generally, if you want to perform mathematical calculations with strings, you should convert the strings to numbers. You should also be aware that input text boxes capture numbers as strings.

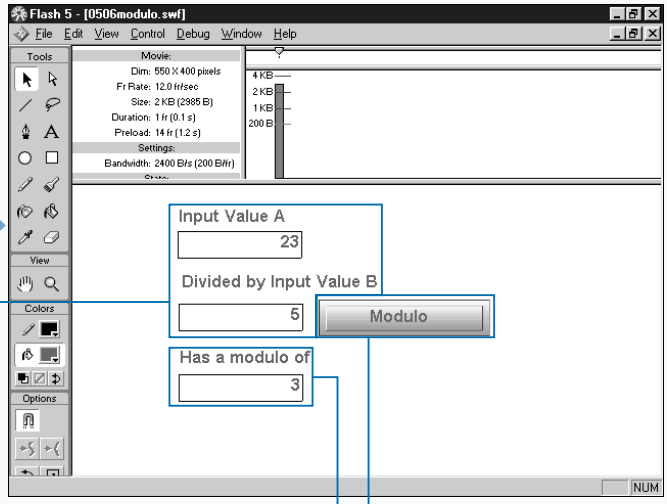
TYPE THIS:

```
on (release) {
    x = "24";
    y = "5";
    a = "10";
    output = number(x) %Number(y)+Number(a);
}
```

RESULT

14





8 Type the mathematical expression.

■ This example finds the modulo of inputA divided by inputB.

9 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

10 Type in values.

11 Click to calculate.

■ ActionScript finds the modulo using the entries in the input text boxes and displays the results in the dynamic text box.

SET VALUES WITH THE INCREMENT OPERATOR

You can use the pre-increment operator (++) to increase the value of an `expression` by `1`. This operator is useful as a counter in a loop. For more about loops, see Chapter 6. The syntax for the pre-increment operator is

```
++expression
```

The `expression` argument represents any numeric, variable, element in an array, or object property. The pre-increment operator adds `1` to the `expression`, assigns `1` plus the `expression` to the `expression`, and returns `1` plus the `expression`. If the `expression` `x` is equal to `4`, `++x` is equal to `5`. The variable `x` is assigned `5`. In the `expression` `y=++x`, `y` is assigned `5`.

Example:

```
x=4;
y=++x;
Result: y is equal to 5, x is equal to 5
z=x;
Result: x is equal to 5, z is equal to 5
```

You can also use the post-increment operator to increase the value of an `expression`. The post-increment operator is similar to the pre-increment operator and also can be used as a counter in a loop. The post-increment operator adds 1 to the `expression`, assigns 1 plus the `expression` to the `expression`, and returns the original value of the `expression`. The syntax for the post-increment operator is

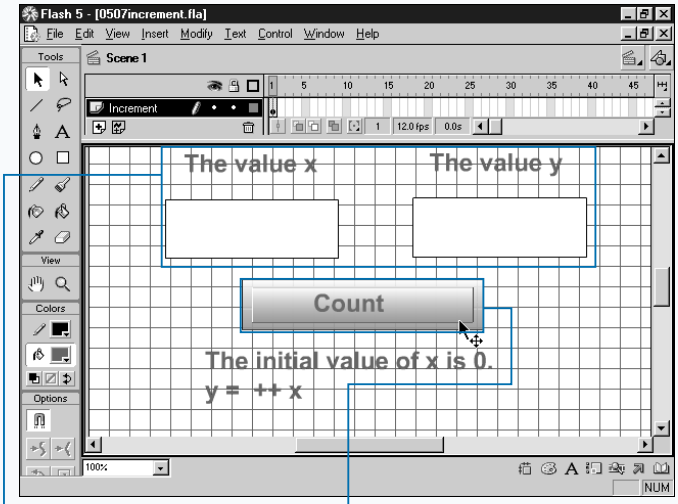
```
expression++
```

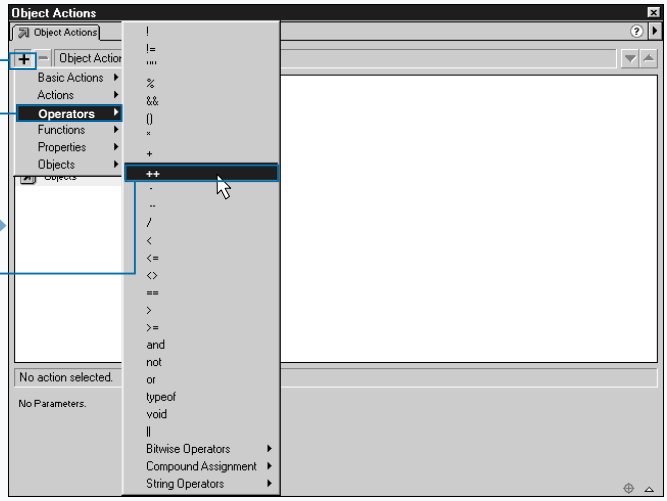
If the `expression` `x` is equal to `4`, `x++` is equal to 5. The variable `x` is assigned `5`. In the `expression` `y = x++`, `y` is equal to `4`.

Example:

```
x=4;
y=x++;
Result: y is equal to 4, x is equal to 5
z=x;
Result: x is equal to 5; z is equal to 5
```

SET VALUES WITH THE INCREMENT OPERATOR





1 Open file increment.fla on the CD-ROM.

2 Create dynamic text boxes.

3 Create a button.

4 Click to select the button.

5 Open the Object Actions panel.

■ The Parameters panel opens.

6 Click ➕ ➪ Operators ➪ ++.

Extra

This script illustrates using the post-increment operator. You attach the script to a button. The `expression x=0` assigns `0` to the variable `x`. You place this statement in the first fame of the movie to initialize the value of `x`. The statement `on (release)` begins the action when the user releases the mouse after clicking the button. The variables `x` and `y` are dynamic text boxes. `x++` increases the value of `x` by `1` and assigns the original value of `x` to `y`.

Frame 1:

```
x = 0;
```

Button:

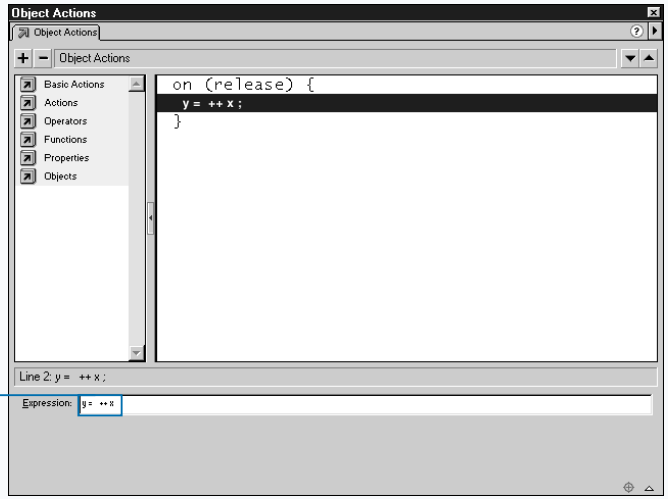
```
on (release) {
    y = x ++;
}
```

The increment operator is a unary operator. Unlike arithmetic operators, it only requires a single operand.

Example

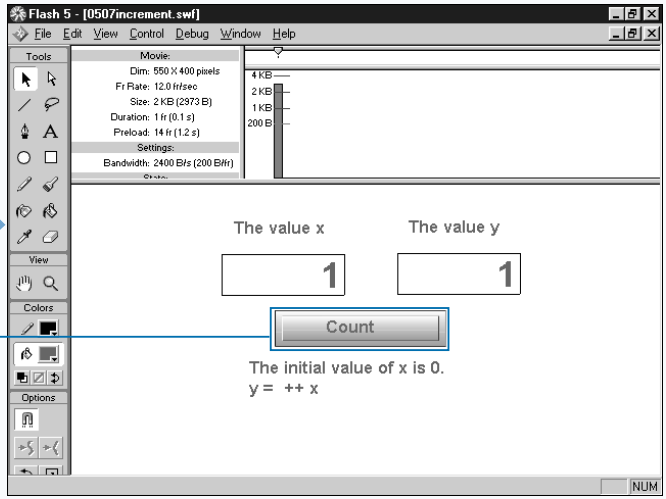
```
++x;
```

The example represents a complete statement.



7 Type the mathematical expression.

■ This example increments x.



8 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

9 Click to calculate.

■ ActionScript uses the pre-increment operator to set the value of x and y. ActionScript displays the results in the dynamic text box.

SET VALUES WITH THE DECREMENT OPERATOR

You can use the pre-decrement operator (--) to decrease the value of an `expression` by `1`. This operator is useful as a counter in a loop. For more about loops, see Chapter 6. The syntax for the pre-decrement operator is

```
--expression
```

The `expression` argument represents any numeric, variable, element in an array, or object property. The pre-decrement operator subtracts `1` from the `expression`, assigns `1` minus the `expression` to the `expression`, and returns 1 minus the `expression`. If the `expression` `x` is equal to `4`, `--x` is equal to `3`. The variable x is assigned 3. In the `expression` `y=--x`, `y` is equal to `3`.

Example:

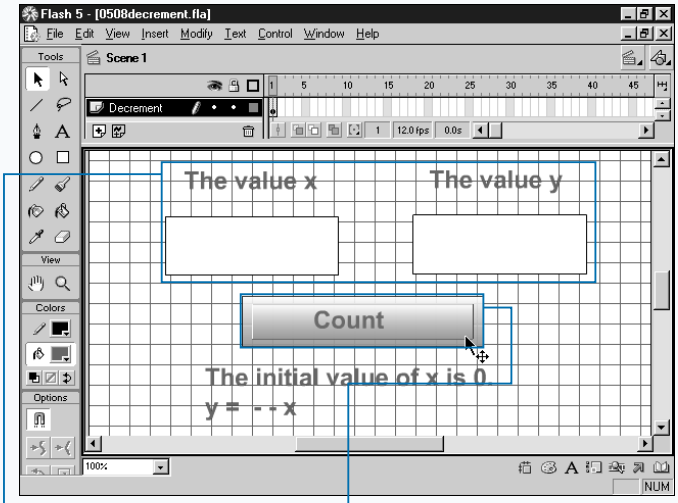
```
x=4;
y=--x;
Result: y is equal to 4, x is equal to 4
z=x;
Result: x is equal to 4, z is equal to 4
```

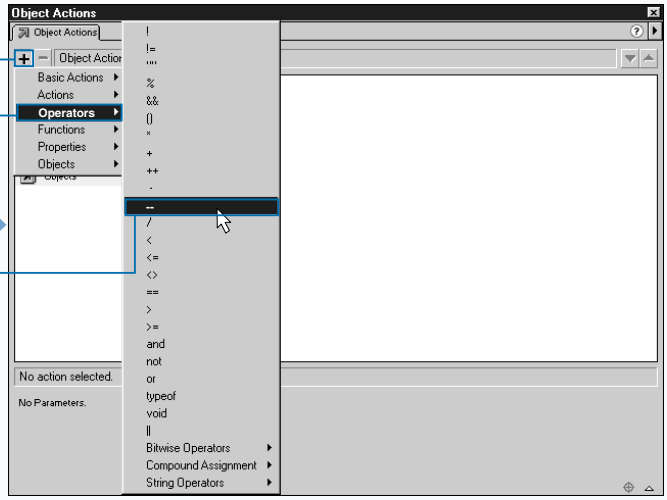
You can also use the post-decrement operator to decrease the value of an `expression`. The post-decrement operator is similar to the pre-decrement operator and also can be used as a counter in a loop. The post-decrement operator subtracts 1 from the `expression`, assigns 1 minus the `expression` to the `expression`, and returns the original value of the `expression`. The syntax for the post-decrement operator is

```
expression--
```

If the `expression` `x` is equal to `4`, `x--` is equal to `3`. The variable x is assigned 3. In the `expression` `y=x--`, `y` is equal to `4`.

SET VALUES WITH THE DECREMENT OPERATOR





1 Open file decrement.fla on the CD-ROM.

2 Create dynamic text boxes.

3 Create a button.

4 Click to select the button.

5 Open the Object Actions panel.

6 Click ⊞ ➪ Operators ➪ -- .

■ The Parameters panel opens.

Apply It

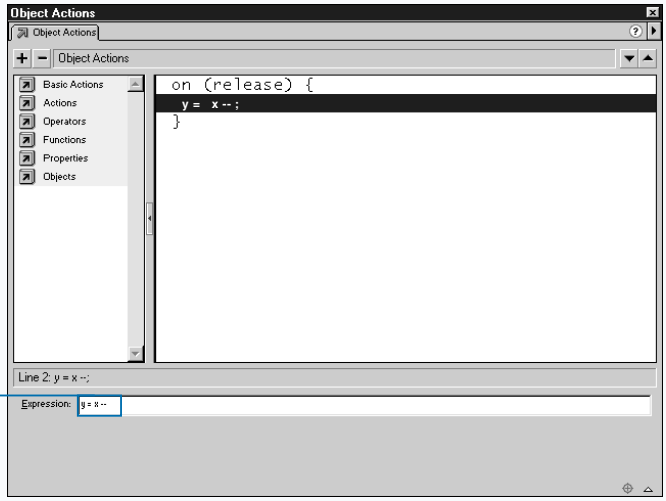
This script illustrates using the post-decrement operator. You attach the script to a button. The `expression x=100` assigns `100` to the variable `x`. You place this statement in the first fame of the movie to initialize the value of `x`. The statement `on (release)` begins the action when the user releases the mouse after clicking the button. The variables `x` and `y` are dynamic text boxes. `x--` decreases the value of `x` by `1` and assigns the original value of `x` to `y`.

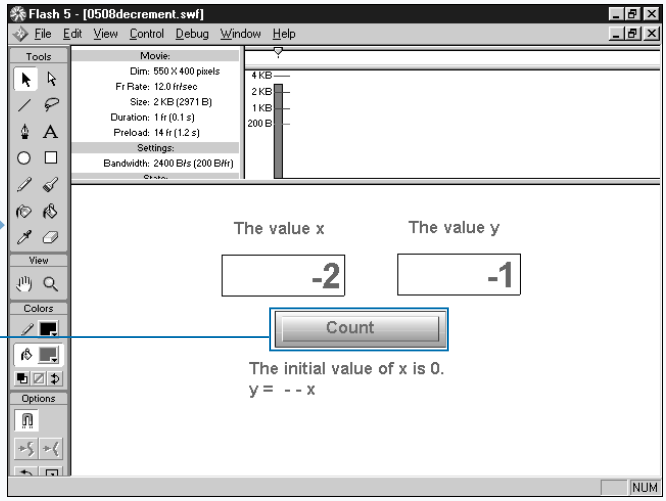
Frame 1:

```
x = 100;
```

Button:

```
on (release) {
    y = x--;
}
```





7 Type the mathematical expression.

■ This example decrements x.

8 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

9 Click to calculate.

■ ActionScript uses the pre-decrement operator to set the value of x and y. ActionScript displays the results in the dynamic text box.

USING LESS THAN OR GREATER THAN

You can use the less than operator (`<`) to compare two `expressions` and determine whether the first `expression` is less than the second `expression`. If `expression1` is less than `expression2`, ActionScript returns the Boolean value `true`. If `expression1` is greater than or equal to `expression2`, Flash returns the Boolean value `false`. The syntax for the less than `expression` is

```
expression1<expression2
```

`Expression` represents any number or string.

You can use the greater than operator (`>`) to compare two `expressions` and determine whether the first `expression` is greater than the second `expression`. If `expression1` is greater than `expression2`, ActionScript returns the Boolean value `true`. If `expression1` is less than or equal to `expression2`, ActionScript returns the Boolean value `false`. The syntax for the greater than operator is

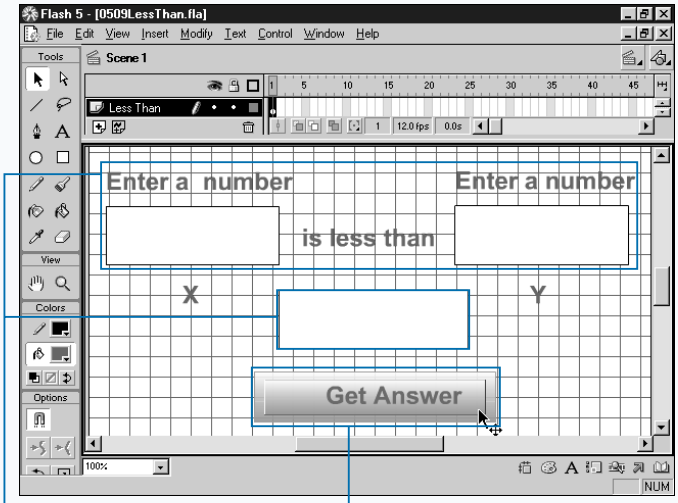
```
expression1>expression2
```

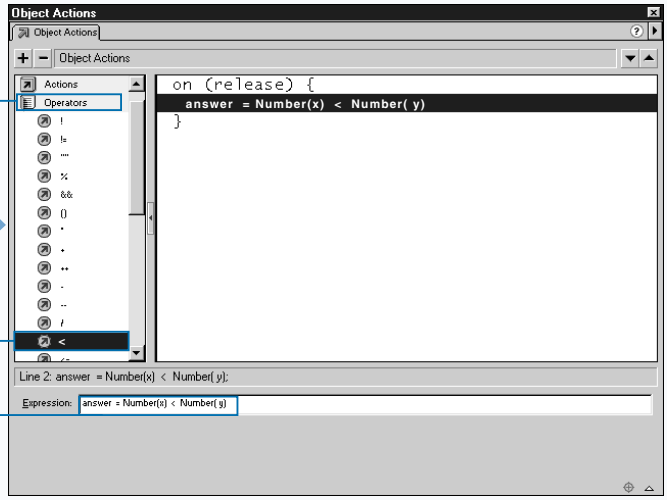
`Expression` represents any number or string.

When used with strings, the less than and greater than operators determine alphabetical order. Comparing the string value of numbers produces unusual results. Use the Number function to convert strings to numbers before doing a comparison. You often use these operators with loops to repeat an action. To learn more about loops, see Chapter 6.

Flash 4 used the `lt` and `gt` operators to compare strings and the < and > operators to compare numbers. Flash 5 deprecates `lt` and `gt`. You should use < and > when authoring for a Flash 5 environment.

USING LESS THAN OR GREATER THAN





1 Open file LessThan.fla on the CD-ROM.

2 Create input text boxes.

3 Create a dynamic text box.

4 Create a button.

5 Click to select the button.

6 Open the Object Actions panel.

7 Click Operators ➪ < to select Less Than.

■ The Parameters panel opens.

8 Type the mathematical expression.

■ This example compares x and y.

■ ActionScript compares the entries in the input text boxes to determine if x < y. ActionScript displays the Boolean result in the dynamic text box.

USING LESS THAN OR EQUAL TO OR GREATER THAN OR EQUAL TO

You can use the less than or equal to operator (<) to compare two `expressions` and determine whether the first `expression` is less than or equal to the second `expression`. If `expression1` is less than or equal to `expression2`, ActionScript returns the Boolean value `true`. If `expression1` is greater than `expression2`, Flash returns the Boolean value `false`. The syntax for less than or equal to is

```
expression1<=expression2
```

`Expression` represents any number or string.

You can use the greater than or equal to operator (>=) to compare two `expressions` and determine whether the first `expression` is greater than or equal to the second `expression`. If `expression1` is greater than or equal to `expression2`, ActionScript returns the Boolean value `true`. If `expression1` is less than `expression2`, ActionScript returns the Boolean value `false`. The syntax for the greater than or equal to operator is

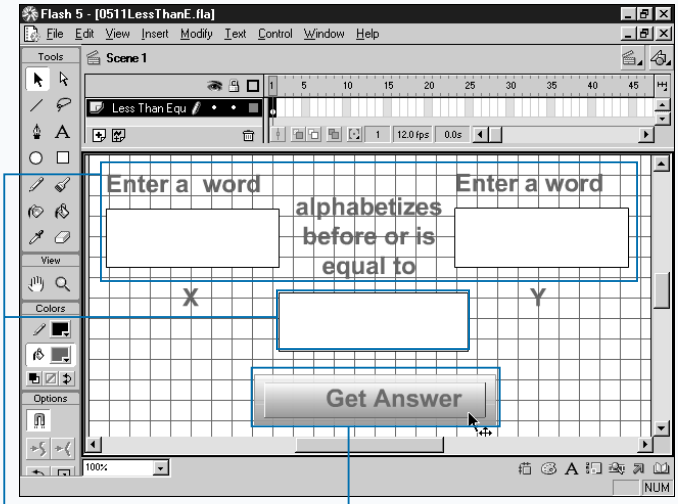
```
expression1>=expression2
```

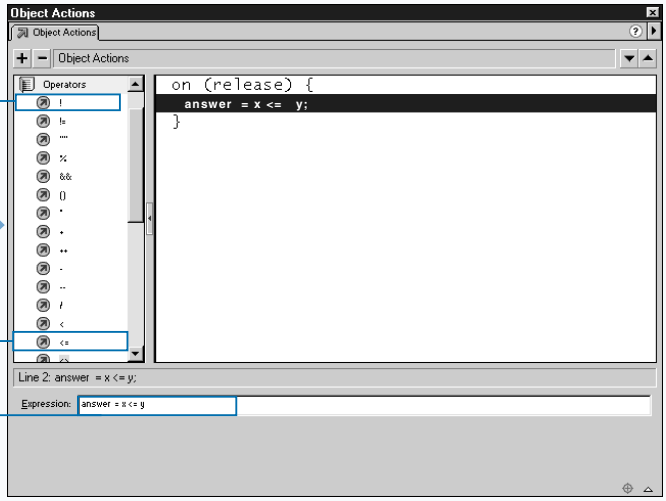
`Expression` represents any number or string.

When used with strings, the less than and equal to and the greater than and equal to operators determine alphabetical order. Comparing the string value of numbers produces unusual results. Use the `Number` function to convert strings to numbers before doing a comparison. You often use these operators with loops to repeat an action. To learn more about loops, see Chapter 6.

Flash 4 used the `le` and `ge` operators to compare strings and the <= and >= operators to compare numbers. Flash 5 deprecates le and ge. You should use <= and >= when authoring for a Flash 5 environment.

USING LESS THAN OR EQUAL TO OR GREATER THAN OR EQUAL TO





1 Open file LessThanE.fla on the CD-ROM.

2 Create input text boxes.

3 Create a dynamic text box.

4 Create a button.

5 Click to select the button.

6 Open the Object Actions panel.

7 Click Operators ➪ <= to select Less Than Or Equal To.

■ The Parameters panel opens.

8 Type the mathematical expression.

■ This example compares two numbers.

■ ActionScript compares the entries in the input text boxes to determine if x <= y. ActionScript displays the Boolean result in the dynamic text box.

USING EQUALITY AND INEQUALITY OPERATORS

When writing ActionScript, you will at times want to know whether two `expressions` are equal. The equality operator (`==`) compares numbers, strings, Booleans, objects, or movie clips and returns `true` if they are equal and `false` if they are not. The equality operator uses this syntax:

```
expression1 == expression2
```

To use the equality operator, place the operator between the `expressions` you want to compare. ActionScript compares strings, numbers, and Booleans by value. It compares objects, movie clips, and arrays by reference. String values are equivalent if they are identical. Both strings must have exactly the same characters in exactly the same position. Numbers and Booleans are equivalent if the `expression` evaluates to the same value. For example, the `expression 3 + 1` returns `4`, and the `expression 2 + 2` returns `false`. Hence, in the example shown here, `x` and `y` are equal, and `z` is equal to `true`.

Example:

```
x=3+1
y=2+2
z=x==y
```

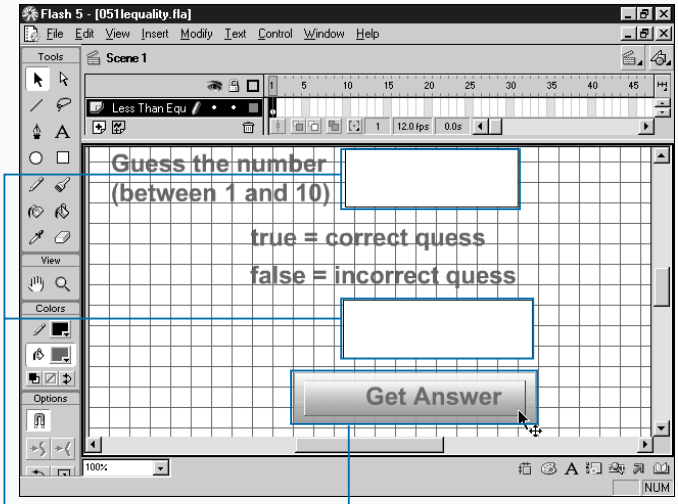
For more on the `Boolean` function, see Chapter 4.

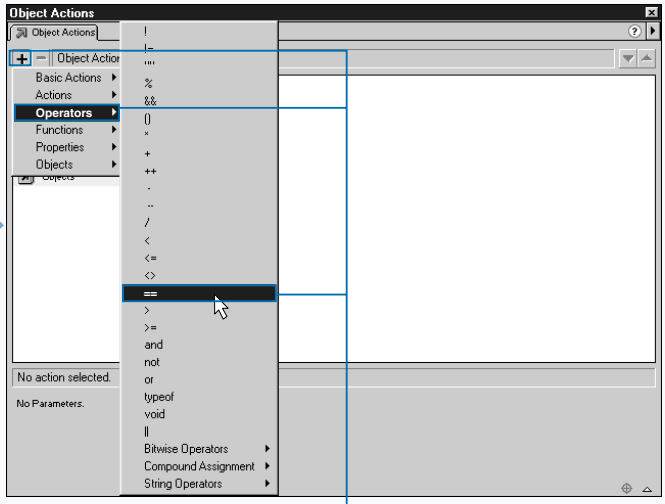
When working with objects or arrays, ActionScript compares by reference. It examines the array, object, or storing variable and returns `true` if the compared `expressions` refer to the same object or array. ActionScript never considers two separate arrays equal, even if they contain exactly the same elements with exactly the same values. ActionScript never considers two separate objects equal, even if they are exactly alike.

The inequality operator `!=` has the same characteristics as the equality operator except it returns `true` if the `expressions` are not equal and `false` if they are equal. The inequality operator uses the syntax shown here:

```
expression1 != expression2
```

USING EQUALITY AND INEQUALITY OPERATORS





1 Open file equality.fla on the CD-ROM.

2 Create input text boxes.

3 Create a dynamic text box.

4 Create a button.

5 Click to select the button.

6 Open the Object Actions panel.

7 Click ⊞ ➪ Operators ➪ ==.

■ The Parameters panel opens.

Extra

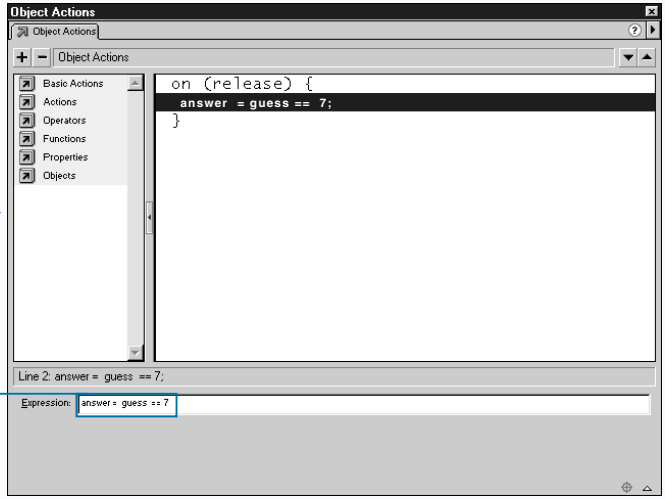
Flash 4 used <> as the inequality operator. The characteristics of the <> operator and the != operator are the same. The <> operator uses this syntax:

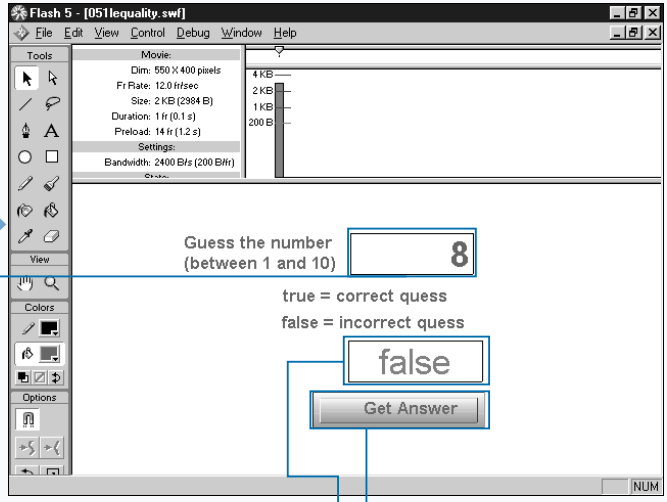
```
expression1<>expression2
```

Flash 5 deprecates <>. You should use != if you are authoring for a Flash 5 environment.

You will frequently use the equality and inequality operators with the if action. Together, the if action and the equality operators give you the ability to evaluate whether a condition is `true` or `false` and to perform an action based on the results. For more on the `if` action, see Chapter 6.

You should not confuse the == operator with the = operator. The = operator assigns the expression on the right side of the operator to the variable on the left side of the operator. The == operator compares two values and returns true if they are equal, false if they are not.





8 Type the mathematical expression.

■ This example compares two values.

9 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

10 Type a response.

11 Click to get a response.

■ ActionScript compares your entry to the value it has and displays in the dynamic text box the Boolean true if they are equal, false if they are not equal.

USING COMPOUND ASSIGNMENT OPERATORS

Compound assignment operators provide you with another method to assign a value to a variable. Compound assignment operators are frequently used in loops to increase the value of the loop. Compound assignment operators use the value of the operand on the left side of the `expression` in the calculation. The syntax for compound assignment operators is

```
operator=
```

The `plus assignment operator` adds the `expression` on the right of the operator, to the `expression` on the left. For example, `x+=5` adds `5` to `x`. The expression `x+=5` also assigns `5+x` to `x`.

The `minus assignment operator` subtracts the `expression` on the right, from the `expression` on the left and assigns the result to the `expression` on the left. All of the other compound assignment operators work in a similar fashion.

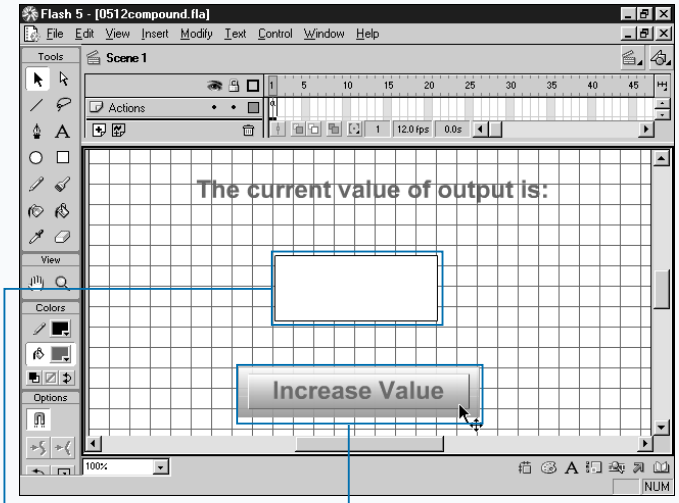
This example uses the compound plus assignment operator:

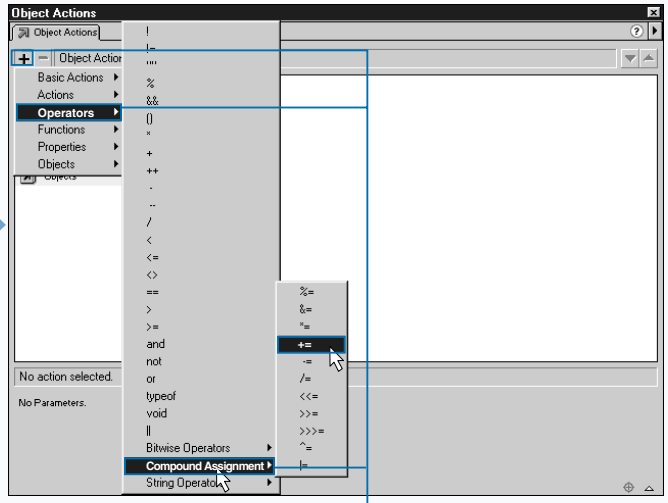
```
x=20
x+=10
y=x
```

ActionScript adds `10` to `x`, which yields `30`. ActionScript assigns `30` to `x`. ActionScript assigns `x` to `y`. So, `y` is equal to `30`.

ActionScript supports all of the compound assignment operators listed on the next page. When using a compound assignment operator, the process is always the same: Use the value of the operand on the right side of the assignment operator, and perform the operation indicated on the `expression` on the left side of the assignment operator.

USING COMPOUND ASSIGNMENT OPERATORS





1 Open file compound.fla on the CD-ROM.

2 Create a dynamic text box.

3 Create a button.

4 Click to select the button.

5 Open the Object Actions panel.

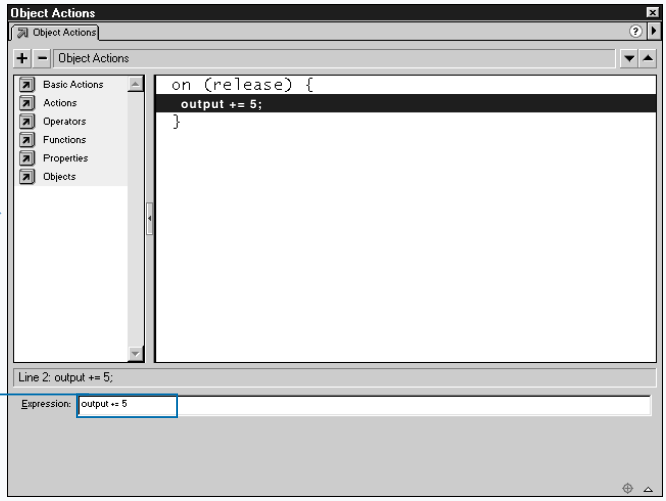
6 Click ⊞ ➪ Operators ➪ Compound Assignment ➪ +=.

■ The Parameters panel opens.

Extra

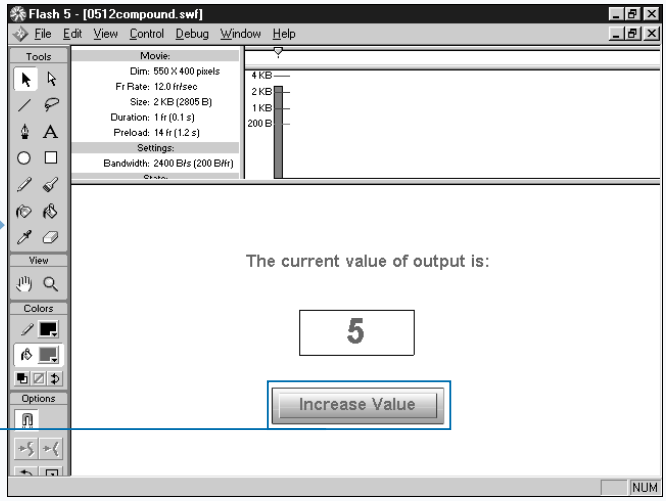
There are several compound operators. Each of them works like the addition assignment operator. They use the value on the left side of the `expression` and the value on the right side of the `expression` to calculate the assigned value. The addition operator adds the values, the subtraction operator subtracts the values, and so on for each of the values listed. The following lists compound operators:

OPERATOR	FUNCTION
+=	Addition and assignment
-=	Subtraction and assignment
*=	Multiplication and assignment
/=	Division and assignment
%=	Modulo and assignment
<<=	Bitwise shift right and assignment
>>=	Shift right zero fill and assignment
^=	Bitwise Xor and assignment
|=	Bitwise Or and assignment
&=	Bitwise And and assignment



7 Type the mathematical expression.

■ This example uses the assignment operator.



8 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

9 Click to calculate.

■ ActionScript adds 5 to output each time you click the button and displays the result in the dynamic text box.

USING LOGICAL OPERATORS

You can use the logical operators logical AND and logical OR to compare two Boolean values and return a third Boolean value. Logical operators enable you to evaluate multiple conditions. They are often used with if statements. ActionScript uses && as the logical AND operator and || as the logical OR operator. To use the logical AND or logical OR operator, place the operator between two expressions that can evaluate to either true or false. The syntax for the logical operators is

Logical AND

```
expression1 && expression2
```

Logical OR

```
expression1 || expression2
```

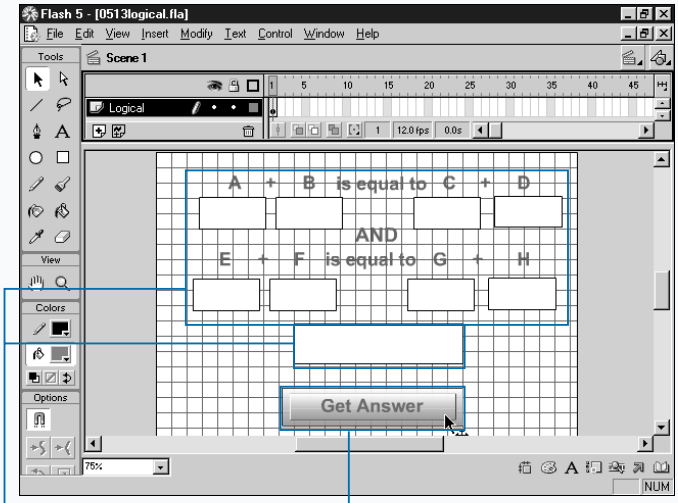
An expression is any statement that can evaluate to true or false.

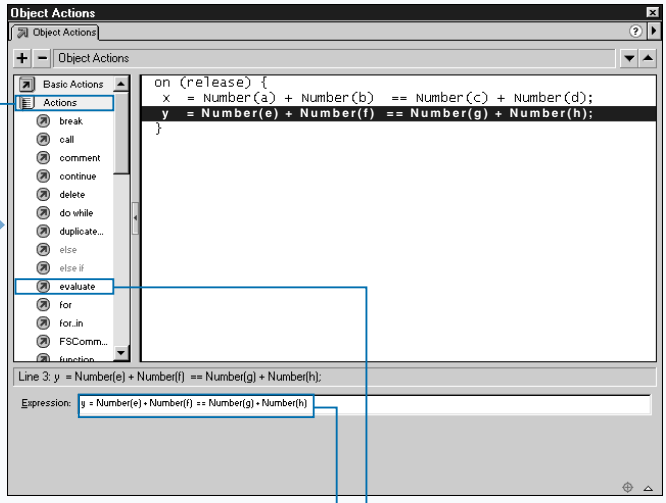
When you use the logical AND, if `expression1` evaluates to `true` and `expression2` evaluates to `true`, the logical AND returns `true`. If `expression1` or `expression2` evaluates to `false`, the logical AND returns `false`. When you use logical OR, if `expression1` or `expression2` evaluates to `true`, the logical OR returns `true`. If `expression1` and `expression2` evaluate to `false`, the logical OR returns `false`.

ActionScript uses the logical NOT to reverse the value of an expression. If an expression evaluates to true, placing an! in front of the expression causes it to evaluate to false. If an expression evaluates to false, placing a ! in front of the expression causes it to evaluate to true. The syntax for the logical NOT expression is

```
! expression
```

USING LOGICAL OPERATORS





1 Open file logical.fla on the CD-ROM.

2 Create input text boxes.

3 Create a dynamic text box.

4 Create a button.

5 Click to select the button.

6 Open the Object Actions panel.

7 Click Actions.

8 Double-click Evaluate.

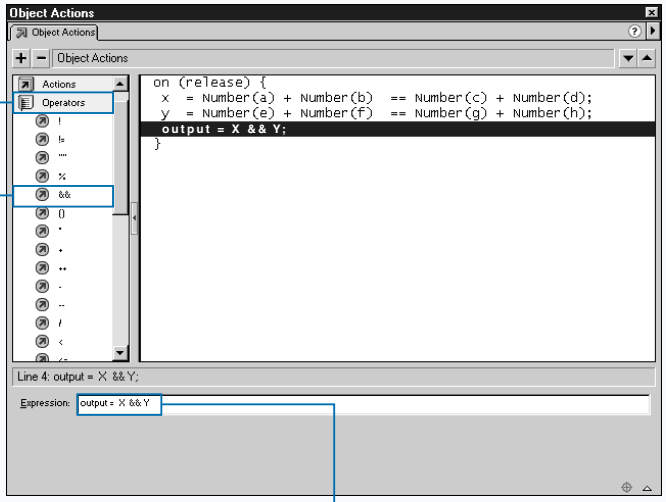
■ The Parameters panel opens.

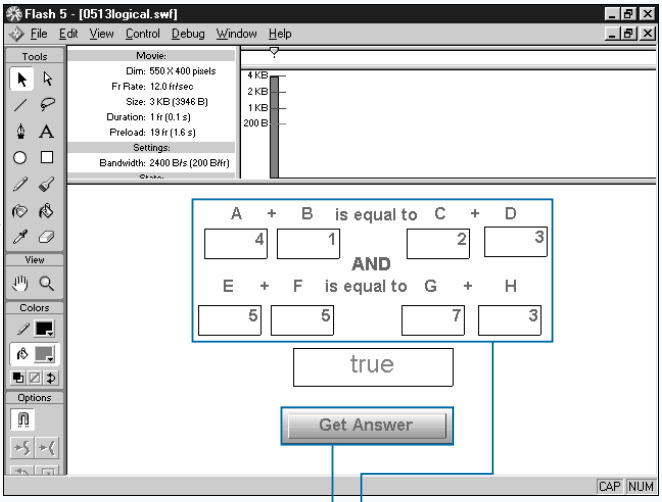
9 Type the conditions to be evaluated.

Extra

The following table provides a handy reference. It lists how ActionScript evaluates logical operators.

EXPRESSION	RETURN
true && true	true
true && false	false
false && true	false
false && false	false
true || true	true
true || false	true
false || true	true
false || false	false
!true	false
!false	true





10 Click Operators.

11 Double-click &&.

■ The parameters panel opens.

12 Type the logical operation.

13 Move to the test environment.

Note: See Chapter 1 for instructions on how to create a button.

14 Type equations.

15 Click to get a response.

■ ActionScript evaluates the values and displays Boolean true if they are true, false if they are not true.

USING IF TO TEST A CONDITION

When driving, each time you reach a stoplight you must make a decision. If the light is green, go. If the light is yellow, proceed with caution. If the light is red, stop. The ActionScript `if`, `else if`, and `else` actions give you the ability to have your script make similar decisions.

Say you need an ActionScript that requires the user to guess a number. If the user guesses the correct number, the script should send a message to the user that says, You are amazing! You can use the `if` action to create your script. The syntax for the `if` action is

```
if (condition){
statement;
}
```

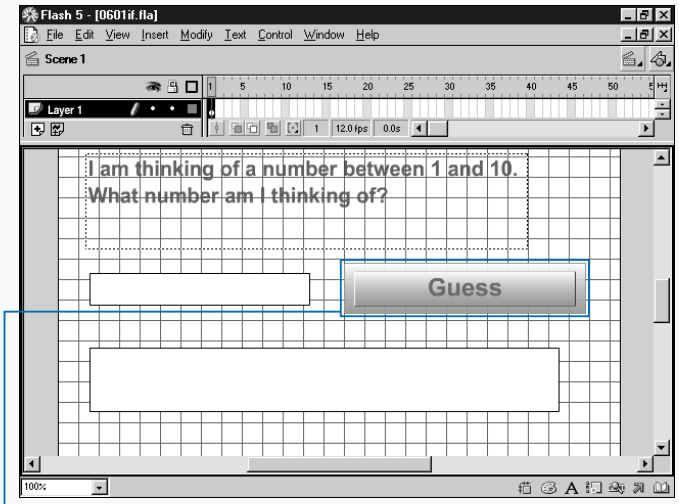
The `if` action takes two arguments, a `condition` and a `statement`. The condition is an `expression` that evaluates to either `true` or `false`. The statement is the instruction to be executed if the condition is `true`. The `if` action evaluates the `condition`. If the `condition` is `true`, ActionScript executes the statement. If the `condition` is not `true`, ActionScript executes the next statement outside of its block of code, for example:

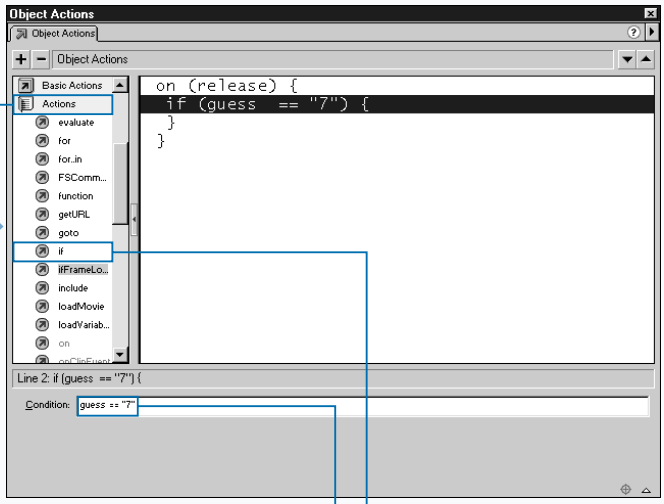
```
on (release) {
if (guess  == "7") {
message = "You are amazing!";
}
}
```

The user makes an entry in the input text box called `guess`. ActionScript compares the value of `guess` with `7`. If `guess` is equal to `7`, ActionScript displays the message "You are amazing."

You use the `if` statement anytime you want to execute a statement only if a specific condition is met. In the above example the script executes when guess is equal to 7. It could be set to execute when guess is greater than 7, less than 7, or some other condition.

USING IF TO TEST A CONDITION





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file if.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Open the Object Actions panel.

3 Click Actions.

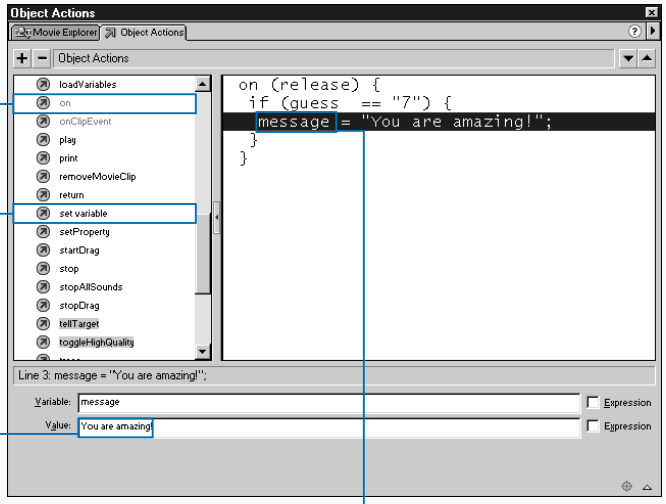
4 Double-click if.

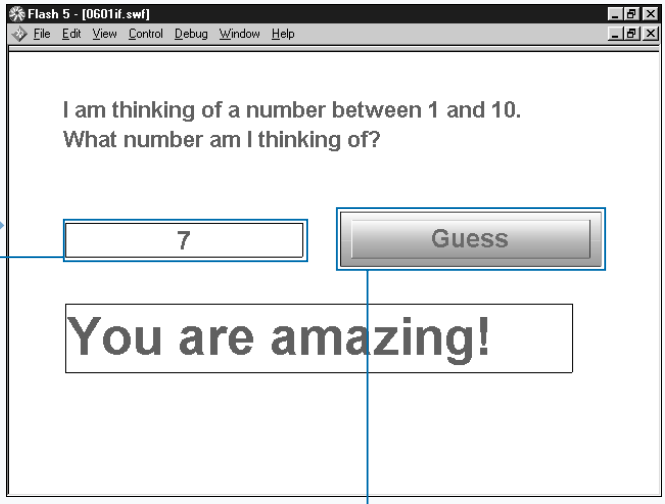
5 Type the condition ActionScript will evaluate.

Extra

When you are working with `if` statements, the operators shown in this table are extremely helpful. For more about operators, see Chapter 5.

OPERATOR	USE
>	Less than
<	Greater than
==	Equal to
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal





6 Click Actions.

7 Double-click set variable.

8 Assign a value to the variable.

■ Associating a variable with a dynamic text box enables you to display the value the variable returns to the user.

9 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

10 Type a number.

11 Click to get a response.

■ ActionScript uses the `if` action to evaluate your entry and responds with a message if the value you enter is the value for which it is looking.

USING IF WITH ELSE

If you have a movie that requires the user to enter a number that is exactly five characters long, you can write a script that uses the `if` action with the `else` action to evaluate the entry. If the entry is five characters long, the movie can send a message to the user that says, Good job. If the entry is not five characters long, the movie can send a message that says, Try again.

The `if` action takes two arguments, a `condition` and a `statement`. The `condition` is an expression that evaluates to either `true` or `false`. The `statement` is an instruction to execute if the condition is `true`. The syntax for the `if` action is

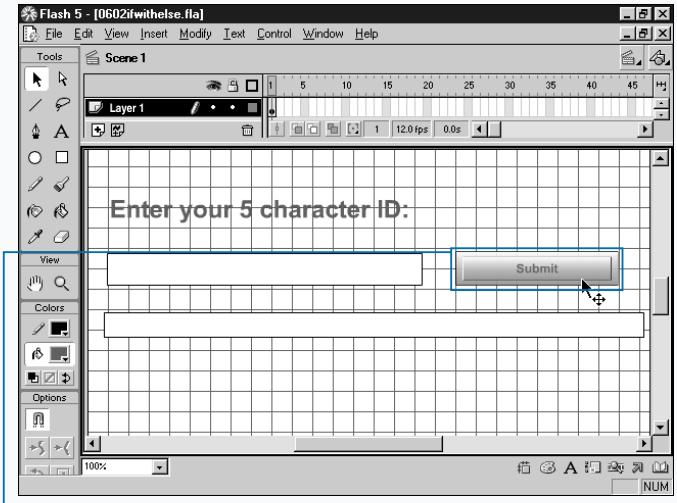
```
if (condition) {
statement(s);
}
```

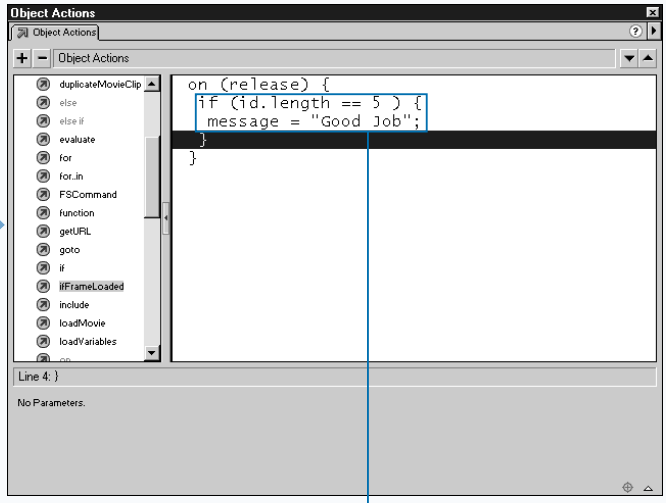
The `else` action takes one argument: `statement`. The `statement` specifies the actions or other conditionals to run if all the other `if` statements return false.

You can use the `if with else` in you code to create a toggle button. With a toggle button you press the button to turn something on and then you press the button again to turn something off. You can create a sound toggle button. You press the button once and the sound starts, you press the button again and the sound stops.

Your code would work something like this. You would set a value to `false` if the music is off. Then you would use an `if` statement to test the value. If the value is `false` you would turn the sound on and set the value to `true`. Your `else` statement would turn the music off whenever the value is not equal to `false`.

USING IF WITH ELSE





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file ifelse.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Open the Object Actions panel.

3 Set the `if` conditions.

Extra

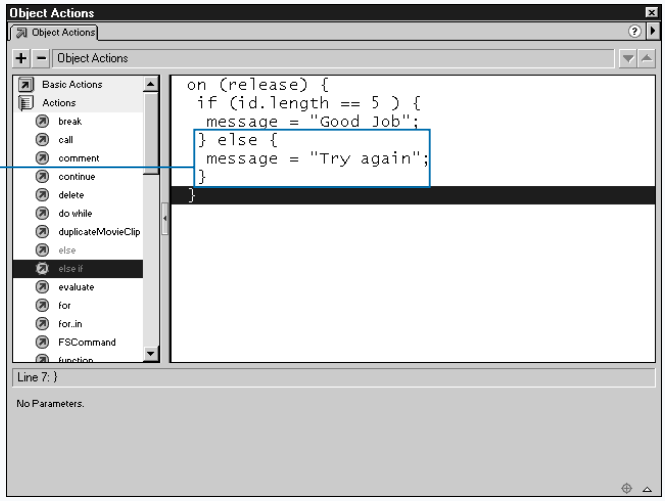
You can use the conditional operator to perform an `if else` evaluation. The conditional operator evaluates `expression1`. If `expression1` evaluates to `true`, the conditional operator executes `expression2`. If `expression1` evaluates to `false`, the conditional operator executes `expression3`. The syntax for the conditional operator is

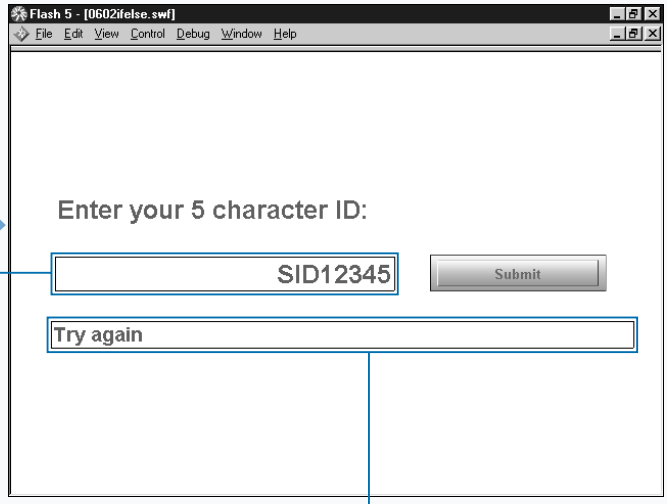
```
expression1?expression2:expression3
```

Example:

```
id.length == 5?message = "Good
job":message = "Try again";
```

In the example, if the length of `id` is equal to `5`, the message field is assigned Good job. If the `length` of `id` is not equal to `5`, the message field is assigned Try again.





4 Set the `else` conditions.

■ Associating a variable with a dynamic text box enables you to display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Test your movie by typing a number.

7 Click to get a response.

■ ActionScript evaluates your entry to see if it is five characters long and responds with Good job if it is, or Try again if it is not.

USING ELSE IF

If you have a situation with multiple possibilities and you want a different action to execute, depending on the condition, use `else if`. You use `else if` when you want your script to respond one way if the condition is A, another way if the condition is B, and another way if the condition is C.

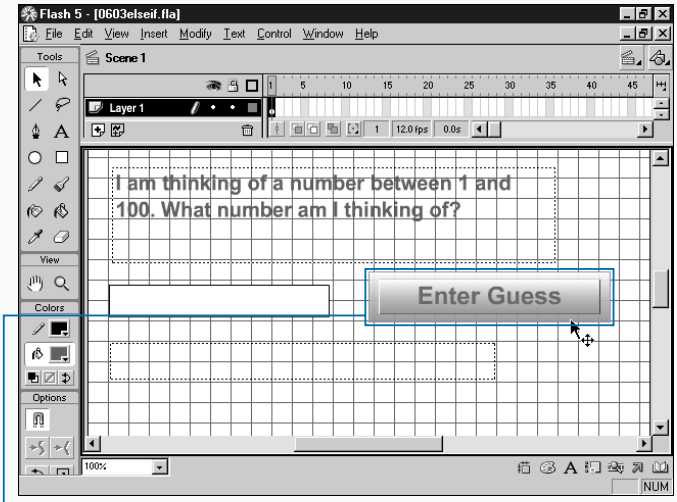
For example, you need a script that requires the user to guess a number. If the user entry is higher than the correct number, the script sends a message to the user that says Lower. If the entry is lower than the correct number, the script sends a message to the user that says Higher. If the entry is correct, the script sends a message to the user that says, You are amazing!

You can use the `else if` action to create your script. The `else if` action evaluates the first condition and executes the statement if the condition is `true`. If the condition is `false`, it evaluates the next condition and executes it if it is `true`. It continues to evaluate statements until it finds a condition that is `true` or there are no more conditions to evaluate. The syntax for the else if action is

```
if (condition) {
statement(s);
}else if (condition) {
statement(s);
}
```

You can can use `else if` whenever you provide the user with several options. For example, you provide the user with options red, green, yellow, and blue. If the user selects red, you can have an object on the Stage turn red. If the user select green, you can have an object on the Stage turn green, and so forth.

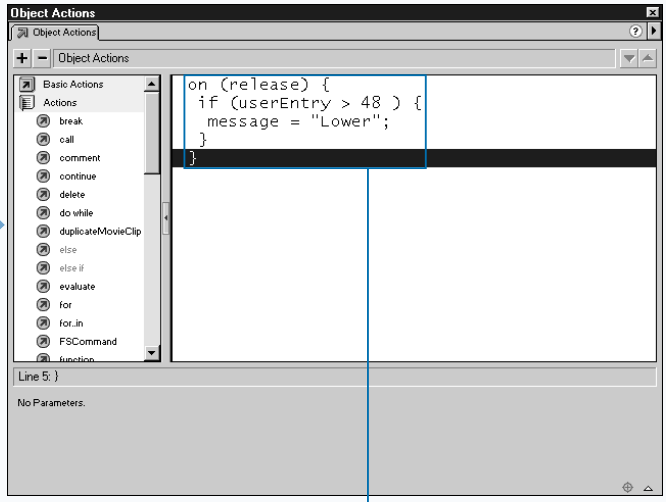
USING ELSE IF



1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file elseif.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.



2 Open the Object Actions panel.

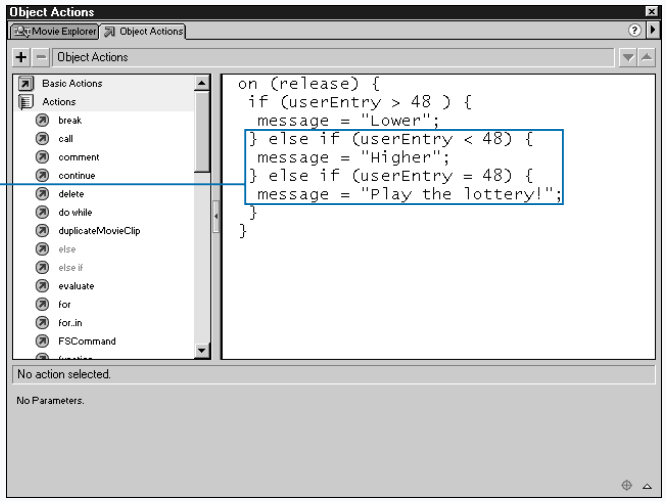
3 Type the statement you want to evaluate using `if`.

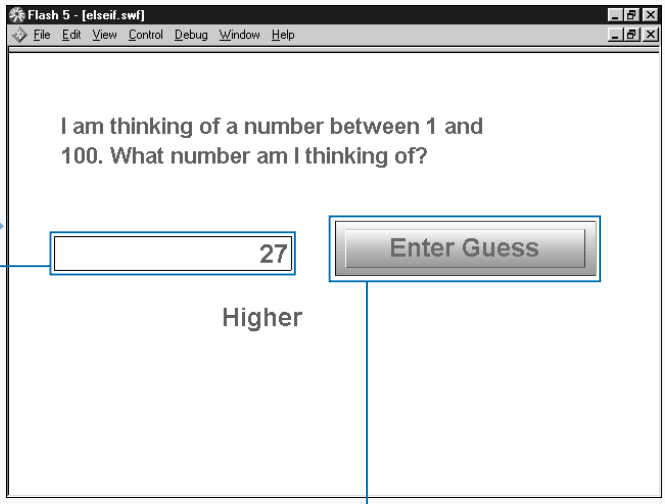
Extra

You can nest your `if` statements. Nesting your `if` statement causes ActionScript to evaluate each nested statement to see if it is true. The following would be true if `guess` is equal to `7`, `tries` is equal to `10`, and `correct` is equal to `10`.

Example:

```
on (release) {
    if (guess  == "7") {
        if (tries == 10) {
            if (correct == 10) {
            }
        }
        message = "AMAZING!";
    }
}}
```





4 Type your `else` statements.

■ Associating a variable with a dynamic text box enables you to display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Type a number to test your movie.

7 Click to get a response.

■ ActionScript evaluates your entry and responds with a message.

CREATE A CONDITIONAL LOOP

When you need to execute ActionScript `statement` or a series of ActionScript `statements` several times, you can use the while action. The `while` action enables you to run `statements` repeatedly. The syntax for the `while` action is

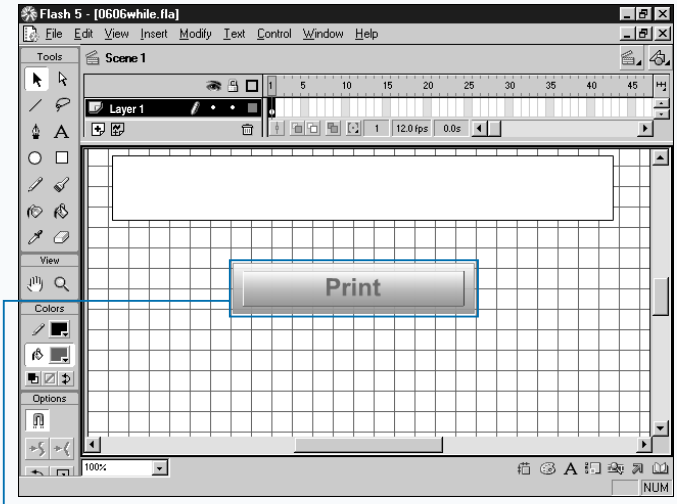
```
while(condition) {
statement(s);
}
```

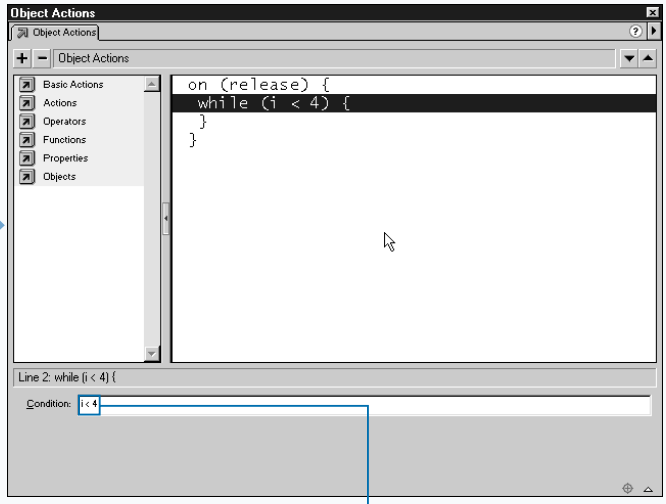
The `condition` is an expression that evaluates to either `true` or `false`. The `statement` is an instruction to execute if the condition is `true`. When ActionScript encounters the `while` action, it evaluates the `condition`. If the condition is `true`, it executes the `statements`. After all the `statements` have been executed, ActionScript returns to the `while` and evaluates the `condition` again. If the condition is still `true`, it executes the `statements`. If the `condition` is `false`, ActionScript executes the first `statement` after the `while` action.

The `while` loop must include a `statement` that changes the `condition` and the `condition` must eventually evaluate to `false`, or the loop will continue infinitely. Programmers refer to this situation as an infinite loop. Often, you use a counter with a loop. For example, you can use the increment operator with a loop and have the loop continue for as long as a value is less than the specified amount. The example that follows uses the increment operator to increase the value of `x` after each execution of the statements. For more on the increment operator, see Chapter 5.

```
x = 0
while (x < 10;) {
your statements;
++x;
}
```

CREATE A CONDITIONAL LOOP





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file while.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Open the Object Actions panel.

3 Type your `while` condition.

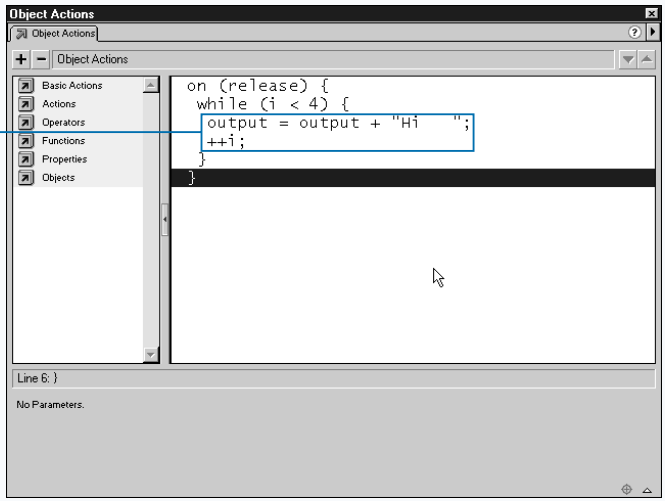
Extra

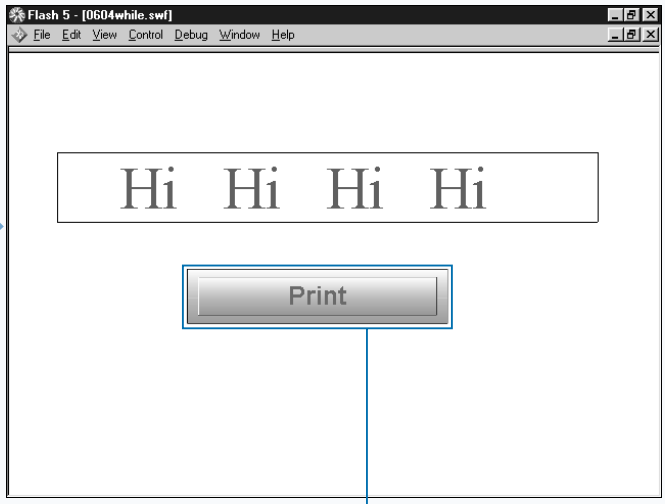
You can also use the `do...while` action to create a loop. The `do...while` action creates a loop like the `while` action except the condition is evaluated at the end of the block of code so that the loop always runs at least once. The syntax for the `do...while` loop is

```
do {
statement;
while (condition);
```

You can place loops inside of loops or use `if` statements in loops. Your scripts can become extremely complex. If you need to break out of a loop, use the `break` action. The `break` action instructs ActionScript to stop executing the loop and go to the next statement.

You can also use the `continue` action. With `while` and `do...while` statements, the `continue` action causes ActionScript to go to the `condition` argument and test whether the `condition` is still `true`.





4 Type your statements.

■ Associating a variable with a dynamic text box enables you to display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button.

■ ActionScript concatenates the word Hi four times.

USING FOR LOOPS

If you have a statement or series of statements that you want to execute repeatedly, let us say ten times, you can write that line of code ten times in a row, and it will executes ten times in a row. Or, you can use a `for` action to execute a statement or series of statements repeatedly. When using the `for` action, you use a counter to specify the number of times the loop will execute. The syntax for the `for` action is

```
for (init; condition; next);{
statement;
}
```

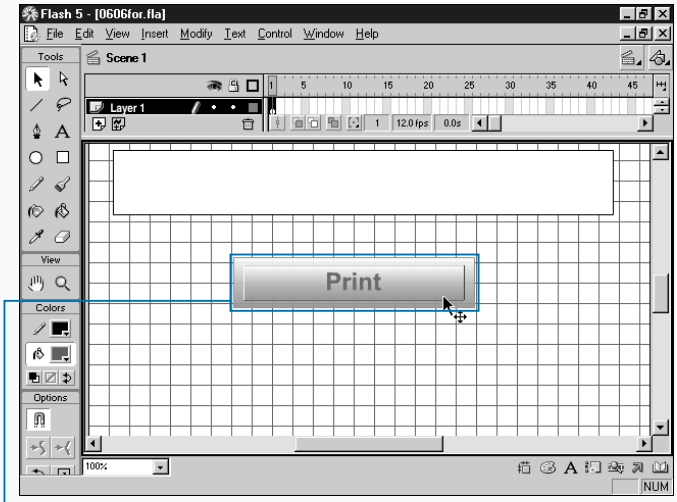
You use the `init` argument to set the initial value of the counter. You use the `condition` argument to set a condition that evaluates to either `true` or `false`. ActionScript evaluates the `condition` on each iteration of the loop. If the condition is `true`, the loop continues. If the condition is `false`, ActionScript goes to the next block of code. You use the `next` argument to reset the counter after each iteration. The `statement` argument is an instruction to execute if the condition is `true`, for example:

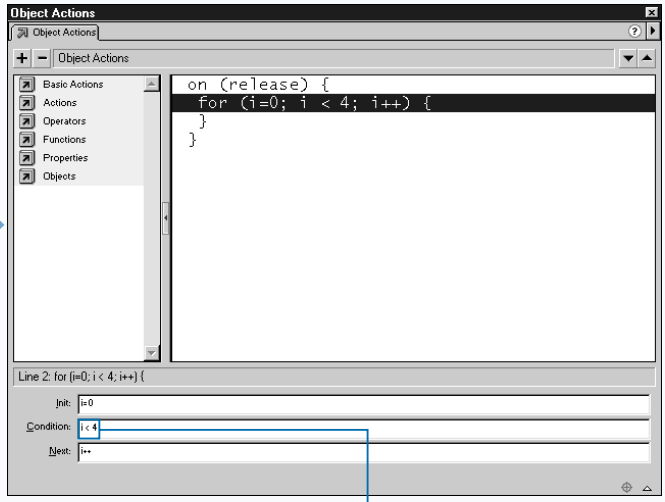
```
for(i=0; i<10, ++i) {
your statements;
]
```

In the example, the expression `i=0` starts the counter with a value of `0`. The expression `i<10` continues executing the statements as long as `i` is less than `10`. The expression `++i` adds one to the value of `i`. The expression `++i` uses the `pre-increment` operator. For more on the `pre-increment` operator, see Chapter 5.

You can use the `for` action when you are making several copies of a movie clip. Just use the code for duplicating a movie clip in the your statements section. Use the condition to specify the number of times you want the movie clip duplicated.

USING FOR LOOPS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file for.fla, which you can find on the CD-ROM that accompanies this book.

■ This example uses a button.

2 Open the Object Actions panel.

3 Type your `for` condition.

Extra

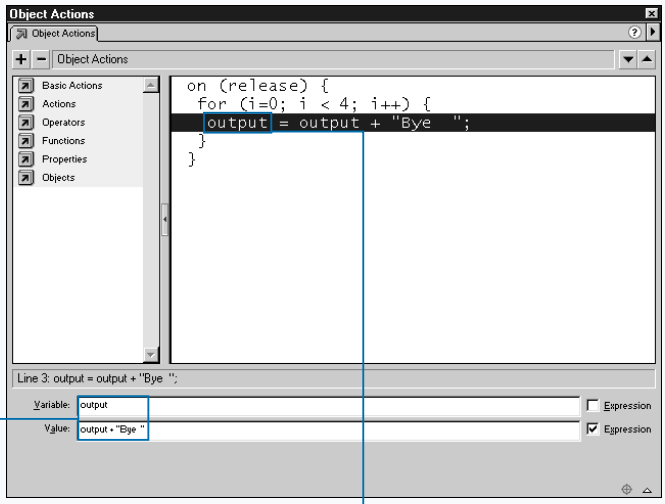
When using a for loop, the `init`, `condition`, and `next` arguments are optional. However, if you do not include them in the `for` statement, you must set them somewhere else in your script. If you omit the init argument, you must still include the semicolon.

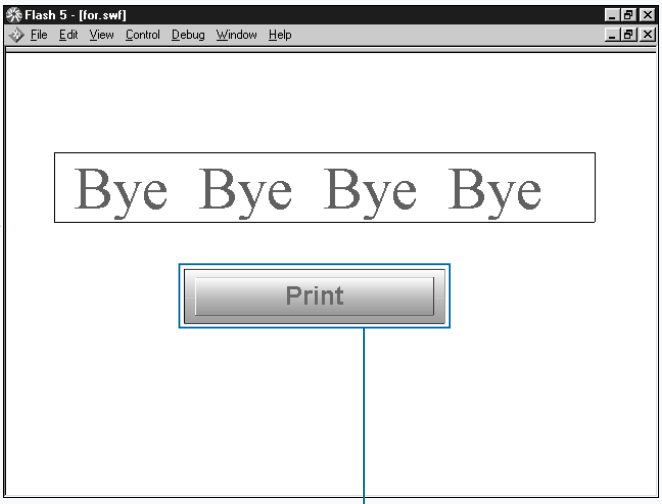
Example

```
i = 1
for(; i<10, ++i) {
your statements;
]
```

Failing to include a condition argument in your script can cause an infinite loop because ActionScript will assume that the condition is always `true`.

Your script can become extremely complex and you may be resetting the condition within the loop. You can use the `continue` operator to break out of the loop, go to the `condition` operator and determine if the loop should continue.





4 Type your statements.

■ Associating a variable with a dynamic text box enables you to display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button.

■ ActionScript concatenates the word Bye four times.

INTRODUCTION TO OBJECTS

Objects enable you to access information, or they graphically represent movie clips on the Stage. Flash predefines objects and you can create your own objects. Most of the objects predefined by Flash have methods. Methods allow you to obtain values or perform actions. Objects can also have properties.

ActionScript predefines the following objects: `Array`, `Boolean`, `Color`, `Date`, `Key`, `Math`, `MovieClip`, `Number`, `Object`, `Selection`, `Sound`, `String`, `XML`, and `XML socket`. Each object enables you to access certain types of information or perform specific actions. For example, the `Math` object has methods that help you perform mathematical calculations. The `MovieClip` object has methods that enable you to perform actions on movie clips.

You can create a new object using the `new` operator. A constructor function is a function used to create objects. You can use the `new` operator to create an object for a predefined object class. When using the `new` operator you must use it with a constructor function. Every object predefined by Flash is also a constructor function.

The syntax for the `new` operator is

```
new constructor().
```

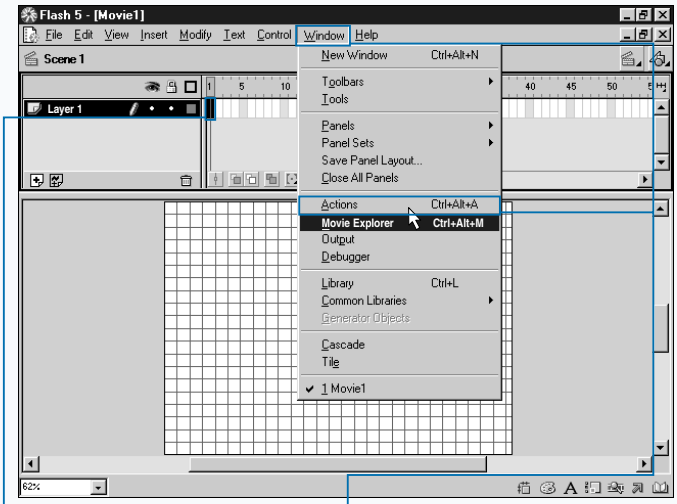
Use the constructor argument to specify the constructor function or the type of object to be constructed. Examples include `Color`, `Sound`, and `Date`. You pass arguments by placing them between the parentheses. The syntax that follows creates a `Color` object:

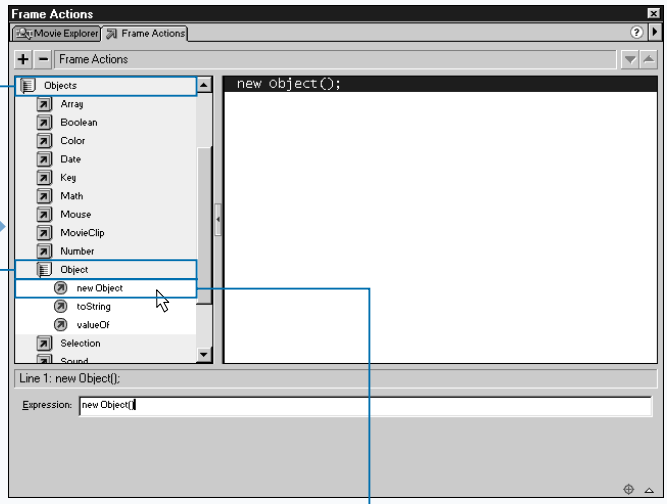
```
newColor = new Color(sampleMC);.
```

Color is the constructor. The variable `newColor` holds the object. The instance `sampleMC` is the movie clip you want to target. Creating an object with the new operator instantiates the object or creates an instance of the object.

An instantiated object has all of the properties and methods of the object from which it was constructed. Some of the objects predefined by Flash require you to instantiate the object before you can access the methods and properties of the object.

INTRODUCTION TO OBJECTS





1 Select the frame, button, or move clip to which you want to add ActionScript.

Note: This example uses file object.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Click Objects to open the Objects category.

4 Click Object to open the Object category.

5 Double-click new Object.

■ The new Object constructor appears in the Action list.

Extra

You can create generic objects. In Normal mode, you can create a generic object using the `new Object` in the Object category. The following syntax creates a generic object: `genericObject = new Object();`. The variable `genericObject` holds the object.

Objects can have properties. You can access the properties and you can assign values to the properties. To access an object property, precede the property name with the `objectName` followed by a dot. For example, if `genericObject` has a `weight` property, you can use the syntax that follows to access the `weight` property and assign it to a variable:

Example:

```
propertyValue = genericObject.weight;
```

In the example, `propertyValue` is a variable.

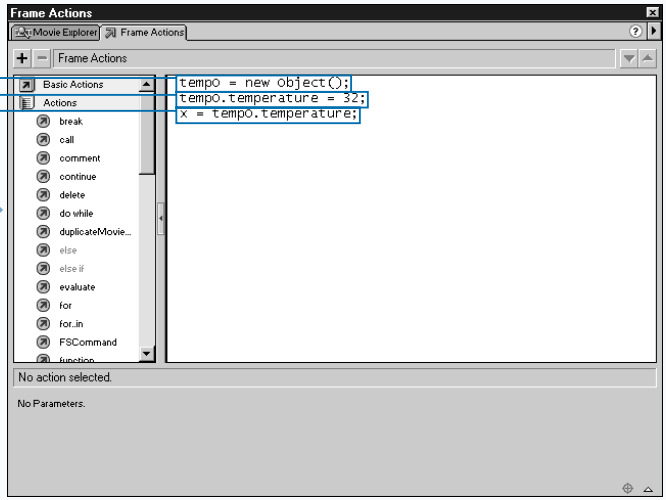
To assign a value to a property, use the syntax that follows:

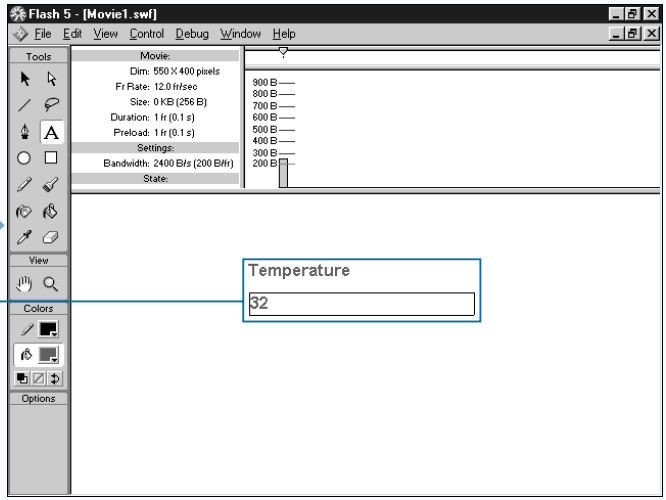
Example:

```
genericObject.weight = 32
```

The example assigns 32 to the `genericObject` weight property.

The `MovieClip` object is the most important object in ActionScript. Because each movie clip has its own Timeline, each movie clip can be a complete animation. You can give each instance of a movie clip a unique name. This enables you to target or perform on each movie clip instance.





6 Assign the new object to a variable.

7 Assign a property to the object.

8 Assign a property to a variable.

9 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

10 Test your movie.

■ You have created a generic object with a temperature property. The temperature property automatically displays on the screen when you enter the test environment.

USING THE MOVIECLIP OBJECT

The methods of the `MovieClip` object give you the ability to perform an action on a movie clip instance or to retrieve a value associated with a movie clip. By giving a unique name to each movie clip instance, you can target instances of a movie clip. You do not need a constructor to call `MovieClip` object methods; instead, precede each method with an instance name followed by a dot, for example:

```
movieClipName.method(arguments);.
```

The `MovieClip` object methods in the table below give you the same functionality as standard actions.

MOVIECLIP OBJECT METHOD	FUNCTION
`DuplicateMovieClips`	Creates a new instance of the specified movie clip. The duplicated movie clip begin playing in frame 1. If you delete the parent movie clip, Flash will delete the duplicate movie clip. Flash does not copy variables in the parent movie clip to the duplicated movie clip.
`getURL`	Opens a Web page in a browser. You can also use the `getURL` method to pass variables by using either the `get` or `post` method.
`gotoAndPlay`	Moves to a specified frame within a specified scene and begins the movie clip.
`gotoAndStop`	Moves to a specified frame within a specified scene and stops the movie clip.
`loadMovie`	Loads the specified movie into the movie clip.
`loadVariables`	Loads variables into a movie clip from an external file.
`nextFrame`	Sends the playhead to the next frame in the specified movie clip and stops the movie clip.
`play`	Starts the specified movie clip.
`prevFrame`	Sends the playhead to the previous frame in the specified movie clip and stops the movie clip.
`removeMovieClip`	Removes a movie clip created by the `duplicateMovieClip` action, `duplicateMovieClip` method, or `attachMovie` method.
`startDrag`	Makes the specified movie clip draggable.
`stop`	Stops a specified movie clip.
`stopDrag`	Stops the dragging of the currently draggable movie clip.
`unloadMovie`	Unloads a movie loaded with the `loadMovie` method.

The table that follows lists an example of each of the movie clip methods that provide the same functionality as standard actions.

ACTION	EXAMPLE
MovieClip.duplicateMovieClip	sampleMC.duplicateMovieClip(newSampleMC,1);
MovieClip.get.getURL	sampleMC.getURL ("http://www.baycongroup.com", "_blank", "GET");
MovieClip.gotoAndPlay	sampleMC.gotoAndPlay (1);
MovieClip.gotoAndStop	sampleMC.gotoAndStop (1);
MovieClip.loadMovie	sampleMC.loadMovieNum ("load.swf", 1);
MovieClip.loadVariable	sampleMC.loadVariablesNum ("sales.txt", 0);
MovieClip.nextFrame	sampleMC.nextFrame();
MovieClip.play	sampleMC.play();
MovieClip.prevFrame	sampleMC.prevFrame();
MovieClip.removeMovieClip	sampleMC.removeMovieClip();
MovieClip.startDrag	sampleMC.startDrag();
MovieClip.stopDrag	sampleMC.stopDrag();
MovieCip.unloadMovie	sampleMC.unloadMovie();

The movie clip object includes several methods that enable you to perform actions that are not standard actions on movie clips. You can apply many objects to a movie by not specifying an instance name. The table that follows lists these actions:

MOVIE CLIP OBJECTS	
Action	Function
attachMovie	Adds an instance of a symbol located in the Library to the Stage.
getBounds	Retrieves the boundaries of the specified movie clip.
getBytesLoaded	Retrieves the number of bytes loaded for the specified movie clip.
getBytesTotal	Retrieves the total number of bytes for the specified movie clip.
globalToLocal	Retrieves the coordinates of the specified Timeline.
hitTest	Detects when two objects overlap or intersect one another.
localToGlobal	Retrieves the coordinates of the main Timeline.
swapDepths	Enables you to specify the depth level on which a movie clip will appear.

ATTACH A MOVIE CLIP

You can use the `attachMovie` method to add an instance of a symbol located in the Library to a movie clip located on the Stage. Use this method when you want to place an instance of a symbol that is not already on the Stage. You may find this method useful when you want to start your movie with a blank Stage and add movie clips as the movie plays.

Before you can attach a movie, you must assign an identifier name and linkage type to the symbol. Use the Symbol Linkage dialog box to assign the identifier name and linkage type.

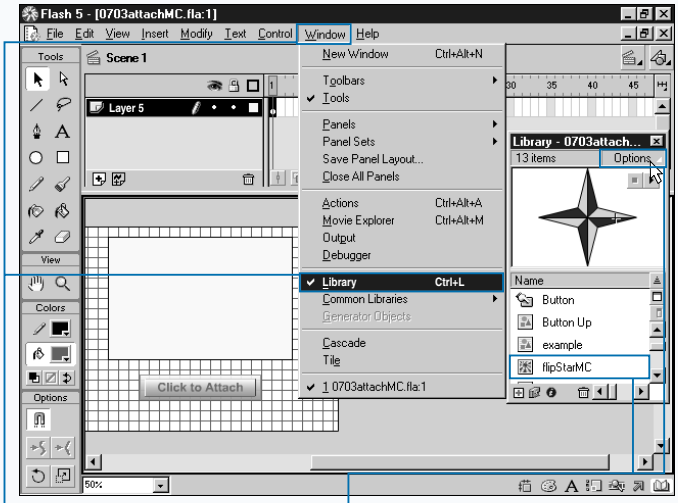
The syntax for the `MovieClip.attachMovie` method is

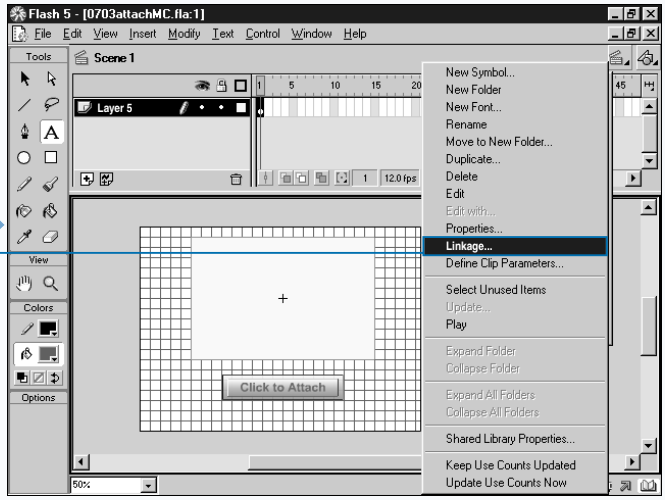
```
MovieClip.attachMovie("id","name",depth);.
```

Use the `MovieClip` argument to specify the movie clip you want to attach. Use the `id` argument to specify the name of the symbol in the Library to place on the Stage. The name you specify should correspond to the name given to the symbol in the Identifier field of the Symbol Linkage Properties dialog box. Also, you must set the symbol to Export in the Symbol Linkage Properties box. Use the `name` argument to specify the instance name you want to give the new instance. Use the `depth` argument to specify an integer that represents the level on which you want to place the new instance. Omit the `MovieClip` argument if you want to attach the movie.

You can use the `attachMovie` method to create multiple instances of the same movie clip and place them on the Stage. You can use the `_x` and/or `_y` properties to set the location of the movie clip.

ATTACH A MOVIE CLIP





1 Click Window ➪ Library to open the Library.

Note: This example uses file attachMC.fla, which you can find on the CD that accompanies this book.

2 Click the item in the Library you want to attach.

3 Click to select the Options menu.

4 Click Linkage on the Options menu.

Apply It

Attaching a movie clip enables you to place an instance of a movie clip located in the Library on the Stage as the movie plays.

TYPE THIS:

```
sampleMC.attachMovie( "addMC","addMC2", 2 );
```

RESULT:

An instance of addMC is taken from that library and added to the Stage.

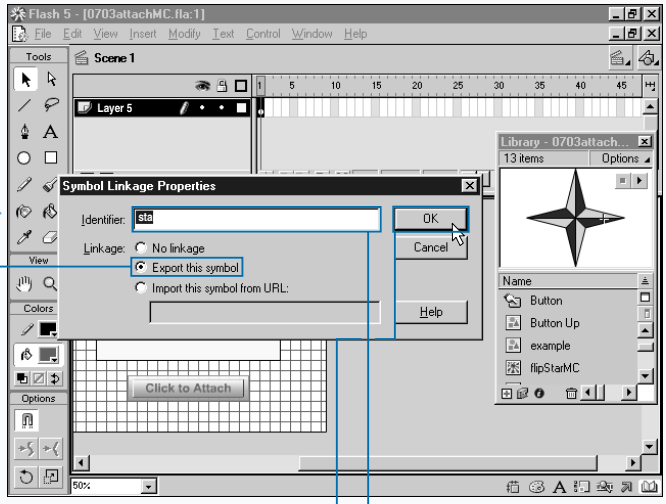
To remove a movie clip, you type the following:

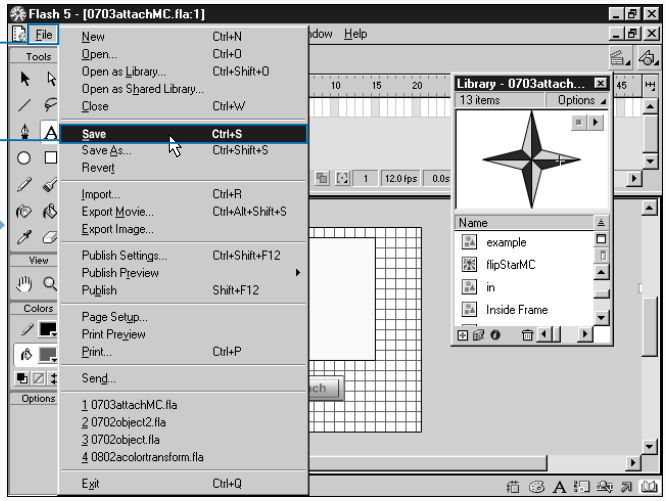
TYPE THIS:

```
removeMovieClip("addMC2");
```

RESULT:

The instance `addMC2` is removed from the Stage.





■ The Symbol Linkage Properties dialog box appears.

5 Select Export This Symbol as the Linkage type.

6 Type a name in the Identifier field.

7 Click OK.

8 Click File ➪ Save to save your file.

■ You must save your file after you set the Symbol Linkage property.

CONTINUED

ATTACH A MOVIE CLIP (CONTINUED)

If you attach several movie clips without setting the location, the movie clips stack on top of each other. You will not be able to discern the new instances from the old instances.

Each instance of an attached movie clip must have a unique name. You can append a unique number to the end of the movie clip name to make each instance unique. If you are making several instances of the same movie clip at once, use a loop to create the instances. You can have each increment of the loop append a unique number to the instance name and place the instance on the Stage at the location you specify.

You use the `removeMovieClip` action to remove instances created with the `attachMovieClip` action. The syntax for the `removeMovieClip` action is

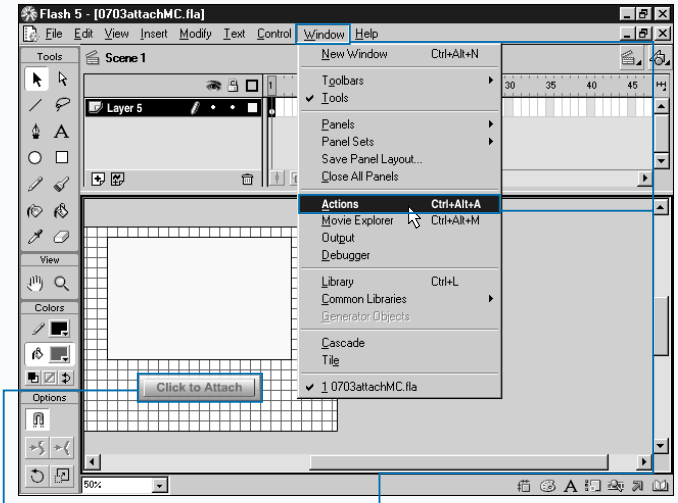
```
removeMoveClip(target);.
```

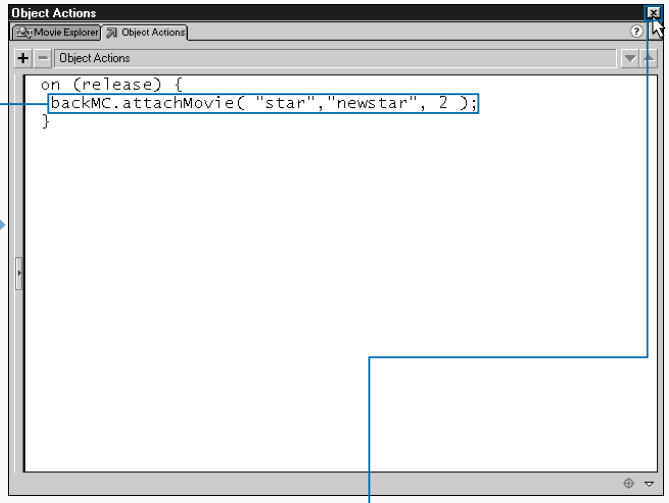
Use the `target` argument to specify the movie clip that you want to remove.

Shared libraries enable you to use assets from one Library in multiple Flash movies. However, you cannot use the `attachMovie` method to create an instance of a symbol imported from a shared Library because you must set attached symbols to Export in the Symbol Linkage Properties dialog box. Flash automatically sets imported symbols to Import in the Symbol Linkage Properties dialog box.

You can also control when an instance appears on the Stage by first placing the instance on the Stage with its `_visible` property set to `false`. When you want the instance to appear, set the `_visible` property to `true`. Use this method when working with symbols imported from shared libraries.

ATTACH A MOVIE CLIP (CONTINUED)





9 Select the frame, button, or movie clip to which you want to add ActionScript.

10 Click Window ➪ Actions to open the Actions panel.

11 Attach the movie.

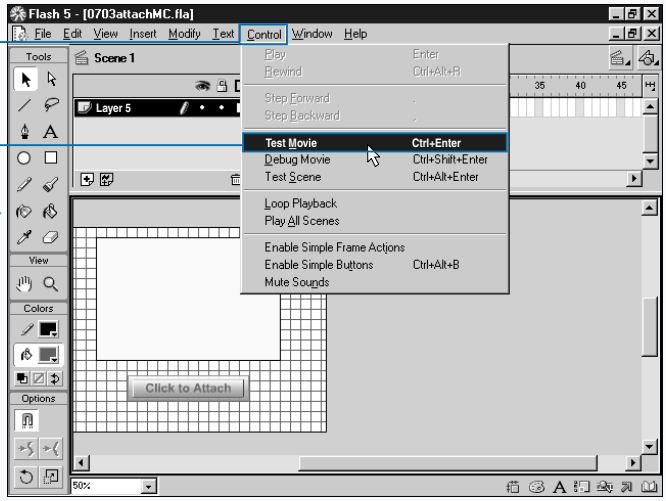
12 Click ☒ to close the Actions panel.

Extra

You can use `attachMovie` to place several instances of a movie clip on the Stage. The script that follows places three instances of a movie clip on the Stage and sets their location. The script is associated with a Frame.

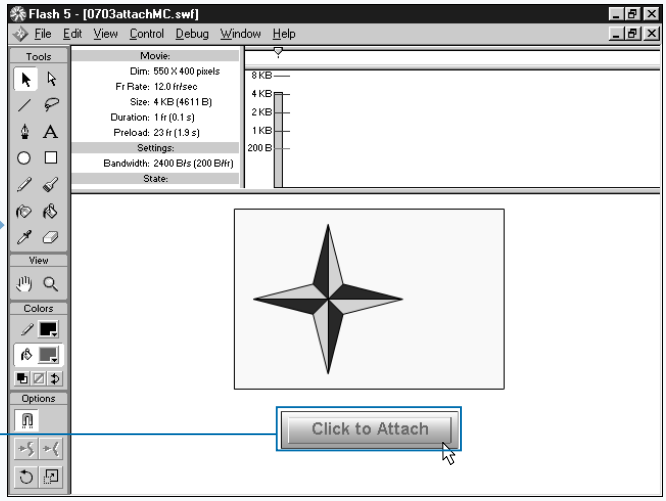
Example:

```
xlocation = 250;
ylocation = 150;
max  = 4;
attachMovie("star","star0", 1);
setProperty ("star0", _x, xlocation);
setProperty ("star0", _y, ylocation);
for  (i=2; i<max; i++) {
     attachMovie("star","star" +  i, i);
     nameMC = "star"+i;
     setProperty (nameMC, _x, xlocation + 100);
     setProperty (nameMC, _y, ylocation + 50);
     xlocation = xlocation + 100;
     ylocation = ylocation + 50;
}
```



13 Click Control ➪ Test Movie to move to the test environment.

Note: See page 38 for instructions on how to test your movie.



14 Click the button to test your movie.

■ ActionScript attaches the movie clip when you click the button.

GET BOUNDS

Let us say you have placed a movie clip on the Stage or you have added a movie clip to the Stage using the `duplicateMovieClip` action or the `attachMovieClip` method. You now want to place another movie clip to the left, right, above, or below that movie clip. You can use the `getbounds` method to obtain the coordinates of the boundary of a movie clip that is on the Stage. You can then position another movie clip on the Stage in a position relative to the original movie clip. The `getbounds` method retrieves the boundaries of a movie clip and it enables you to position objects on the Stage.

You may also want to know the boundaries of a movie clip so that you can trigger an action when the mouse comes within a certain distance of a movie clip. For example, you could make a pumpkin glow if the mouse comes within 10 pixels of the pumpkin.

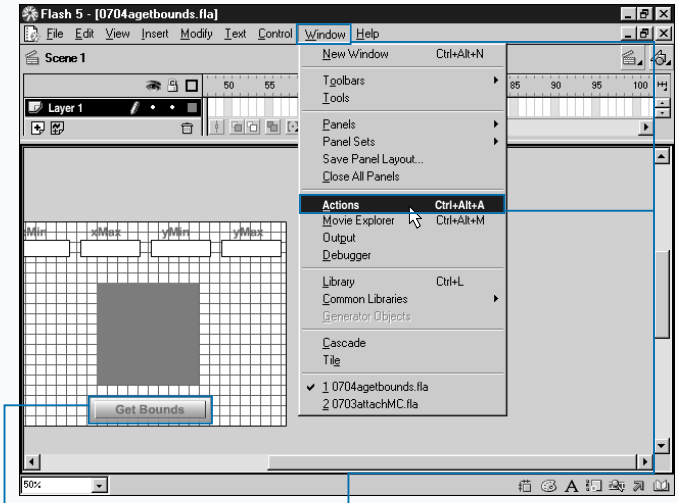
The syntax for the `MovieClip.getbounds` method is

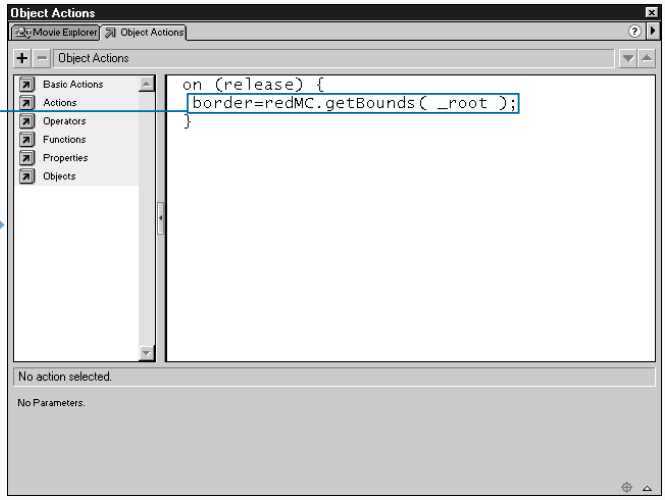
`MovieClip.getbounds(targetCoordinateSpace);`.

Use the `MovieClip` argument to specify the movie clip whose coordinates you want to obtain. Use the `targetCoordinateSpace` argument to specify the target path of the timeline whose coordinates you want to use as a point of reference. The `getbounds` method returns an object with `xMin`, `xMax`, `yMin`, and `yMax` properties. The `xMin` property represents the left edge of the movie clip, the `xMax` property represents the right edge, the `yMin` property represents the top edge, and the `yMax` property represents the bottom edge.

To obtain the `xMin`, `xMax`, `yMin`, and `yMax` properties, assign the results of the `getbounds` method to a variable. Then precede the property with the variable name followed by a dot.

GET BOUNDS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file getbounds.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

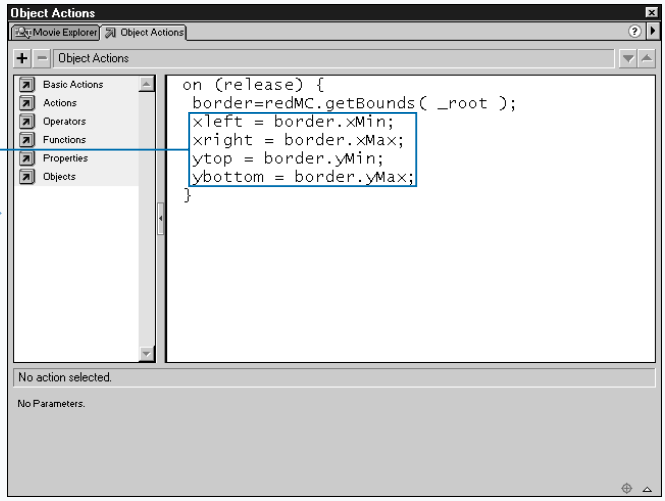
3 Retrieve the bounds of a movie clip.

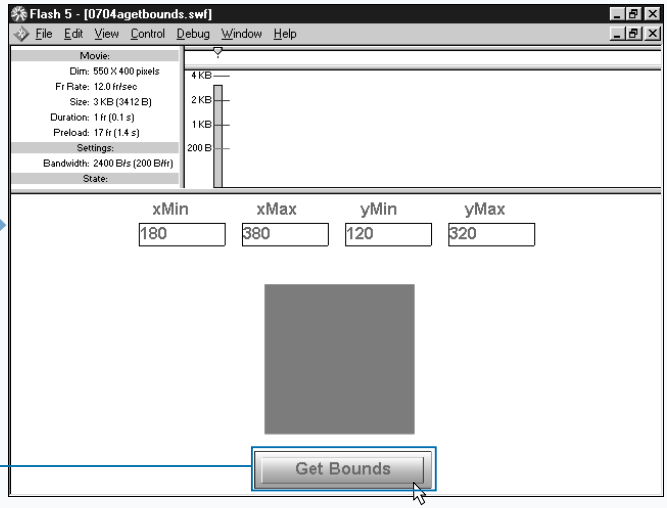
Extra

You can use `getbounds` to help you place an attached movie clip on the Stage. The example that follows attaches the star movie clip and places two instances of the star movie clip on the Stage side by side.

Example:

```
on (release) {
     backMC.attachMovie( "star","star1", 2 );
     border=_root.backMC.star1.getBounds(_root.backMC.star1);
     xright = border.yMax;
     backMC.attachMovie( "star","star2", 3);
     setProperty (_root.backMC.star2, _x, xright);
}
```





4 Assign each property to a variable.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button to test your movie.

■ When you click the button the `xMin`, `xMax`, `yMin`, and `yMax` properties display on the screen.

SWAP DEPTHS

When you place objects on the Stage, Flash gives them a depth level and stacks them. Movie clips with a higher depth level appear to be in front of movie clips with a lower depth level. For example, if you place two thumbnail photographs on the Stage and the user clicks on the photograph, the image changes to a full screen view. The problem is that the image with the lower depth level will appear to be behind the image with the higher depth level when you increase its size. A similar problem can occur when you create a number of draggable objects. Each object is on its own level. You may want the object the user is dragging to always be on top or always be on bottom. You can use the `swapDepths` method to change the stacking order of the movie clips so that the images always appear on the level that you desire.

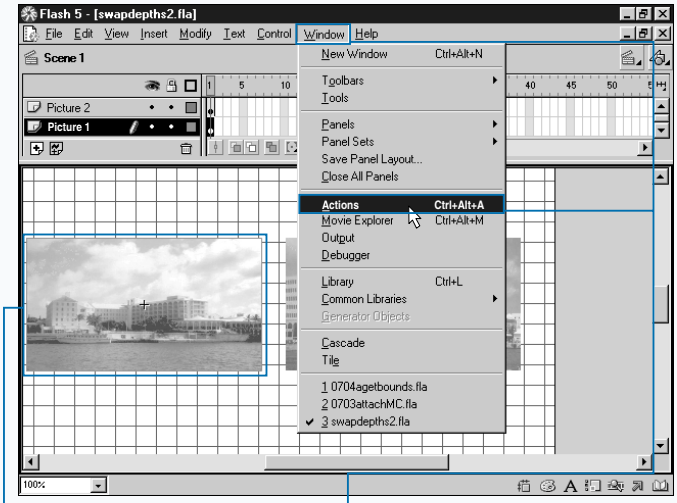
The syntax for `MovieClip.swapDepths` is

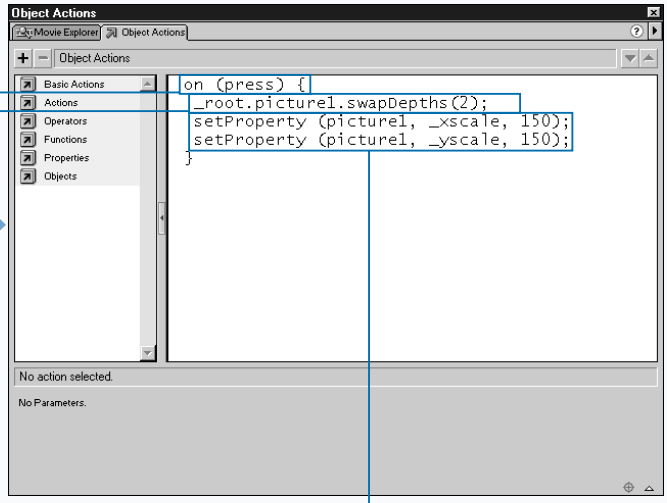
`movieClipName.swapDepths(depth/target) .`

Use the `movieClipName` argument to specify the name of the movie clip for which you what to swap the depth. Use the `target` argument to specify the movie clip with which you want to swap the depth. In other words, if you want to swap the depth of movie clip A with movie clip B, use the syntax that follows: `A.swaptDepths(B);`. Alternatively, you use the depth argument to specify the depth number instead of using the `target` argument.

When using the `swapDepths` method, both movie clips must have the same parent. If a movie clip is tweening when you call the `swapDepths` method, the tweening stops.

SWAP DEPTHS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file swap.fla, which you can find on the CD that accompanies this book.

Note: The script is associated with the button that has been converted to a movie clip. To see the script in the example file, double-click the movie clip and open the Actions panel.

2 Click Window ➪ Actions to open the Actions panel.

3 Set the handler to `on (press)`.

4 Swap the depth of the movie clip.

5 Increase the size of the movie clip.

Apply It

You can use `swapDepths` to swap the depth of movie clip A with movie clip B. The script that follows is associated with a button and swaps the dept of pictureA with pictureB.

TYPE THIS:

```
on (release) {
    pictureA.swapDepths(pictureB);
}
```

RESULT:

Each time the user clicks the button ActionScript swaps the depth of pictureA with pictureB.

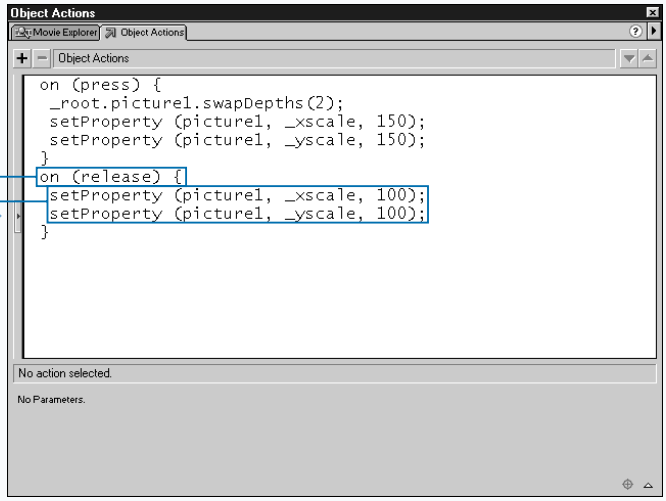
Alternatively, you can use a depth number to change the depth of a movie clip. The script that follows is associated with a button.

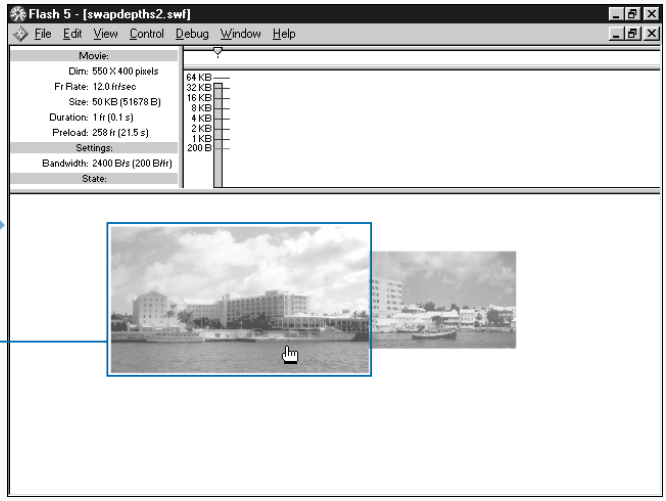
TYPE THIS:

```
on (release) {
    pictureA.swapDepths(100);
}
```

RESULT:

Each time the user releases the mouse after clicking the button, pictureA moves to depth 100.





6 Set the handler to `on (release)`.

7 Reduce the size of the movie clip to its original size.

■ If you want to adjust the depth of several symbols, repeat steps 1 through 7 for each symbol.

8 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

9 Click an image to test your movie.

■ When you click the image, the depth level swaps to the top level and the image enlarges.

CHECK MOVIECLIP LOAD

You have created movie clips A and B. Movie clip B is a large movie clip. You want to load movie clip B into movie clip A. You do not want movie clip B to begin playing until it is completely loaded. Use the `getBytesTotal` method to determine the size of movie clip B in bytes. Then use the `getBytesLoaded` method to obtain the number of bytes loaded for movie clip B. When `getBytesTotal` equals `getBytesLoaded`, start movie clip B.

The syntax for the `MovieClip.getBytesLoaded` method is

```
MovieClip.getBytesLoaded(); .
```

The syntax for the `getBytesTotal` method is

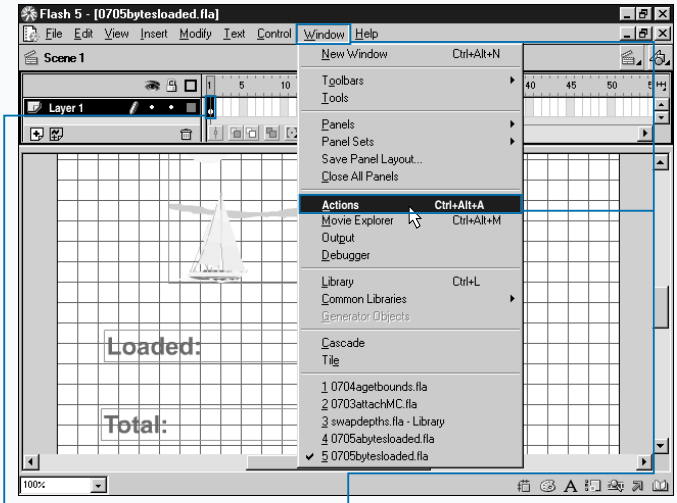
```
MovieClip.getBytesTotal() .
```

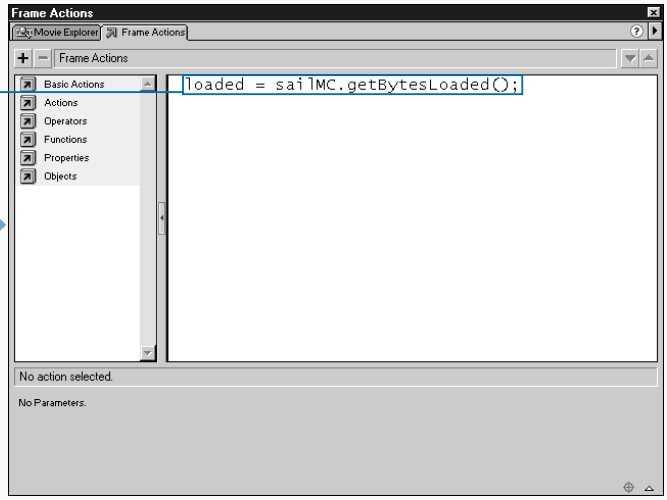
When using `MovieClip.getBytesLoaded` or `MovieClip.getBytesTotal`, use the `MovieClip` argument to specify the movie clip for which you want to obtain the bytes loaded or the total number of bytes. Together, `getBytesTotal` and `getBytesLoaded` enable you to monitor the progress of the loading of a movie clip. Note that it is not always necessary to wait for a movie clip to load completely before it starts. You can specify the exact number of bytes that need to load before starting a movie clip.

Use the `getBytesLoaded` and `getBytesTotal` methods with movie clips that have been loaded — not with movie clips that are internal to the movie. If you use these methods with movie clips that are internal to the movie, the value for `getBytesLoaded` and `getBytesTotal` will be identical, because internal movies clips load automatically.

Preloaders are movies that run while ActionScript waits for a movie or movie clip to load. You can create a preloader movie clip that runs until a movie clip loads.

CHECK MOVIECLIP LOAD





1 Select the frame, button, or movie clip to which you want to add ActionScript.

■ This example uses a frame.

2 Click Window ➪ Actions to open the Actions panel.

3 Get the bytes loaded.

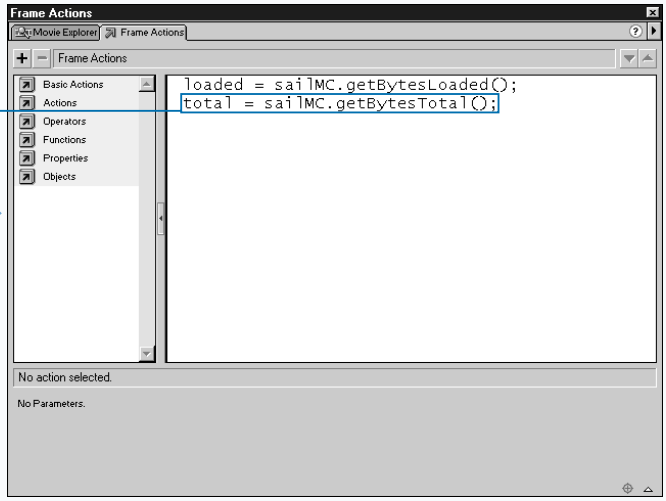
■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

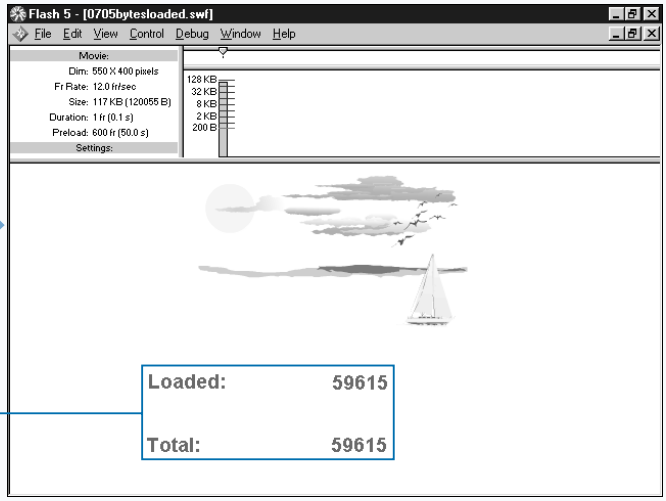
Extra

Use the `getBytesLoaded` method with `getBytesTotal` to determine the percent of a movie clip that has been loaded. This is useful if you want to run a preloader while another movie clip loads.

Example Frame 1

```
loaded = sailMC.getBytesLoaded();
total = sailMC.getBytesTotal();
percent = (loaded/total) * 100;
```





4 Get the total bytes.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Test your movie.

■ When the movie plays, the bytes loaded and the total bytes automatically display on the screen.

DETECT COLLISION

You use the `hitTest` method to detect when two objects are in collision. When one object overlaps or intersects another object, ActionScript considers the objects to be in collision. When programming, use `hitTest` to determine if one object touches or overlaps with another object. The syntax for the `MovieClip.hitTest` method is

```
MovieClip.hitTest(target);
MovieClip.hitTest(x,y,shapeFlag)
```

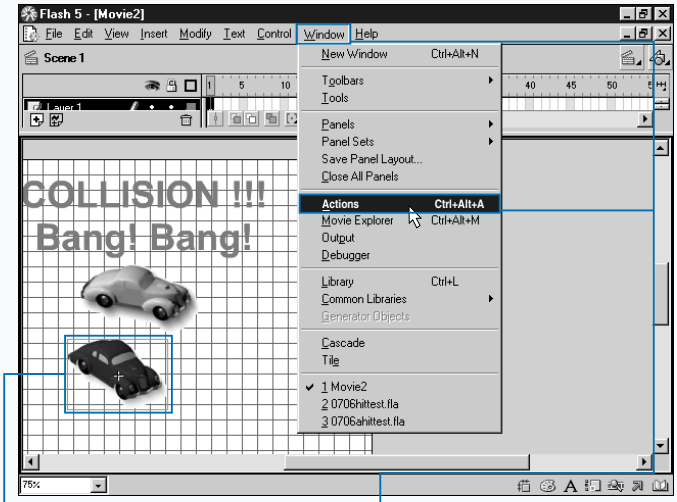
The `hitTest` method has two uses. You use the syntax `MoveiClip.hitTest(target);` to determine when the object specified in the `target` argument has collided with the instance specified in the `MovieClip` argument. You use the `target` argument to specify the path of the object that might collide with the movie clip in the `MovieClip` argument.

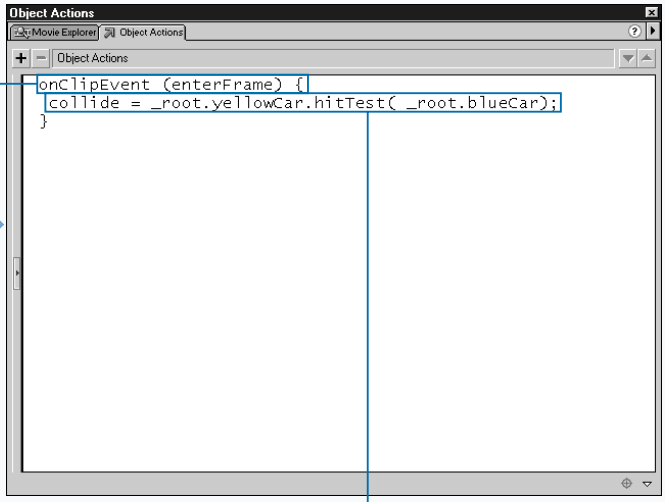
You use the syntax `MovieClip.hitTest(x,y,shapeFlag)` to determine when the instance specified in the `MovieClip` argument has collided with the specified x- and y-coordinates. Use the `MovieClip` argument to specify the target path of the instance you are testing.

Set the `shapeFlag` to the Boolean value `true` if you want ActionScript to use the object specified in the `MovieClip` argument to determine whether a collision has occurred. Set the `shapeFlag` to the Boolean value `false` if you want ActionScript to use the bounding box of the object specified in the `MoveClip` argument to determine whether a collision has occurred. The bounding box is the rectangular area that surrounds the movie clip when you select it. ActionScript returns the Boolean value `true` when the objects are in collision and the Boolean value `false` when the objects are not in collision.

You can use the `_xmouse` and `_ymouse` properties as the x- and y-coordinates to determine whether the pointer is over a movie clip. The `_xmouse` and `_ymouse` properties return the x- and y-coordinates of the pointer.

DETECT COLLISION





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses hittest.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Set the handler to `enterFrame`.

4 Determine if the movie clips are in collision.

Extra

You can use `hitTest` to determine if two objects overlap. The script in this example returns the Boolean value `true` to the variable `collision` when the user drags a blue dot and causes it to collide with a red box. The example assigns the variable `collision` to a dynamic text box. The `blueDot` is a draggable movie clip. The `redBox` is a movie clip. You assign the action to `redBox`.

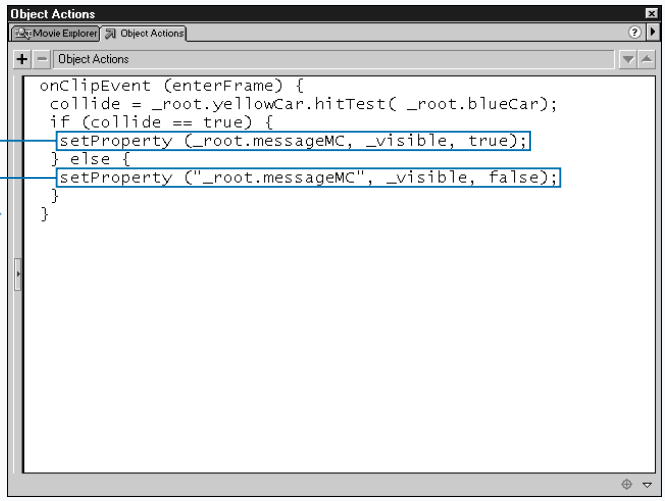
Example

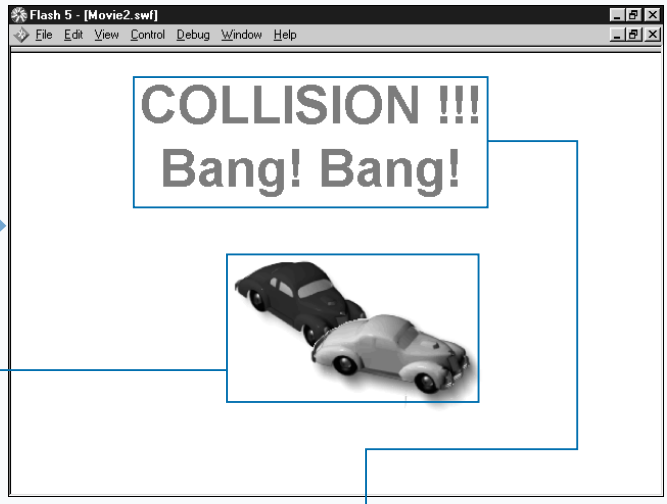
```
onClipEvent (enterFrame) {
  _root.collision =_root.redBox.hitTest( _root.blueDot);
}
```

You can use `hitTest` to determine if two objects are in collision. The script in this example returns the Boolean value `true` to the variable `collision` when the user drags a blue dot and causes it to collide with the x- and y-coordinates of a red dot. The example assigns the variable `collision` to a dynamic text box. The `blueDot` is a draggable movie clip. The `redDot` is a movie clip that motion-tweens across the Stage. The variables `a` and `b` are assigned to dynamic text boxes. They capture and display the x- and y-coordinates of `redDot`. You assign the action to `blueDot`.

Example

```
onClipEvent (enterFrame) {
  _root.a = getProperty ( _root.redDot, _x );
  _root.b = getProperty ( _root.redDot, _y );
  _root.collision = _root.blueDot.hitTest(_root.a, _root.b, false );
}
```





5 Set the messageMC visible property to true if the movie clips are in collision.

6 Set the messageMC visible property to false if the movie clips are not in collision.

7 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

8 Test your movie by dragging a car so that it touches the other car.

■ When one car touches the other car, a message appears on the screen.

GET THE X- AND Y-COORDINATES

You can use the x- and y-coordinates to assist you when placing or locating objects on the Stage. Every timeline has x- and y-coordinates. The x-coordinate marks the distance from the left side of the movie or movie clip to the right side of the movie or movie clip. The y-coordinate marks the distance from the top of the movie or movie clip to the bottom of the movie or movie clip. On the main timeline, the zero x-coordinate is on the left side of the movie. The x-coordinate increases as you move to the right. The zero y-coordinate is at the top of the movie. The y-coordinate increases as you move downward.

The zero points of both the x- and y-coordinates of a movie clip are located in the center of the movie clip. As you move to the left of the center, the x-coordinate values decrease as negative values. As you move to the right of the center, the x-coordinate values increase as positive values. As you move up from the center, the y-coordinate values decrease as negative values. As you move down from the center, the y-coordinate values increase as positive values.

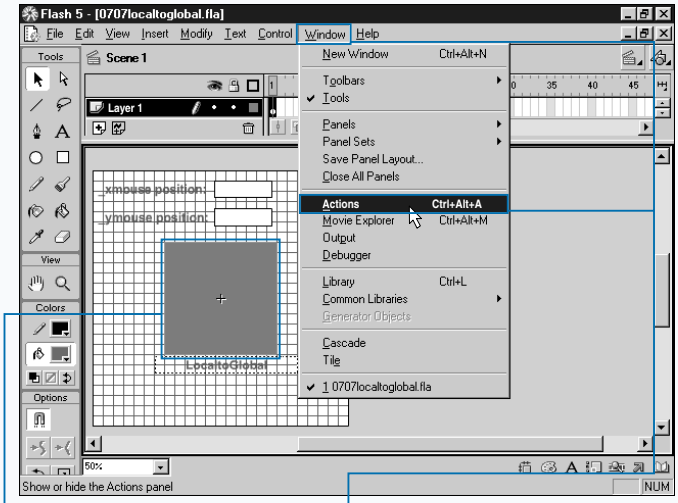
When retrieving coordinates, use the `globalToLocal` method to retrieve the coordinates of the movie clip. Use the `localToGlobal` method to retrieve the Stage coordinates.

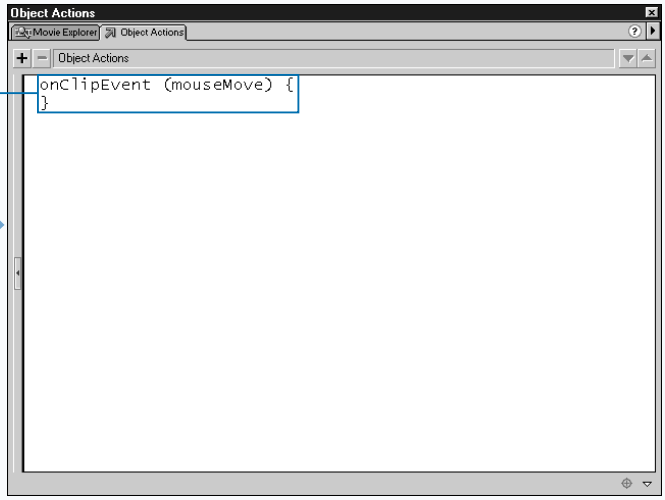
The syntax for the `MovieClip.globalToLocal` and `MovieClip.localToGlobal` methods is

```
movieClipName.globalToLocal(point),.
movieClipName.localToGlobal(point); .
```

Use the `new object()` constructor to create an object. Use the `movieClipName` argument to specify the movie whose coordinates you want to use. Use the `point` argument to identify the object you created with `the new object()` constructor. The `globalToLocal` and `localToGlobal` methods create an object with x-coordinate and y-coordinate properties. You can retrieve and use the properties in your script.

GET THE X- AND Y-COORDINATES





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file localtoglobal.fla, which you can find on the CD that accompanies this book.

■ This example uses a movie clip.

2 Click Window ➪ Actions to open the Actions panel.

3 Set the event handler to `mouseMove`.

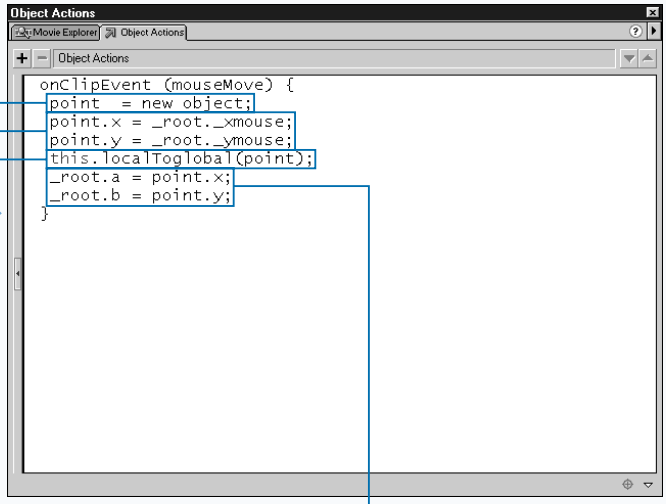
■ This causes the script to execute every time the user moves the mouse.

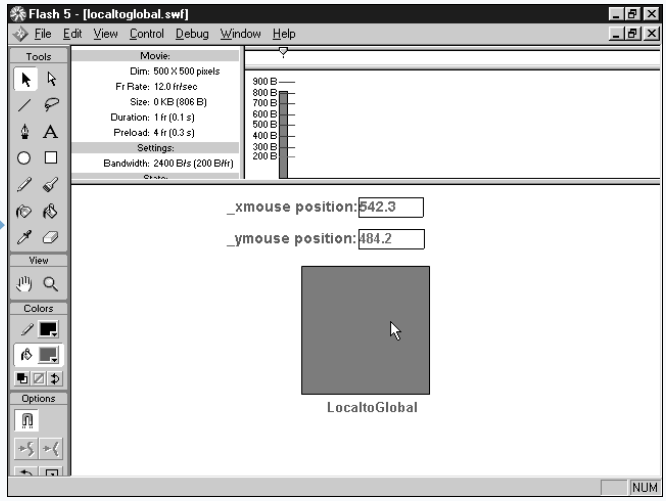
Extra

You can retrieve the `_xmouse` and `_ymouse` coordinates. In the script that follows, each time the user moves the mouse, this script displays the local coordinates of the mouse. The variables `_root.a` and `_root.b` are dynamic text boxes. The script creates a new object named position and then assigns the `_xmouse` and `_ymouse` properties to the `x-` and `y-` properties of the position object. The script converts the `x-` and `y-`properties to local coordinates and displays the coordinates on the screen. The Keyword this refers to is the current Timeline.

Example

```
onClipEvent (mouseMove) {
  position = new object();
  position.x = _root._xmouse;
  position.y = _root._ymouse;
  this.globalToLocal(position);
  _root.a = position.x;
  _root.b = position.y;
}
```





4 Create a new object.

5 Assign the `_xmouse` and `_ymouse` properties to the `x-` and `y-` properties of the new object.

6 Use the current timeline as the object whose coordinates you retrieve.

7 Assign the coordinates to a variable.

■ If you associate the variables with dynamic text boxes, you can display the values the variables return to the user.

8 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

9 Test your movie by dragging you mouse around the screen.

■ The x and y coordinates of the mouse display on-screen. As you drag the mouse, the coordinates change.

CREATE AN ARRAY

An array is a list of values separated by commas. Arrays enable you to group values together and are useful when you need to store and retrieve lists of data. An array can contain strings, numbers, or Boolean values. When working with an array, enclose string values in quotes.

The following example assigns the name George, the number 23, and the Boolean value `false` as an array to the variable `nameAge`: `nameAge = ["George",23, false];`.

Each value in an array is an element. ActionScript assigns each element a unique consecutive number called an index. The first item in an array has an index of [0], the second an index of [1], and so forth. In the example, George has an index of [0], 23 has an index of [1], and false has an index of [2].

You can use the constructor `new Array` to create an array. The syntax for the constructor `new Array` is

```
arrayName = new Array(e0,e1,...eN) .
```

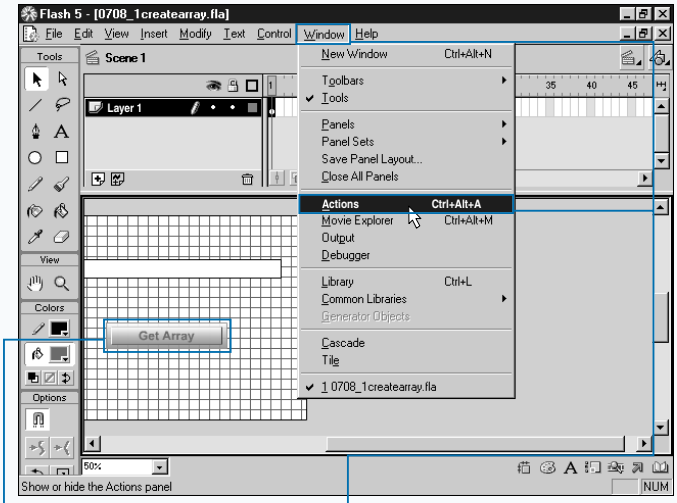
Use the `arrayName` argument to specify the variable that names the array. Use arguments `e0` to `eN` to specify the elements in the array.

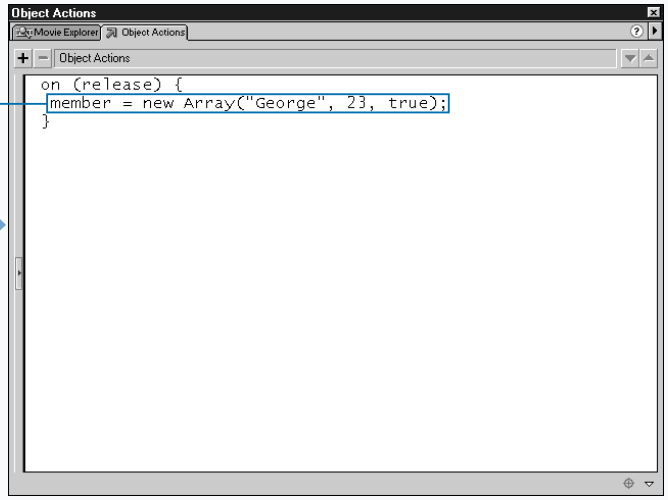
You can also use the array access operator `[ ]` to create an array. The syntax for creating an array with the array access operator is

```
arrayName = [a0, a1,... aN]; .
```

Use the `arrayName` argument to specify the variable that names the array. Use arguments `a0` to `aN` to specify the elements in the array.

CREATE AN ARRAY





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file createarray.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create an array.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

Apply It

You can create an array using the access operator. The script that follows is an example. It includes strings, numbers, and Boolean values.

TYPE THIS:

```
nameAge = ["George",23,false,"Jane",45,true];
```

RESULT:

The variable nameAge returns: George,23,false,Jane,45,true.

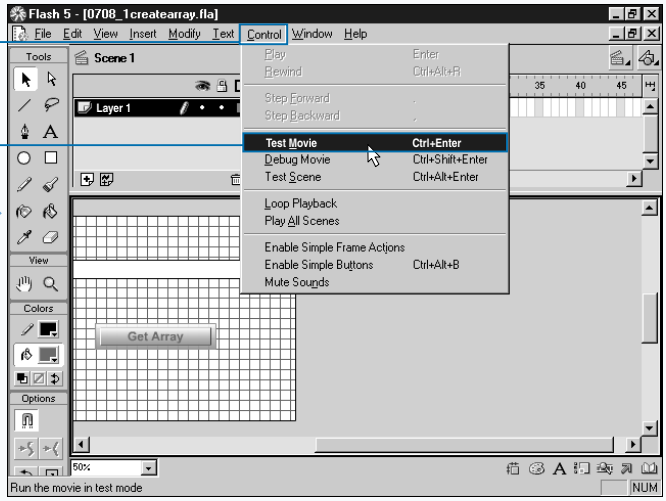
You can add elements to the array.

TYPE THIS:

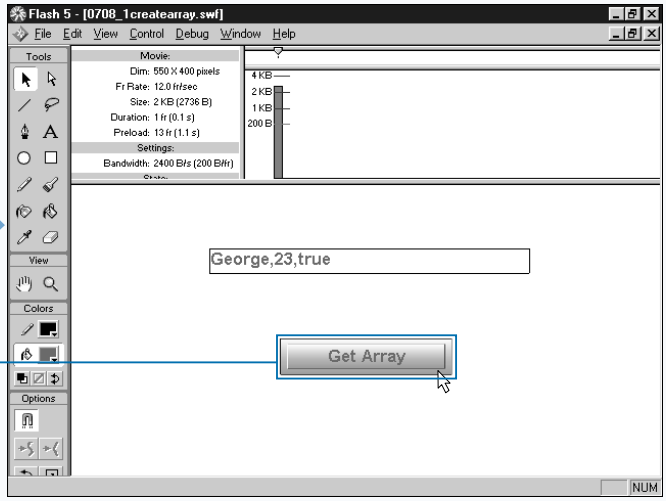
```
nameAge[6] = "Jen";
Name[7] = 13;
nameAge[8] = false;
```

RESULT:

The variable nameAge returns: George,23,false,Jane,45,true,Jen,13,false



4 Click Control ➪ Test Movie to move to the test environment.



5 Click the button to test your movie.

■ Clicking the button retrieves the array.

CONTINUED

CREATE AN ARRAY (CONTINUED)

After you create an array, you can add elements to it. In fact, you can create an empty array and add the elements later. The syntax for creating an empty array using the new array constructor is `arrayName = new Array ();`. The syntax for creating an empty array using the array access operator is `arrayName = [ ];`.

To access or assign elements to an array, refer to the element by using the syntax `variableName[N];`. Use `variableName` to specify the variable to which you assigned the array. Use the `N` argument to specify the index position of the element you want to access or assign. To add values to the array, specify the array name, followed by the index value enclosed in the array access operator, and assign a value.

This example creates a new array: `trees = new Array();`.

This example assigns pine, maple, and birch to elements [0], [1], and [2] in the array trees:

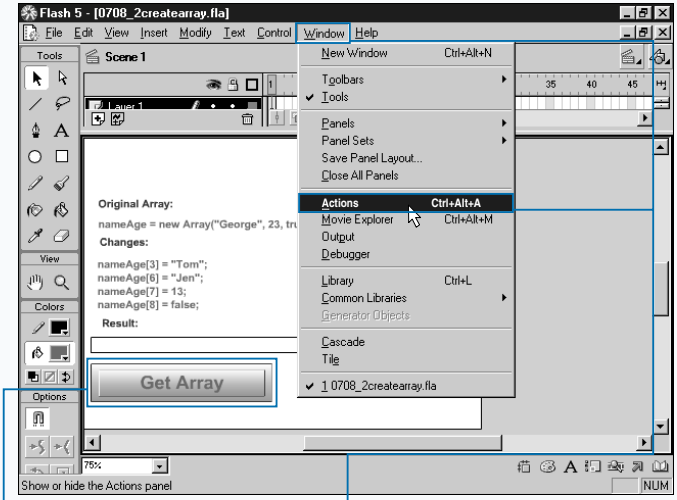
```
trees[0] = "pine";
trees[1] = "maple";
trees[2] = "birch";
```

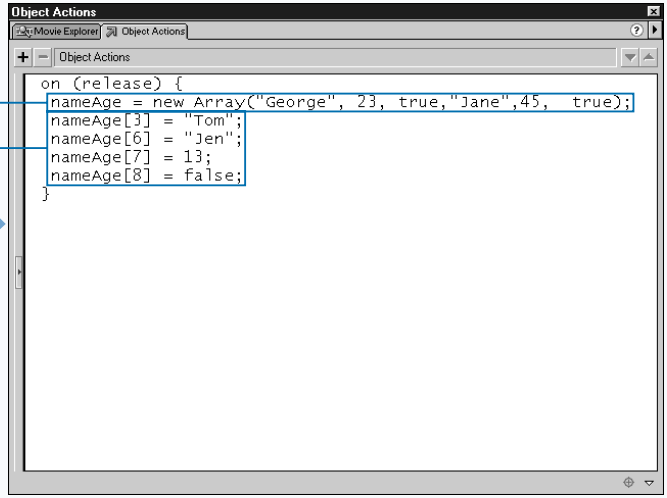
You can change the value of an element in an array by assigning it a new value. This example assigns the value birch to the element `trees[1]`: `trees[1] = "birch";`

You can access elements in an array. This example accesses element 1 of the trees array and assigns it to the variable wood. In other words, it assigns maple to the variable wood:

```
trees = ["pine","maple","birch"];
wood = trees[1];
```

CREATE AN ARRAY (CONTINUED)





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file createarray1.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Use the `new Array` method to create your array.

4 Use the `set variables` action to assign values to your array.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

Apply It

You can change the values of the element in an array. The array nameAge contains George,23,false,Jane,45,true,Jen,13,false.

TYPE THIS:

```
nameAge[3] = "Tom";
nameAge[4] = 56;
NameAge[5] = false;
```

RESULT:

The Array nameAge returns:
George,23,false,Tom,56,false,Jen,13,false

You can create an empty array:

```
colors = new Array ();
```

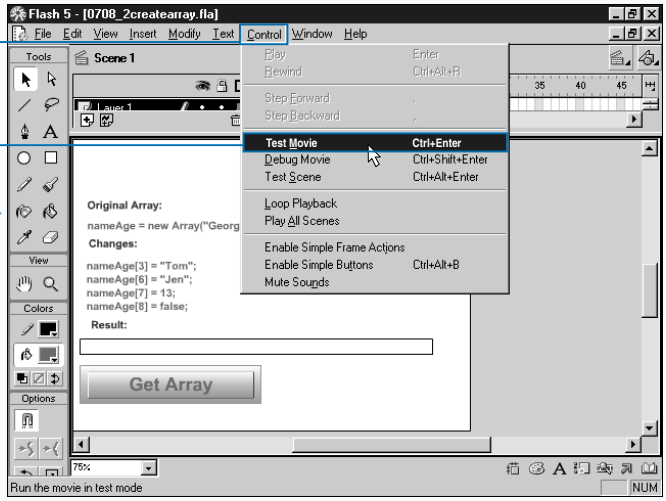
You can assign values to the empty array.

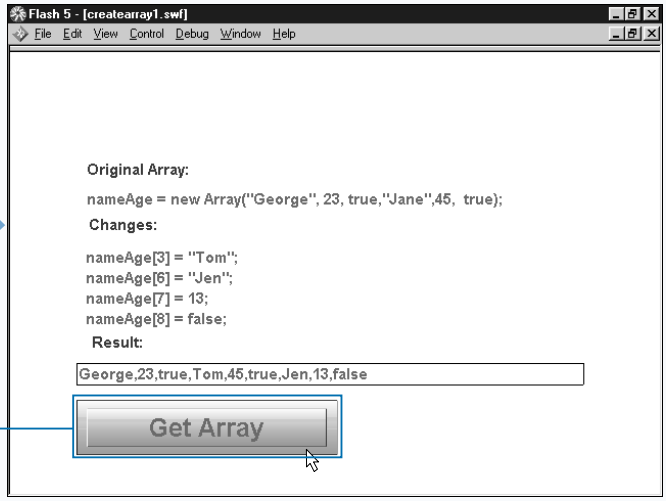
TYPE THIS:

```
colors[0] = "pink";
colors[1] = "yellow"
colors[2] = "blue"
```

RESULT:

pink,yellow,blue





5 Click Control ➪ Test Movie to move to the test environment.

6 Click the button to test your movie.

■ Clicking the button retrieves the current values assigned to the array.

FIND THE LENGTH OF AN ARRAY

Knowing the length of an array can be useful, particularly if you are looping through the data. Knowing the length helps you determine how many times you need to loop. The length of an array is equal to the highest index value plus 1. If the highest index value in an array is nine, the array has a length of ten. The `Array.length` property returns the length of an array.

The syntax for the `Array.length` property is

```
arrayName. length; .
```

Use the `arrayName` argument to specify the name of the array whose length you want to obtain.

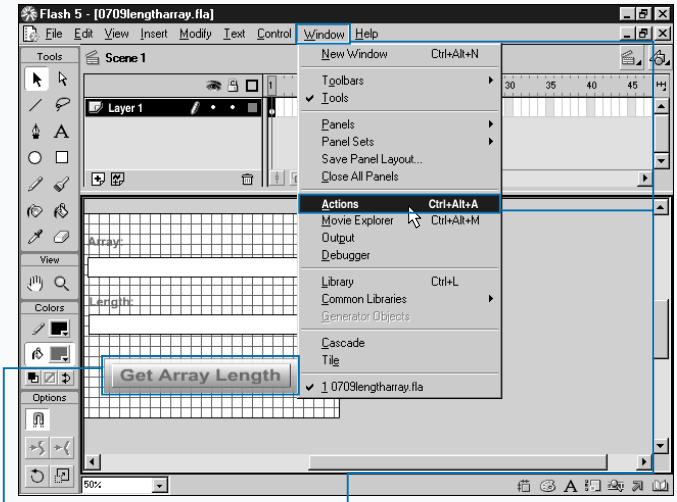
You can create an empty array of a specified length by using the constructor `new Array`. The syntax for creating an empty array of a specified length is

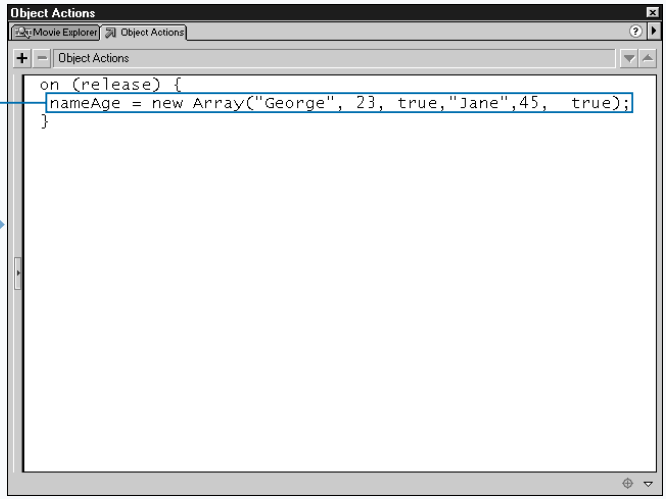
```
arrayName = new Array(length); .
```

Use the `arrayName` argument to specify the name of the array. Use the `length` argument to specify an integer that represents the length of the array. For example, in `trees = new Array(6)`, note that even though you have not assigned any elements to the array, the array has a length of 6. The array has a length of 6 because ActionScript reserves the space for the elements.

Flash updates the length property automatically as you add elements to an array. For example, if you use the `new Array` constructor to create an empty array, the array has a length of 0. If you add index position [1], the array has a length of 1. If you add index position [2], the array has a length of 2. If you then add index position [10], the array has a length of 10. Positions [3] through [9] do not have values assigned to them; however, ActionScript reserves space for these elements.

FIND THE LENGTH OF AN ARRAY





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file length.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create an array.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

Apply It

If you are not ready to add elements to an array, you can create an empty array.

TYPE THIS:

```
nameAge = new Array();
```

RESULT:

The length property will return : 0

If you create an empty array with a length of 3, ActionScript reserves space for the elements

TYPE THIS:

```
nameAge = new Array(3);
```

RESULT:

The array will return: ,,

And the length property will return: 3

You can assign a value to index position [1].

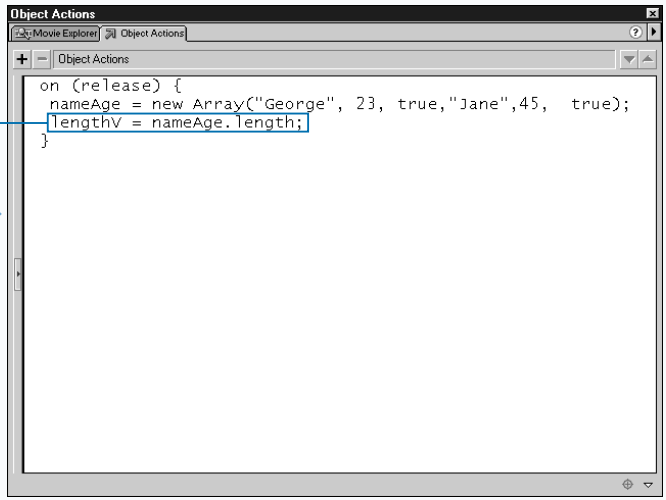
TYPE THIS:

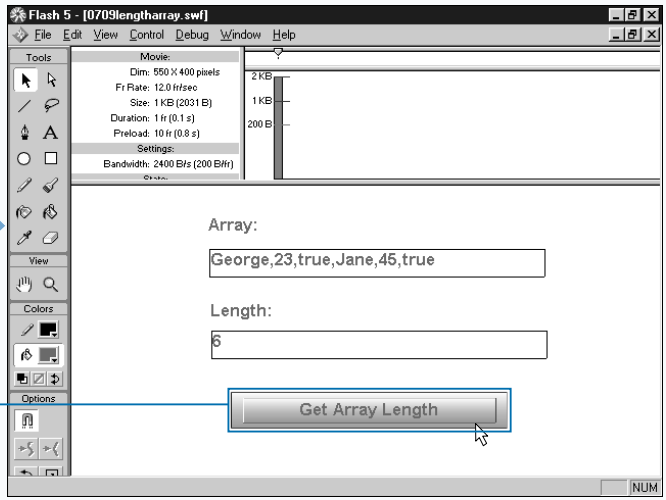
```
nameAge[1] = 23;
```

RESULTS:

The array will return: ,23,

And the length property will return: 3





4 Find the length of the array.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button to test your movie.

■ Clicking the button retrieves the array and the array length.

ADD ELEMENTS TO AN ARRAY

After you create an array, you might need to add additional elements. You can add them by assigning elements to an index position; however, the `push`, `unshift`, and `concat` methods provide more efficient ways to add a large number of elements to an array.

You use the `push` method to add elements to the end of an array. The `push` method adds one or more elements to the end of the array and returns the new length.

The syntax for the `push` method is

```
arrayName.push(v1,v2,...vN) .
```

Use the `arrayName` argument to specify the name of the array to which you want to add elements. Use arguments `v1` to `vN` to specify the elements you want to add.

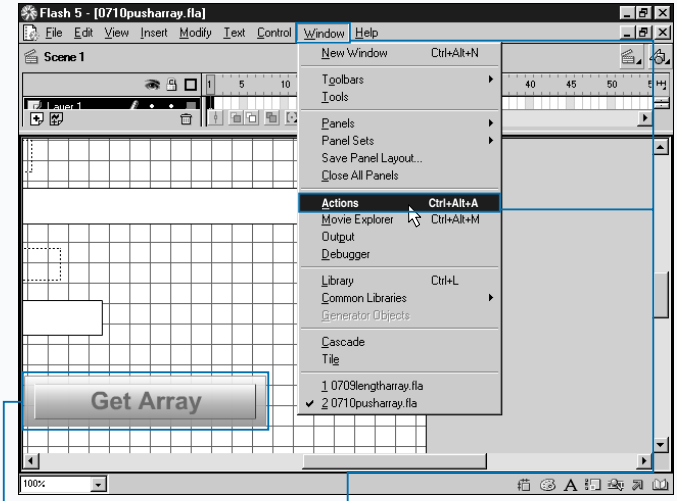
You can also use the `unshift` method to add elements to an array. The `unshift` method adds one or more elements to the beginning of the array and returns the new length. The syntax for the `unshift` method is

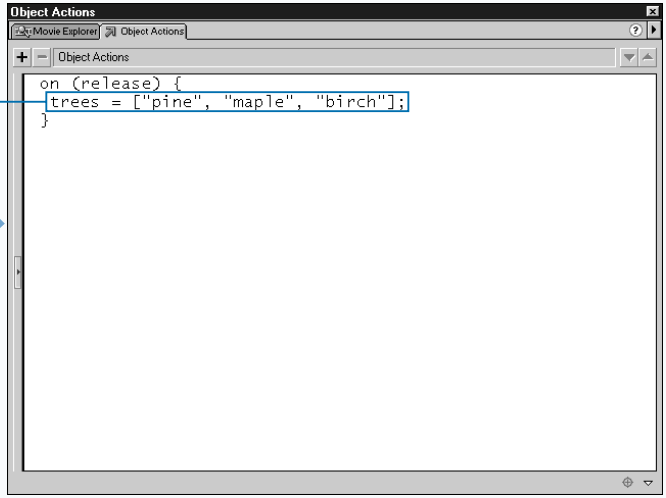
```
arrayName.push(v1, v2,...vN) .
```

Here, also, use the `arrayName` argument to specify the name of the array. Use arguments `v1` to `vN` to specify the elements you want to add.

You can use the `concat` method to create a new array by concatenating two or more arrays. The syntax for the `concat` method is `newArray = arrayName.concat(v1,v2,...vN)`. Use `newArray` to specify the name of the array you want to create by using the `concat` method. Use `arrayName` to specify the array to which you want to concatenate. Use arguments `v1` to `vN` to specify the arrays you want to concatenate to the array in the `arrayName` argument.

ADD ELEMENTS TO AN ARRAY





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file pusharray.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create an array.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

Extra

You can create an array.

Example:

```
trees = ["pine","maple","birch"];
```

You can add elements to the beginning of the array using the `unshift` method.

```
lengthV = trees.unshift("pear","apple");
```

The `trees` array will return:

```
pear,apple,pine,maple,birch
```

The `unshift` method will return:

```
5
```

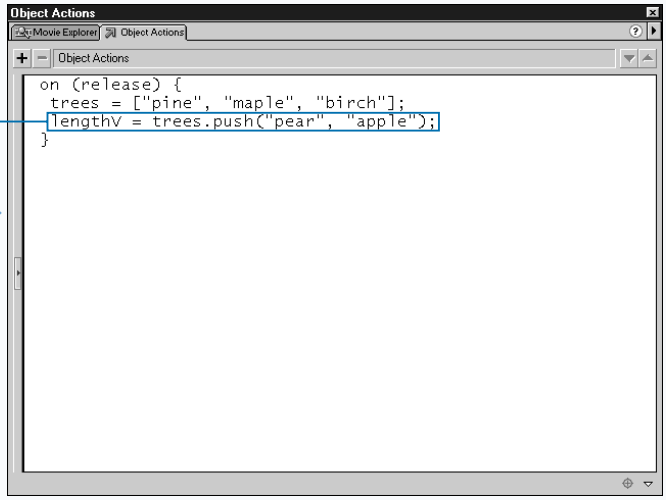
You can concatenate two or more arrays.

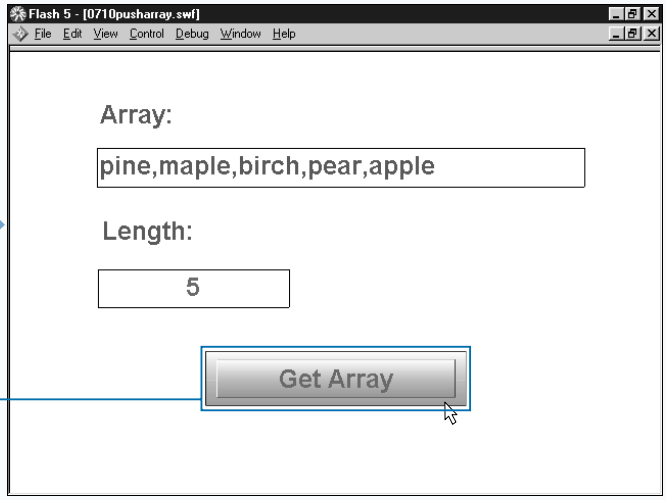
Example:

```
trees = ["pine","maple","birch"];
fruit = ["apple","pear","peach"];
fruitTrees = trees.concat(fruit);
```

The `fruitTrees` array will return:

```
pine,maple,birch,apple,pear,peach
```





4 Add elements.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button to test your movie.

■ Clicking the button retrieves the elements of the array and the array length.

REMOVE ELEMENTS FROM AN ARRAY

As you process information, you might need to retrieve an element value and remove it from the array at the same time. The `pop` and `shift` methods remove elements from an array. You use the `Array.pop` method to remove the last element from an array and retrieve its value.

The `pop` method uses the following Syntax:

```
arrayName.pop() .
```

Use the `arrayName` argument to specify the name of the array.

You use the `shift` method to remove the first element from an array and return its value. The `shift` method uses the following Syntax:

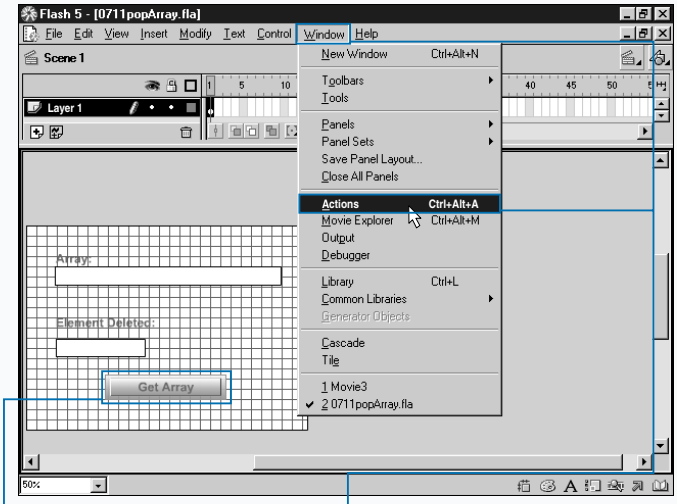
```
arrayName.shift() .
```

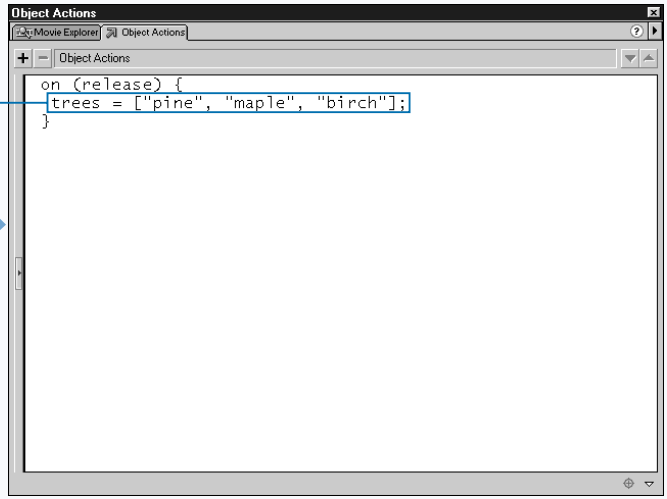
You can use the `splice` method to add, remove, or replace elements in an array. The `splice` method uses the following Syntax:

```
arrayName.splice(start,count,v1,v2,...vN); .
```

Use the `arrayName` argument to specify the array in which you want to add, remove, or replace elements. Use the `start` argument to specify the index value of the element where the insertion or deletion begins. Use the `count` argument to specify the number of elements to delete. The deletion includes the elements specified in the `start` argument. If you do not want to delete any elements, use `0` as the `count` argument. Use `v1` to `vN` to specify the values to insert at the point specified in the `start` argument. The `splice` method returns the deleted elements.

REMOVE ELEMENTS FROM AN ARRAY





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file poparray.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create an array.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

Apply It

You can use the `shift` method to remove and return the first element in an array.

TYPE THIS:

```
trees = ["pine","maple","birch"];
deleteV = trees.shift();
```

RESULT:

The variable `deleteV` returns `pine`, and the variable `trees` returns `maple,birch`.

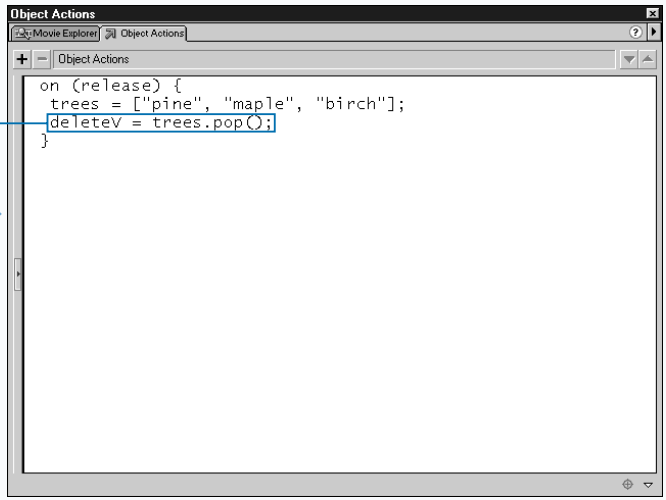
You can use the `splice` method to add, remove, or replace elements in an array. In the example shown here, you delete and add elements. The deletion begins at index position 1. The script deletes one element, and then adds the values apple and pear to the array.

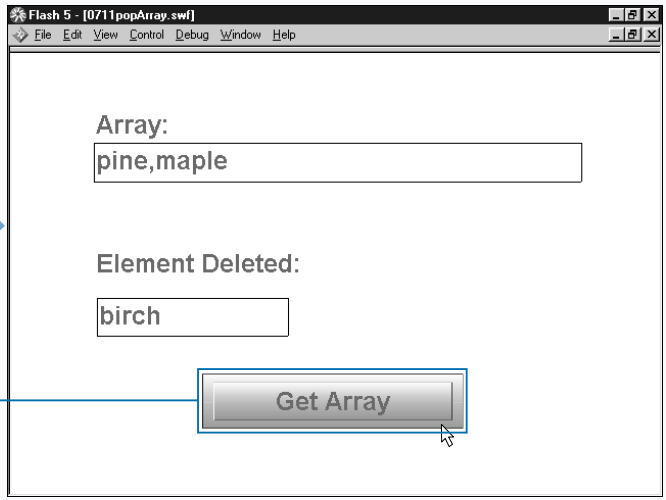
TYPE THIS:

```
trees = ["pine", "maple", "birch"];
trees.splice(1, 1, "apple", "pear");
```

RESULT:

The variable `trees` returns `pine,apple,pear,birch`.





4 Remove an element.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button to test your movie.

■ Clicking the button retrieves the current elements in the array and the deleted element.

EXTRACT OR REVERSE AN ARRAY

If you need to create an array from an existing array, you can use the `slice` method. The `slice` method extracts a section of an array and uses that section to create a new array.

The syntax for the `Array.slice` method is

```
newArray = arrayName.slice(start, end);.
```

Use the `newArray` argument to specify the name of the array you want to create. Use the `arrayName` argument to specify the name of the array from which you want to extract a section. Use the `start` argument to specify the index position for the point at which you want the extraction to start. Use a negative number if you want the count to begin at the final element. The final element has a value of -1. Use the `end` argument to specify the index position for the point at which you want the splice to end. If you do not include an `end` argument, ActionScript extracts all of the elements from the start position to the end of the array. Use a negative number if you want the count for the end argument to begin at the final element. The resulting array does not include the `end` argument element. For example, if you have the array:

```
colors = ["green","yellow","red","white",
"pink","orange"]
```

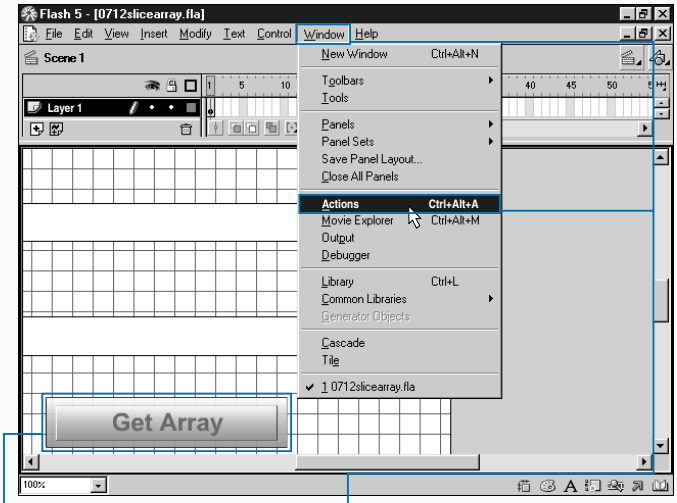
and you want to extract red, white, and pink and create an array called `myColors`, you use the syntax `mycolors = colors.slice(2,5);`.

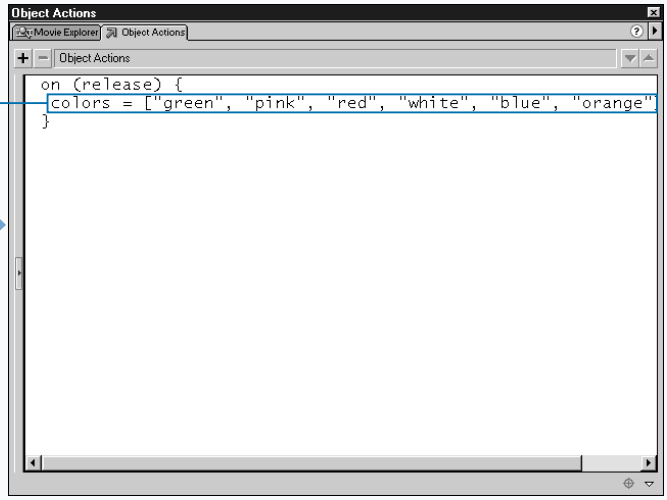
You can use the `reverse` method to reverse the order of an array. The syntax for the `Array.reverse` method is

```
arrayName.reverse().
```

Use the `arrayName` argument to specify the array you want to reverse. If an array includes the values 1, 2, 3 in that order, the `reverse` method returns 3, 2, 1.

EXTRACT OR REVERSE AN ARRAY





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file slicearray.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create an array.

■ If you associate the variables with dynamic text boxes, you can display the values the variables return to the user.

Extra

You use the `slice` method to extract a section of an existing array and use it to create a new array. If you find it easier to begin your count from the end of the array and work backwards, use negative numbers. If you have the array shown here.

```
trees = ["pine","maple","birch","apple","pear",
"peach","chestnut"];
```

Use this syntax to extract apple, pear, and peach and create an array named fruit.

```
fruit = trees.slice(-4,-1);.
```

The variable fruit will return: `apple,pear,peach.`

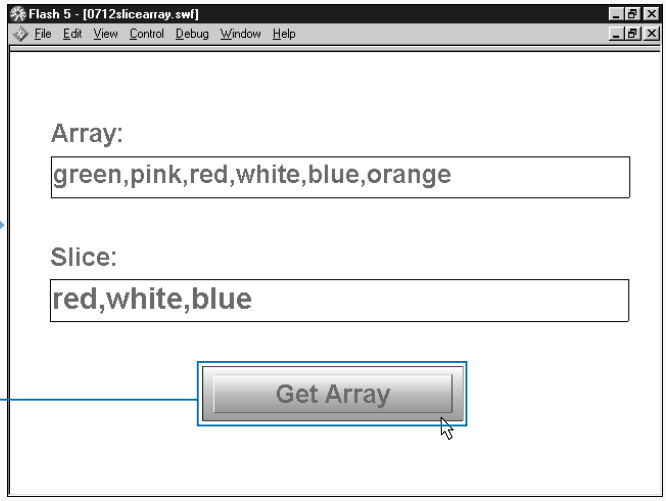
You use the `reverse` method to reverse the order of an array. If you have the array shown here.

```
numbers = [100, 200, 300, 400, 500];
```

Use the following syntax to reverse the array.

```
numbers.reverse ( ); 500,400,300,200,100.
```

The variable numbers will return `500, 400, 300, 200, 100.`



4 Extract a section of the array.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button to test your movie.

■ Clicking the button displays the original array and the array created by the slice.

CONVERT AN ARRAY TO A STRING

An array can contain strings, numbers, or Boolean values. At times, you might need to convert an array to a string to make manipulating or reading the contents of the array easier. After you convert an array to a string, you can use all the string methods to manipulate your data. That means that you can do things like convert the array to upper or lower case, find a substring, retrieve a character, or find the index value of a string. You can also concatenate the sting or assign the string to a variable.

You use the `toString` method to return every element in an array as a string. The `toString` method separates each element with commas. The syntax for the `Array.toString` method is

```
arrayName.toString();.
```

Use the `arrayName` argument to specify the name of the array whose elements you want to convert to a string.

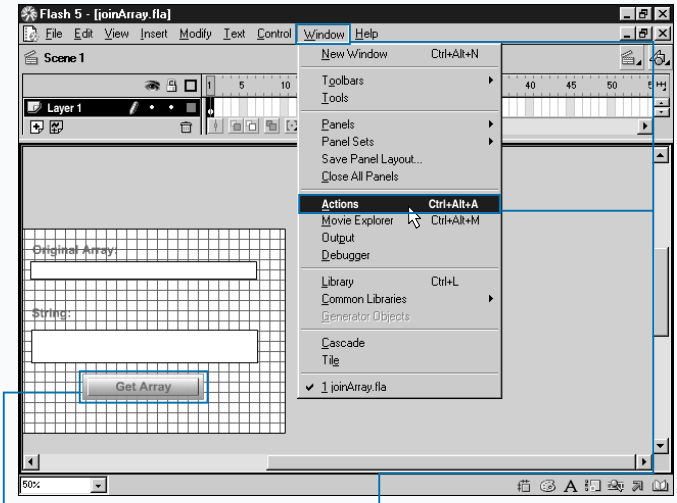
You can also use the `join` method to concatenate elements in an array and return a string. When you use the `join` method, you can specify the character or string you want to use to separate the elements in the array.

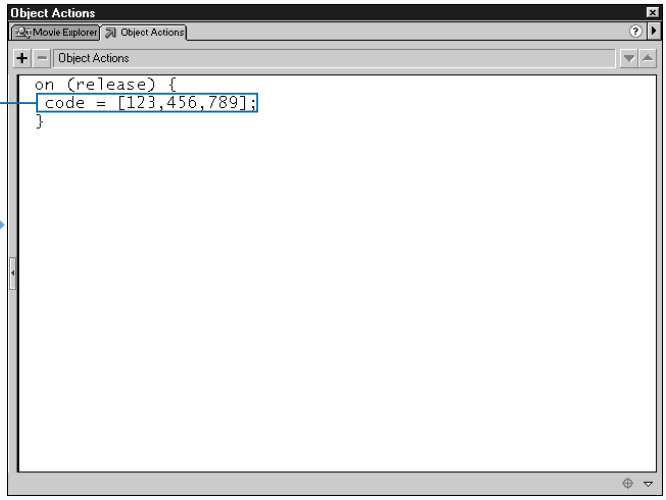
The syntax for the `Array.join` method is

```
ArrayName.join(separator).
```

Use the `arrayName` argument to specify the name of the array whose elements you want to concatenate. Use the `separator` argument to specify a character or string you want to use to separate the elements in the array. You can use any character or string of characters you want. If you do not specify a separator, ActionScript uses a comma.

CONVERT AN ARRAY TO A STRING





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file joinarray.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create an array.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

Extra

You can use the `Array.toString` method to return every element in an array as a string. If you have the array shown here:

Example

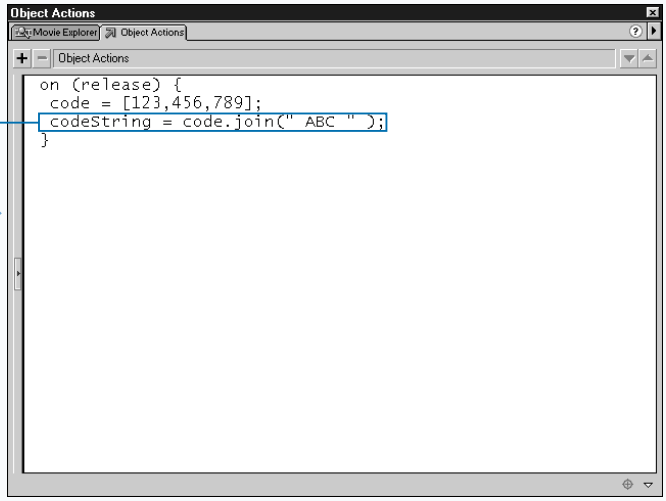
```
nameAge = ["George", 23, false,"Jane", 45, true];
```

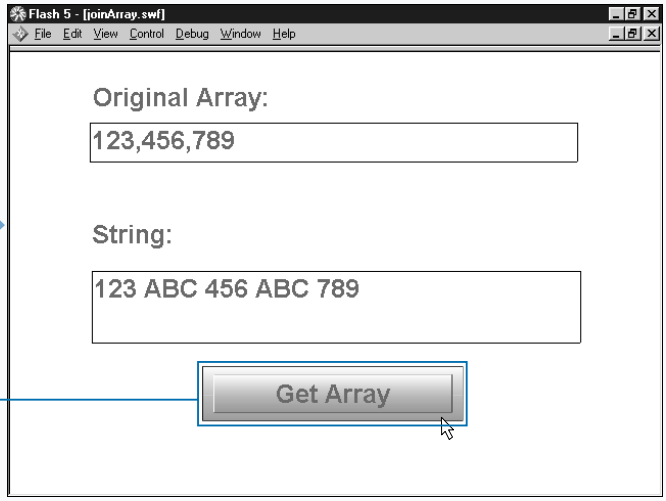
You can use this syntax to return the array as a string:

```
nameAge.tostring ( ) ; .
```

RESULT:

```
George,23,false,Jane,45,true.
```





4 Convert the array to a string.

■ Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

5 Click the button to test your movie.

■ Clicking the button displays the original array and the new string.

CHANGE COLORS

You can use Color object methods to change the color of movie clips located on the Stage. The Color object method enables you to change the color of movie clips as the movie plays.

Changing the color of a movie clip is a two-step process. First you use the constructor `new Color()` to create an instance of the Color object. Then you use the `Color.setRGB` method to specify the color to which you want to change the movie clip.

The syntax for the `new Color()` constructor is

```
objectName = new Color(target);.
```

Use the `objectName` argument to name the Color object. Use the `target` argument to specify the instance name of the movie clip for which you want to change the color.

The syntax for the `Color.setRGB` method is

```
objectName.setRBG(0xRRGGBB);.
```

Use the `objectName` argument to specify the Color object. Each color has a hexadecimal value. Use RRGGBB to specify the hexadecimal value that represents the color to which you want to change the movie clip. There is a list of hexadecimal values in the Appendix of this book. Flash displays hexadecimal values next to the selected color in the Stroke and Fill panels.

The following example changes the color of a movie clip:

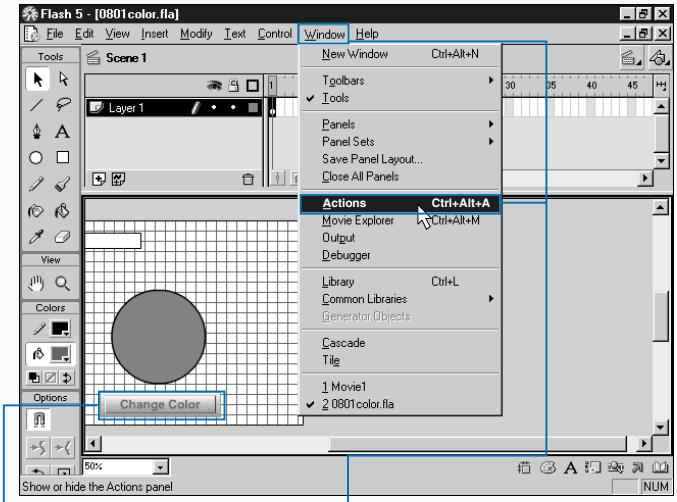
```
newColor = new Color(sampleMC);.newColor.
setRGB(0xFF0000);.
```

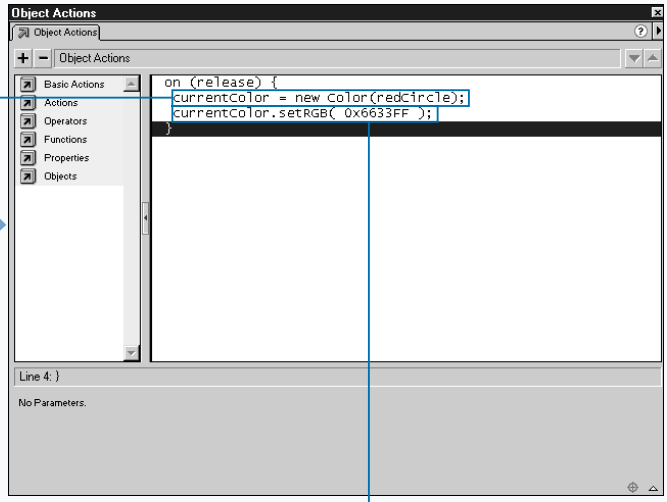
You use the `getRGB` method to retrieve the numeric value of the most recent `setRBG` call. The syntax for the `Color.getRGB` method is

```
objectName.getRGB();.
```

Use the `objectName` argument to specify the Color object. The `setRGB` method returns the color value in base 10.

CHANGE COLORS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file color.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create an instance of the Color object using the `new Color` constructor.

4 Set the color using `setRGB`.

Extra

Usually, you count in base 10. Hexadecimal numbers are numbers in base 16. Understanding base 16 is easy. You count from 0 to 9, and then you add a column and make 10. When counting in hexadecimal, counting goes from 0 to F before adding a column. Instead of counting

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32
```

you count

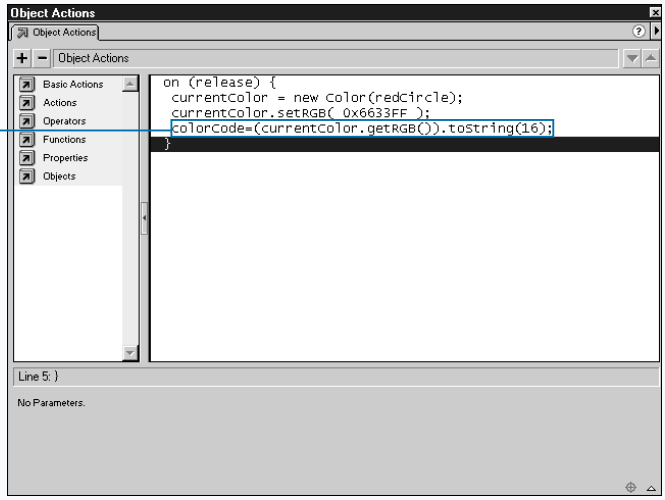
```
1 2 3 4 5 6 7 8 9 a b c d e f 10 11 12 13 14 15 16 17 18 19 1a 1b
1c 1d 1e 1f 20 21 22 23 24
```

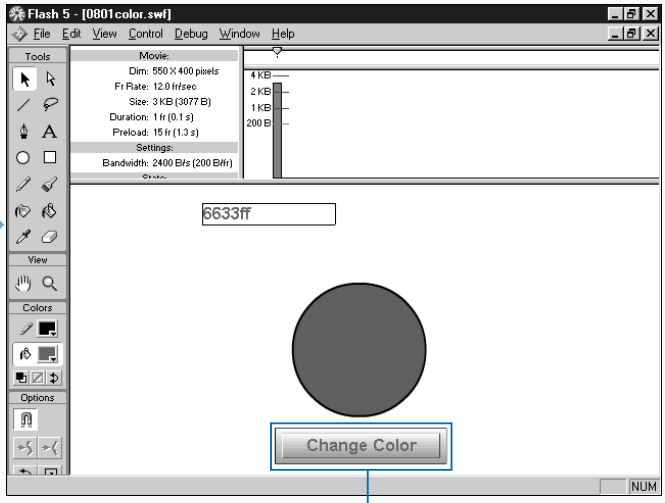
The `getRGB` method returns the color value in base 10. You can use the `Number.toString` method to convert a base 10 number to base 16.

The syntax for the `Number.toString` method is

```
number.toString(radix) .
```

Use the number argument to specify the number you want to convert to a string. Use the radix argument to specify the numeric base. You can specify any base from 2 to 36. If you do not specify a base, ActionScript uses base 10. This example converts the results of a `getRGB` call to base 16 or hexadecimal: `(newColor.getRGB()).toString(16);` .





5 Get the hexadecimal code using `getRGB`.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

6 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

7 Click the button to test your movie.

■ Clicking the button changes the color of the movie clip and displays the hexadecimal code.

WORK WITH COLOR TRANSFORM VALUES

If you want a greener green, a deeper purple, or a rosier red, you want to use the `setTransform` method. The `setTransform` method enables you to make subtle adjustments to color and change the alpha value. You can adjust the percentage of red, green, blue, or alpha from +100 to –100 percent. You can also adjust the red, green, blue, or alpha offset values by an amount ranging from –255 to +255. Flash combines the adjustments you make to produce a new color.

The parameters for making color adjustments are `ra`, `rb`, `ga`, `gb`, `ba`, `bb`, `aa`, and `ab`. Parameters beginning with r adjust the red value, parameters beginning with g adjust the green value, parameters beginning with b adjust the blue value, and parameters beginning with a adjust the alpha value. Parameters ending with a adjust the percentage. Parameters ending with b adjust the offset value.

Setting a color transform is a four-step process: 1) Use the constructor `new Color()` to create an instance of the Color object. Refer to Chapter 7 for more information on creating objects. 2) Create a generic object. Refer to Chapter 7 for more information on creating generic objects. 3) Assign values to the `Color.setTransform` properties. 4) Set the color transform.

You use the `Color.setTransform` method to set the color transform. The syntax for the `Color.setTransform` method is

```
objectName.setTransform(transformObject);.
```

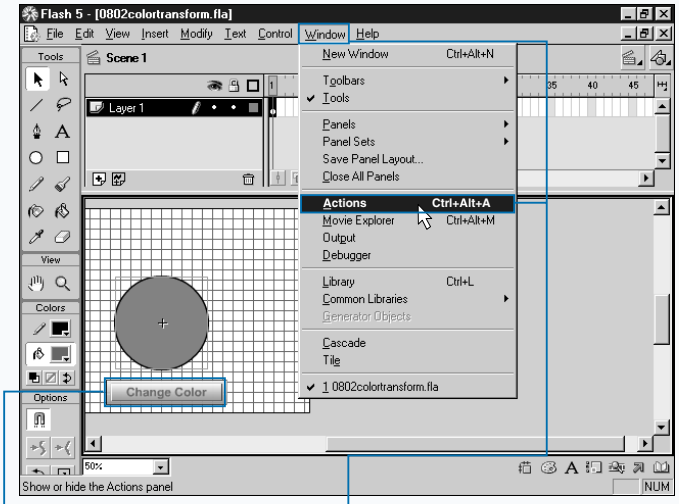
Use the `objectName` argument to specify the variable to which you stored the Color object. Use the `transformObject` argument to specify the variable to which you assigned the new object.

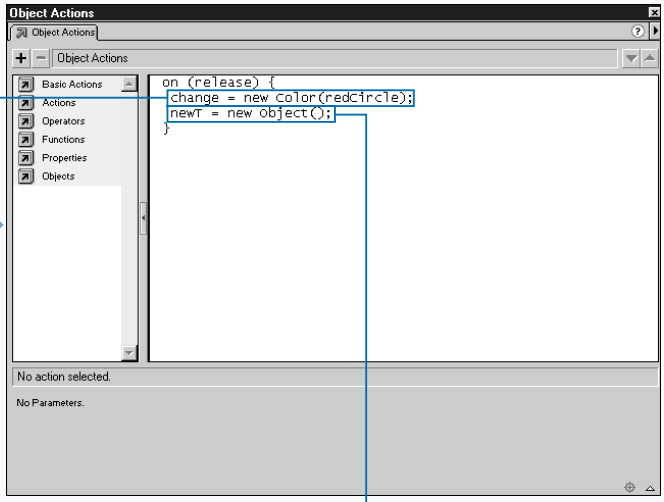
You use the `Color.getTransform()` method to retrieve the transform values set by the most recent `setTransform` call. The syntax for the `Color.getTransform()` method is

```
objectName.getTransform().
```

Use the `objectName` argument to specify the variable to which you stored the Color object.

WORK WITH COLOR TRANSFORM VALUES





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file colortransform.fla, which you can find on the CD-ROM that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create a new Color object using the `new Color` constructor.

4 Create a new generic object using `new Object`.

Apply It

If you need to find the value of color transform properties, you can retrieve them.

TYPE THIS:

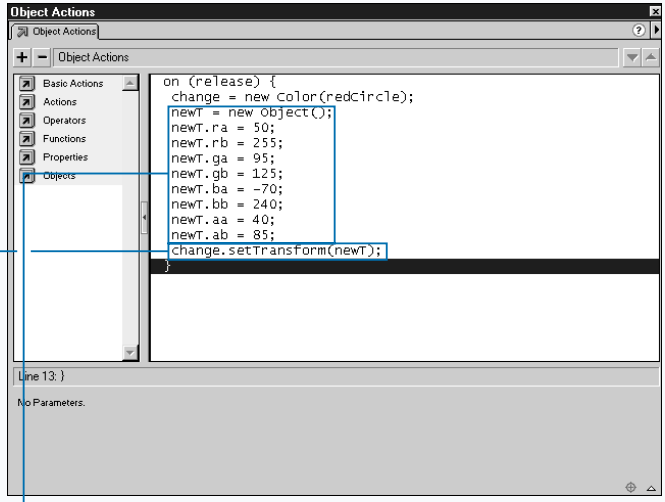
```
// Gets the transform properties
colorCode = change.getTransform():
raV = colorCode.ra;
rbV = ColorCode.rb;
gaV = colorCode.ga;
gbV = colorCode.gb;
baV = colorCode.ba;
bbV = colorCode.bb;
aaV = colorCode.aa;
abV = colorCode.ab;
```

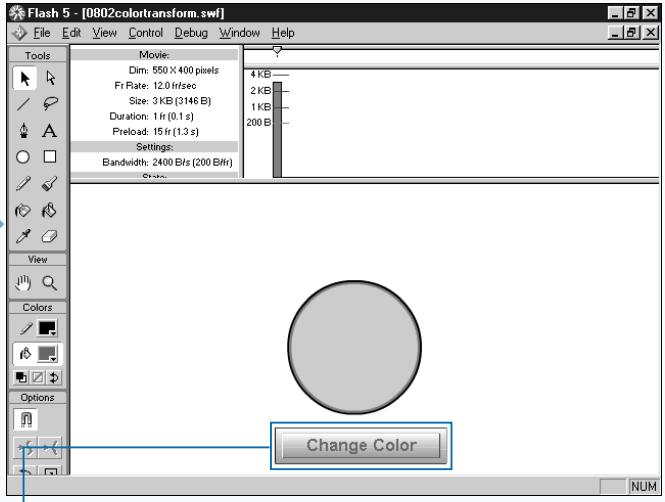
RESULT:

Flash retrieves the color transform values.

ra=50, r=225, ga=95, gb=125, ba =-70, bb = 240, aa = 40, ab = 85

Values retrieved may not be exact.





5 Assign color.setTransform values to the properties.

6 Set the color transform.

7 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

8 Click the button to test your movie.

■ Clicking the button applies the color transform.

USING THE MOUSE OBJECTS AND PROPERTIES

The mouse pointer is usually a small arrow. You can make it a dog, or perhaps you want a wand, a pencil, or a heart. Whatever you want, you can have it. You use the Mouse object to show or hide the mouse pointer. Hiding the mouse pointer enables you to create a custom pointer. You can make any movie clip a custom pointer. Because movie clip pointers are so flexible, you can create special effects with custom pointers.

Use the `Mouse.hide` method to hide the mouse pointer. The syntax for the `Mouse.hide` method is

```
Mouse.hide();.
```

The `Mouse.hide` method does not take any arguments.

After you have hidden the mouse, you use the `Mouse.show` method to make the pointer visible again. The syntax for the `Mouse.show` method is

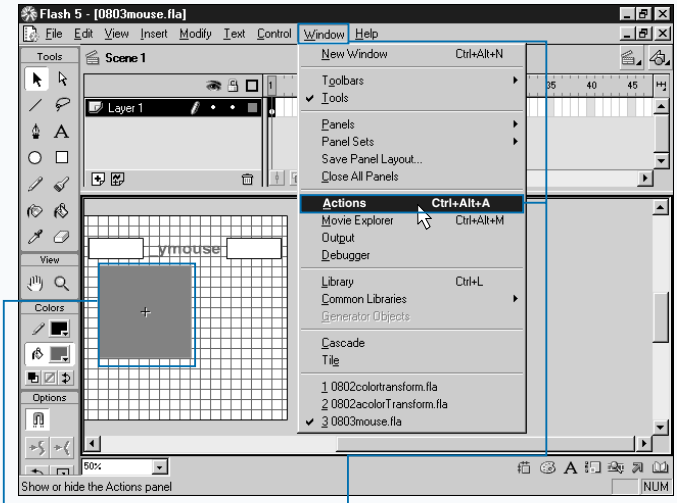
```
Mouse.show();.
```

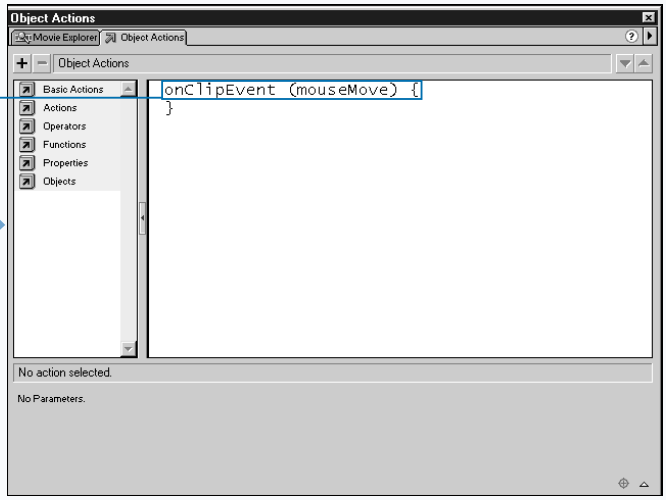
The `Mouse.show` method does not take any arguments.

If you need to know the x and y coordinates of the mouse, use the `_xmouse` and `_ymouse` properties to retrieve them. The `_xmouse` and `_ymouse` properties are read-only, which means you can retrieve the `properties` but you cannot set them. The `_xmouse` property returns the x-coordinate of the mouse location. The `_ymouse` property returns the y-coordinate of the mouse location.

The syntax for the `_xmouse` and `_ymouse` properties is `instanceName._property;`. Use the `instanceName` argument to specify the name of the movie clip for which you want to retrieve the `_xmouse` or `_ymouse` property. Use the `_property` argument to specify the property you want to retrieve.

USING THE MOUSE OBJECTS AND PROPERTIES





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file mouse.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Set the clip event to `mouseMove`.

■ This will cause the script to execute each time the mouse moves.

Apply It

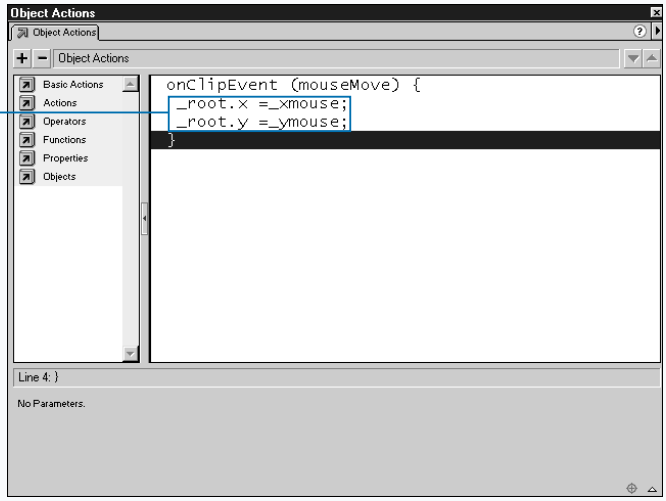
Use the script shown here to create a custom pointer. You can associate it with any movie clip. The script hides the default mouse pointer when the movie clip to which it is attached loads. The script then makes the movie clip draggable, thereby turning the movie clip into a custom pointer. You should place a custom pointer on the top layer of the timeline so that it appears in front of all other objects on the Stage.

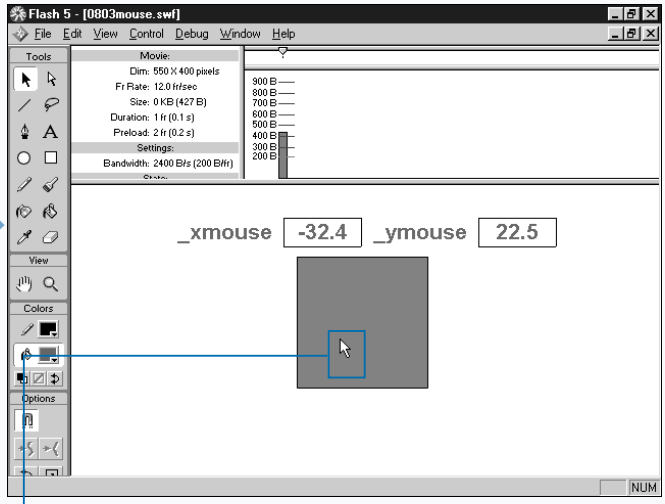
TYPE THIS:

```
onClipEvent (load) {
 Mouse.hide();
 startDrag (this, true);
}
```

RESULT:

The movie clip to which you attached the script becomes the pointer.





4 Assign `_xmouse` and `_ymouse` to variables.

■ If you associate the variables with dynamic text boxes, you can display the values the variables return to the user.

5 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

6 Drag your mouse to test your movie.

■ As you drag your mouse, you will see the `_xmouse` and `_ymouse` values change.

USING THE DATE OBJECT

Flashplayer uses the system clock on the computer running Flashplayer to determine the date and time. You can use the information retrieved to perform many types of date arithmetic, including the calculation of elapsed time and the calculation of the amount of time between two dates.

Before calling date methods, you must use the `new Date` constructor to create an instance of the Date object. Use the syntax `objectName = new Date();` . Use the `objectName` argument to name the Date object.

ActionScript provides several methods you can use to retrieve date and time values using the syntax `objectName.method();` . Use the `objectName` argument to specify the name you assigned to the Date object. Use the method argument to specify the method you want to use.

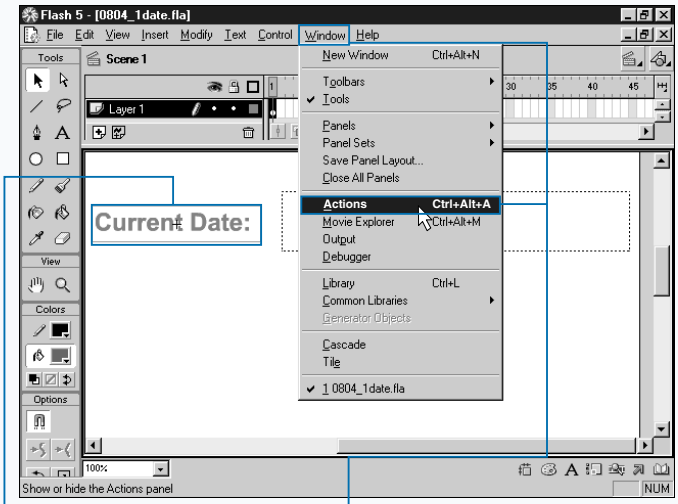
Some date and time methods retrieve the local date and time. Other date and time methods retrieve the date and time based on Greenwich Mean Time — also referred to as Universal Coordinated Time.

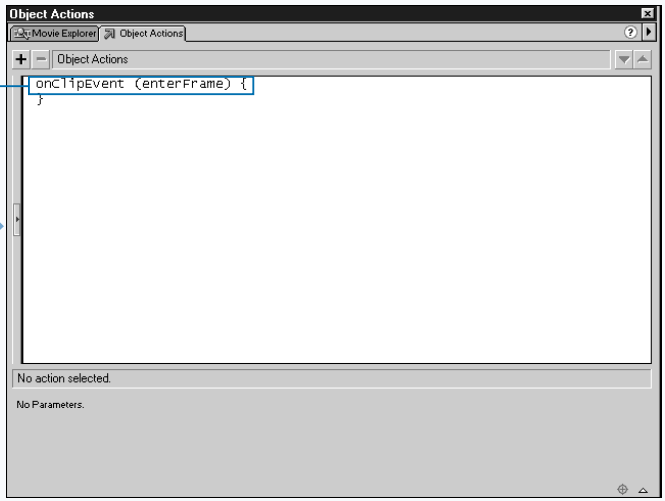
Use the `getMonth()` method to retrieve the current month based on local time. Use the `getUTCMonth()` method to retrieve the current month based on Universal Coordinated Time. The `getMonth()` and `getUTCMonth()` methods return an integer from 0 to 11. The value 0 represents January, the value 1 represents February, and so forth.

Use the `getDate()` method to retrieve the day of the month based on local time. Use the `getUTCDate()` method to retrieve the current day based on Universal Coordinated Time. The `getDate()` and the `getUTCDate()` methods return an integer from 1 to 31, representing the day of the month.

Use the `getFullYear()` method to retrieve a four-digit number representing the year based on local time. Use the `getUTCFullYear()` method to retrieve a four-digit number representing the year based on Universal Coordinated Time.

USING THE DATE OBJECT





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file date.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Set the event to `enterFrame`.

■ This causes the script to execute each time the cursor enters the frame.

Apply It

Flash queries the system clock of the computer that is running Flashplayer and uses the information returned to determine date and time values. If the system clock is incorrect, the retrieved values will be incorrect.

You can use the `new Date` constructor to retrieve the current date.

TYPE THIS:

```
currentDate = new Date();
```

RESULT:

The `new Date` constructor returns the current date in the format shown here.:

`Sun Oct 7 08:39:05 GMT-0400 2002.`

You can use the `getMonth` method to return the current month.

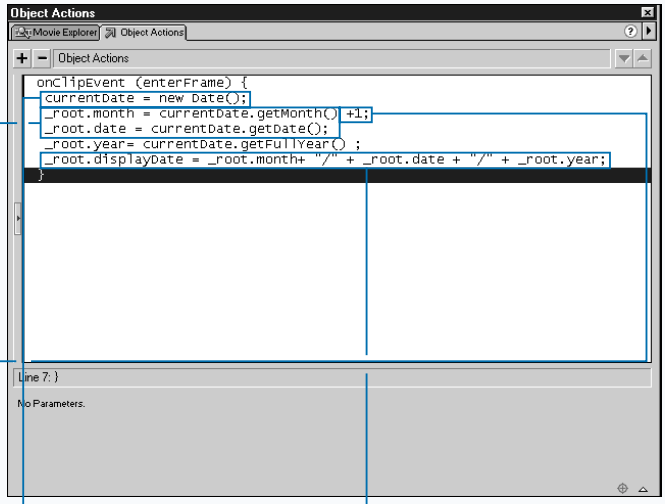
TYPE THIS:

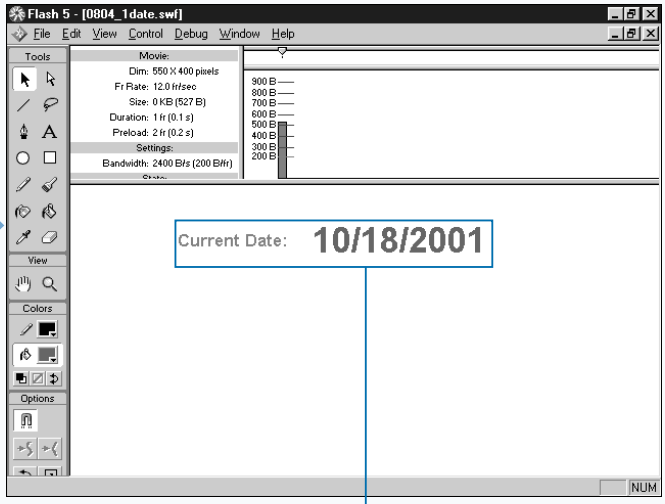
```
currentDate = new Date();
month = currentDate.getMonth();
```

RESULT

If February is the current month, ActionScript will return 1.

The `getFullYear()` and `getUTCFullYear()` methods return a four-digit year. They return 2002 to represent the year 2002. You can use the `getYear()` method to retrieve the year based on local time. The `getYear()` method returns the full year minus 1900. The `getYear()` method returns 102 to represent the year 2002.





4 Create an instance of the Date object.

5 Get the current month, date, and full year.

■ Add +1 to the month value because the month value returns 0 for January and 1 for February and so forth.

6 Concatenate the values and assign them to a variable.

■ This puts the date in a standard date format. If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

7 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

8 Test your movie.

■ When you enter the test environment, the screen will automatically display the current date.

GET DATE AND TIME VALUES

ActionScript date and time values make it easy to create a clock. Just retrieve the hour, minute, and second and display them on the screen and you have a real time system clock.

Use the `getDay()` method to retrieve the day of the week based on local time. Use the `getUTCDay()` method to retrieve the day of the week based on Universal Coordinated Time. The `getDay()` and `getUTCDay()` methods return 0 to represent Sunday, 1 to represent Monday, and so forth.

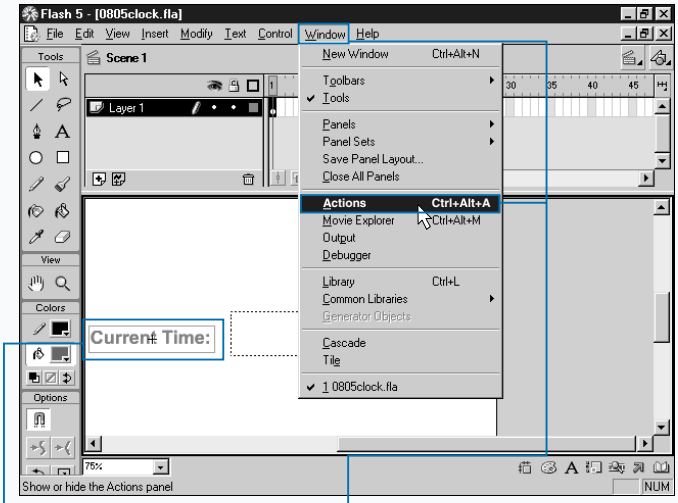
Use the `getHours()` method to retrieve the hour of the day based on local time. Use the `getUTCHours()` method to retrieve the hour of the day based on Universal Coordinated Time. The `getHours()` and `getUTCHours()` methods return an integer from 0 to 23. The value 0 represents midnight and the value 23 represents 11 p.m.

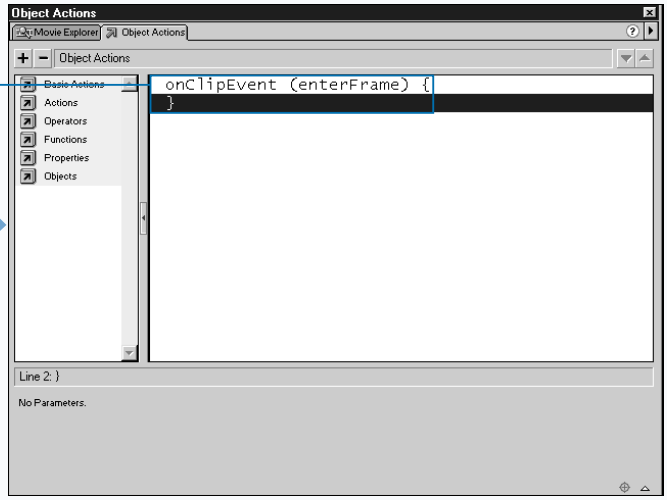
Use the `getMinutes()` method to retrieve the number of minutes that have elapsed in the current hour based on local time. Use the `getUTCMinutes()` method to retrieve the number of minutes that have elapsed in the current hour based on Universal Coordinated Time. The `getMinutes()` and `getUTCMinutes()` methods return an integer from 0 to 59.

Use the `getSeconds()` method to retrieve the number of seconds that have elapsed in the current minute based on local time. Use the `getUTCSeconds()` method to retrieve the number of seconds that have elapsed in the current minute based on Universal Coordinated Time. The `getSeconds()` and the `getUTCSeconds()` methods return an integer from 0 to 59.

Use the `getMilliseconds()` method to retrieve the number of milliseconds that have elapsed in the current second based on local time. Use the `getUTCMilliseconds()` method to retrieve the number of milliseconds that have elapsed in the current second based on Universal Coordinated Time. The `getMilliseconds()` and `getUTCMilliseconds()` methods return an integer from 0 to 999.

GET DATE AND TIME VALUES





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file clock.fla, which you can find on the CD that accompanies this book.

2 Click Windows ➪ Actions to open the Actions panel.

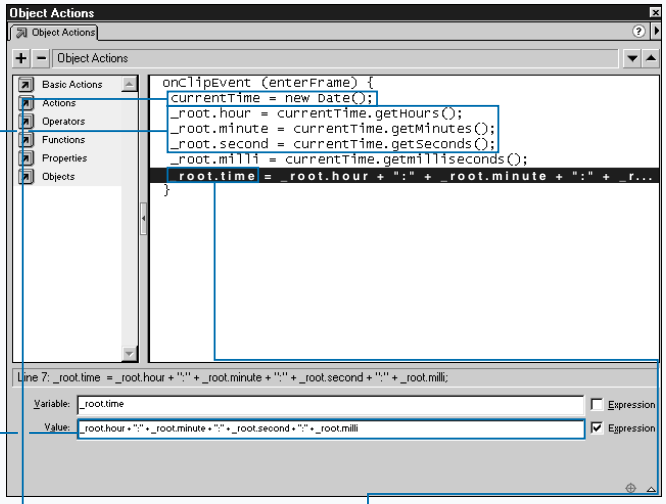
3 Set the event to `enterFrame`.

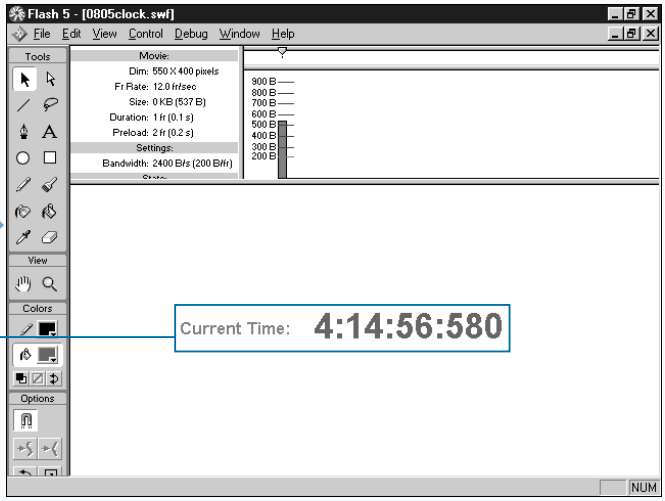
■ This causes the script to execute each time the cursor enters the frame.

Extra

A Web page user can be in any time zone. When displaying information that is time specific, you can display the information in universal time and provide the user with the difference between universal time and local time. Use the `getTimezoneOffset` method to retrieve the difference in minutes between local time and Universal Coordinated Time. The `getTimezoneOffset` method does not need to make adjustments for daylight savings time because the computer system clock makes the adjustments.

Use the `date.getTime` method to retrieve the number of milliseconds since midnight January 1, 1970, universal time. This method is useful when you need to compare an instant in time across two or more time zones. You may want to use this method if you create a game for players in different time zones. Using this method, you can calculate the elapsed time for each player using the same base.





4 Create an instance of the Date object.

5 Assign the date values to variables.

6 Concatenate the values.

■ This gives the values the appearance of a clock.

7 Assign concatenated values to a variable.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

8 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

9 Test your movie.

■ When you enter the test environment, the screen will automatically display the time.

SET THE DATE

The set date methods are useful when you want to obtain date information from the user and when you want to perform date arithmetic. The set date methods enable you to set a date.

Use the `new Date` constructor to create a Date object with a specific date and time, using the syntax

```
objectName = new Date(year,month,date,
hour,minute,second);.
```

Use the `objectName` argument to name the object. Use the `year` argument to specify the year. You can use 00 to 99 to indicate a `year` between 1900 and 1999. Otherwise, use four digits to indicate the year. Use the `month` argument to indicate the month. Use 0 to represent January, 1 to represent February, and so forth. Use the `date` argument to specify the day of the month. Use an integer from 1 to 31. The `date` argument is optional. Use the `hour` argument to specify the hour. Use an integer from 0 to 23. Use 0 to represent midnight and 23 to represent 11 p.m. The `hour` argument is optional. Use the `minute` argument to specify the minute. Use an integer between 0 and 59. The `minute` argument is optional. Use the `second` argument to specify the second. Again, use an integer between 0 and 59. The `second` argument is also optional.

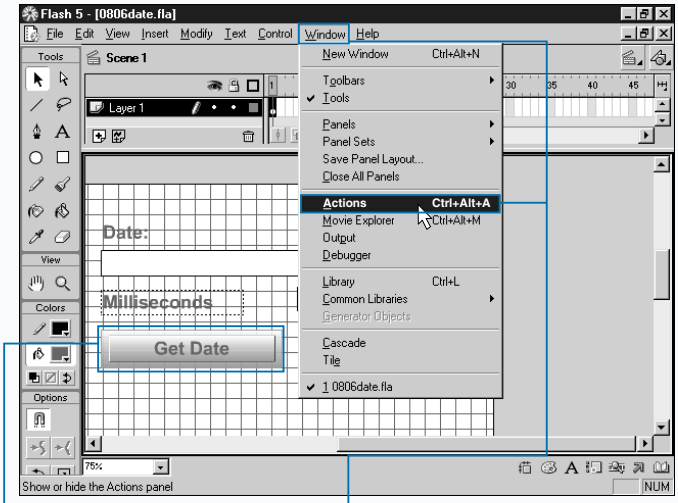
Some date methods set date and time by using the local time. Other date and time methods set the date and time based on Universal Coordinated Time.

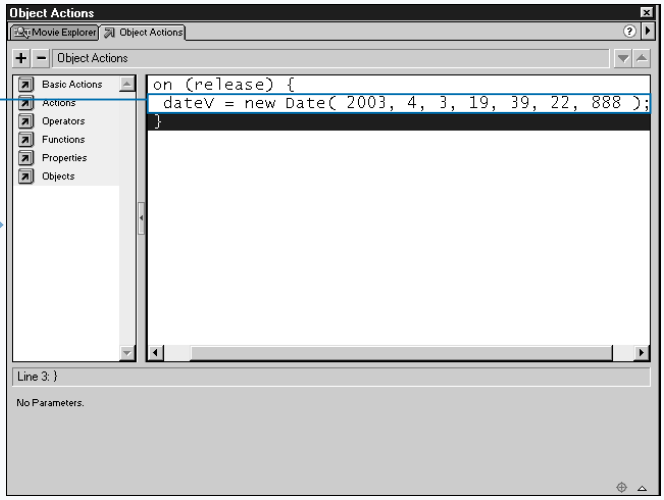
Use the `Date.toString` method to convert a Date object to a string. The `Date.toString` method uses the syntax

```
objectName.toString();.
```

Use the `objectName` argument to specify the Date object you want to convert to a string.

SET THE DATE





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file date.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create an instance of the Date object.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

Apply it

ActionScript stores dates as the number of milliseconds since January 1, 1970, 00:00:00. The Date object range is -100,000,000 days to 100,000,000 days relative to January 1, 1970 UTC.

If you need to calculate age, use the script that follows. In this script, the user inputs his birth date, and the script calculates his age. The variables `monthV`, `DayV`, and `YearV` are input text fields used to collect the date of birth of the user. The script subtracts the current date from the birth data and converts milliseconds to years to determine the age.

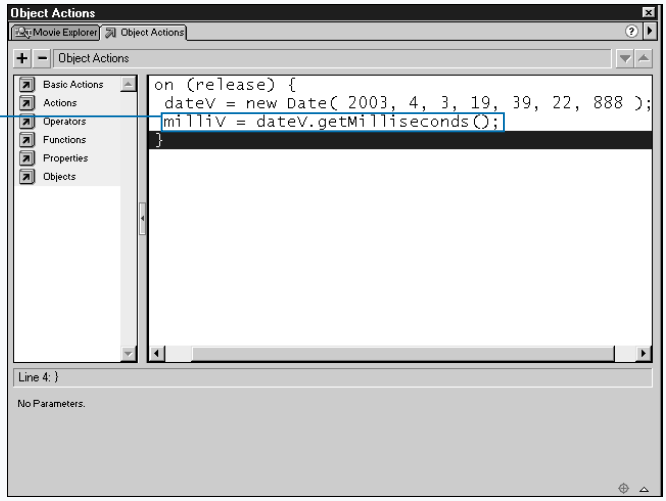
TYPE THIS:

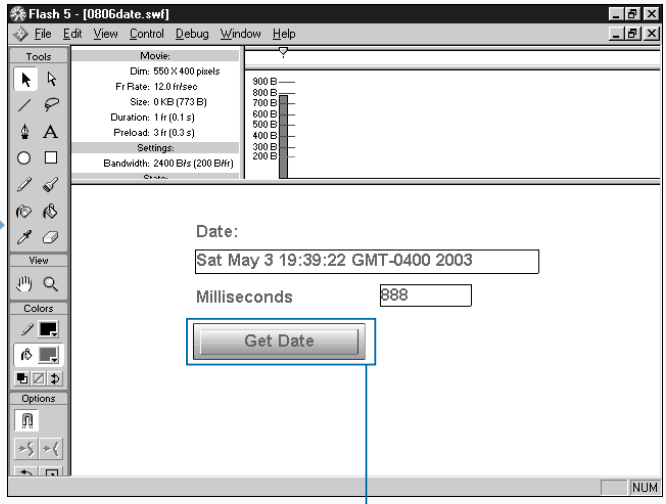
Button

```
on (release) {
     monthU = monthV-1;
     birth = new Date( yearV,MonthU, DayV );
     today = new Date();
     age = Math.floor ( (((((today-birth)/1000)/60)/60)/24)/365);
     result = "You are " + age + " years old.";
}
```

RESULT:

The script calculates an age based on the data entered.





4 Get the milliseconds.

■ By default, the Date object does not display milliseconds.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

6 Click the button to test your movie.

■ The screen displays the Date object.

SET DATE VALUES

With ActionScript, you can set the date in local time or in universal time. You can set a birthdate, the date a project is due, or any other date you want to set.

Use the `setMonth` method to set the month in local time. Use the `setUTCMonth` method to set the month in Universal Coordinated Time. The syntax for the `setMonth` method is

```
objectName.setMonth(month,date);.
```

The syntax for the `setUTCMonth` method is

```
objectName.setUTCMonth(month,date);.
```

Use the `month` argument to specify the month. Use 0 to represent January, 1 to represent February, and so forth. Use the `date` argument to specify the day of the month. Use an integer from 1 to 31. The `date` argument is optional.

Use the `setDate` method to set the day of the month in local time. Use the `setUTCDate` method to set the day of the month in Universal Coordinated Time. The syntax for the `setDate` method is

```
objectName.setDate(date);.
```

The syntax for the `setUTCDate` method is

```
objectName.setUTCDate(date);.
```

Use the `date` argument to specify an integer from 1 to 31 representing the appropriate day of the month.

Use the `setFullYear` method to set a four-digit number representing the year in local time. Use the `setUTCFullYear` method to set a four-digit number representing the year in Universal Coordinated Time. The syntax for the `setFullYear` method is

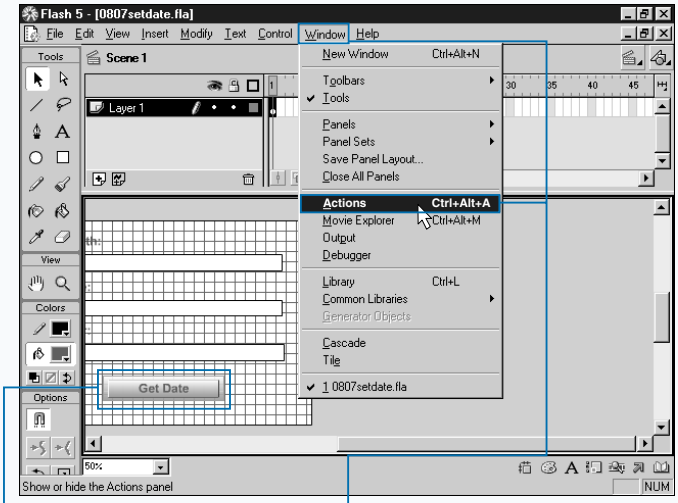
```
objectName. setFullYear(year,month,date);.
```

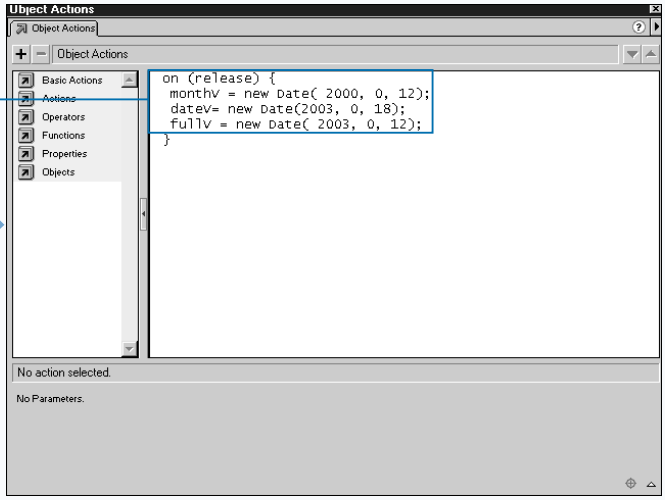
The syntax for the `setUTCFullYear` method is

```
objectName. setUTCFullYear(year,month,date);.
```

Use the `year` argument to specify a four-digit year. Use the `month` argument to specify an integer from 0 to 11 that represents the month. Use 0 to represent January, 1 to represent February, and so forth. Use the `date` argument to specify an integer from 1 to 31 to represent a day of the month.

SET DATE VALUES





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file setdate.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create instances of the Date object.

■ If you associate the variables with a dynamic text box, your can display the value the variables return to the user.

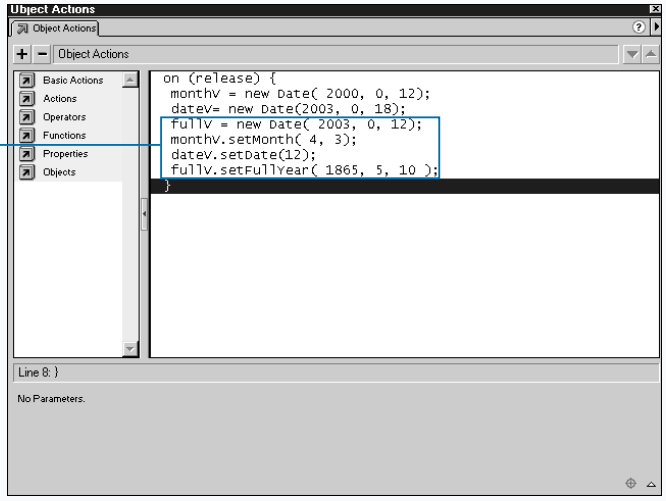
Apply It

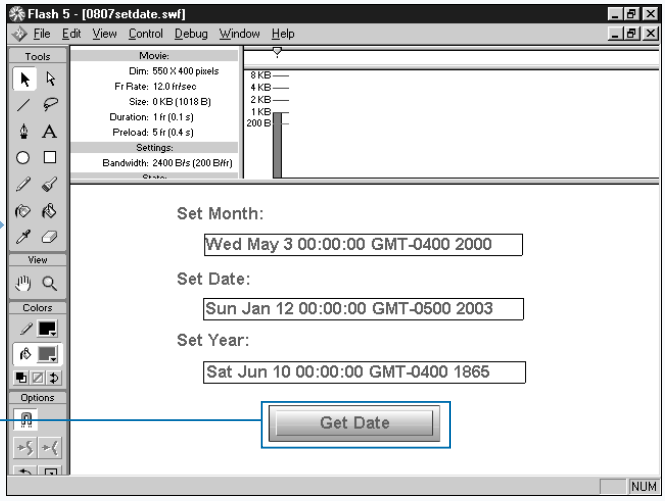
You can update the date. For example, if you set the date to January 18, 2003 and you later want to set the date to January 12, 2003, you can use the `setDate` method to change the day without affecting the month and the year.

TYPE THIS:	RESULT:
`dateV = new Date(2003, 0, 18)`	The date is set to January 18, 2003.

TYPE THIS:	RESULT:
`dateV.setDate(12)`	The date is reset to January 12, 2003.

You can use the `setYear` method to set a four-digit year in local time. The syntax for the `setYear` method is `objectName.setYear(year);`. Use the `year` argument to specify a four-digit year.





4 Change the date values.

5 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

6 Click the button to test your movie.

■ The screen displays the changes you made to the Date objects.

SET TIME VALUES

You can use time values to record time. Using ActionScript you can set the time down to the millisecond. Use the `setHours` method to set the hour in local time. Use the `setUTCHours` method to set the hour in Universal Coordinated Time. The syntax for the `setHours` method is `objectName.setHours(hour);`.

The syntax for the `setUTCHours` method is

`objectName.setUTCHours(hour);`.

Use the hour argument to specify an integer from 0 to 23. The value 0 represents midnight; the value 23 represents 11 p.m.

Use the `setMinutes` method to set the number of minutes that have elapsed in local time. Use the `setUTCMinutes` method to set the number of minutes that have elapsed in Universal Coordinated Time. The syntax for the `setMinutes` method is `objectName.setMinutes(minute);`.

The `setUTCMinutes` method syntax is

`objectName.setUTCMinutes(minute);`.

Use the `minute` argument to specify an integer from 0 to 59 that represents the number of minutes.

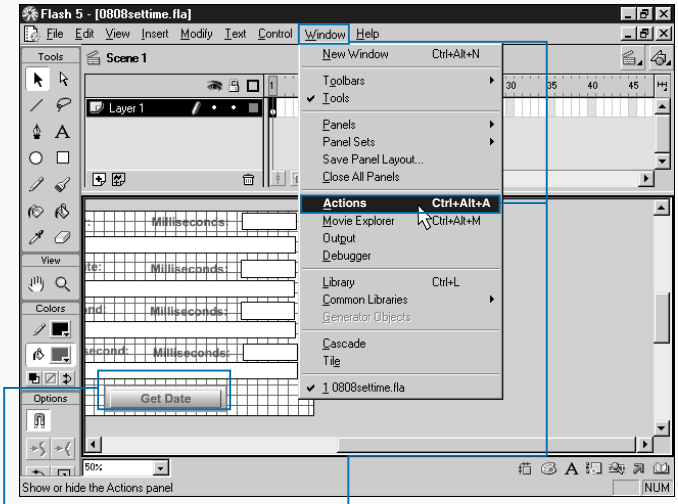
Use the `setSeconds` method to set the number of seconds that have elapsed in a minute in local time. Use the `setUTCSeconds` method to set the number of seconds that have elapsed in a minute in Universal Coordinated Time. The syntax for the `setSeconds` method is

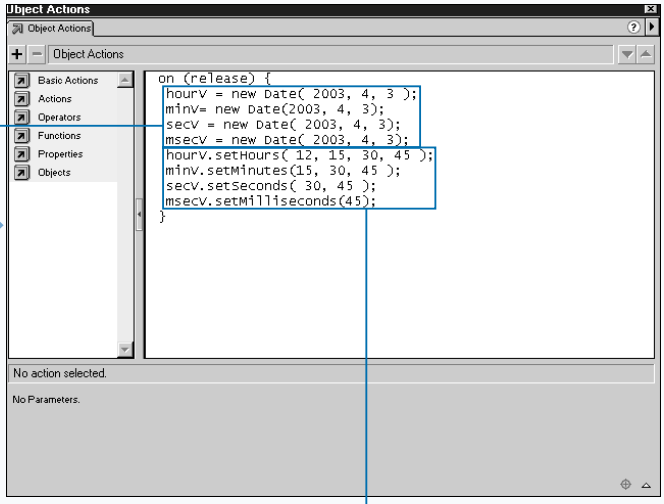
`objectName.setSeconds(second);`.

The syntax for the `setUTCSeconds` method is `objectName.setUTCSeconds(second);`. Use the `second` argument to specify an integer from 0 to 59 that represents the number of seconds.

Use the `setMilliseconds` method to set the number of milliseconds that have elapsed in a second in local time. Use the `setUTCMilliseconds` method to set the number of milliseconds that have elapsed in Universal Coordinated Time. The syntax for the `setMilliseconds` method is `objectName.setMilliseconds(millisecond);`. The syntax for the `setUTCMilliseconds` method is `objectName.setUTCMilliseconds(millisecond);`. Use the `millisecond` argument to specify an integer from 0 to 999 that represents the number of milliseconds.

SET TIME VALUES





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file settime.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create instances of the Date object.

4 Change the time values.

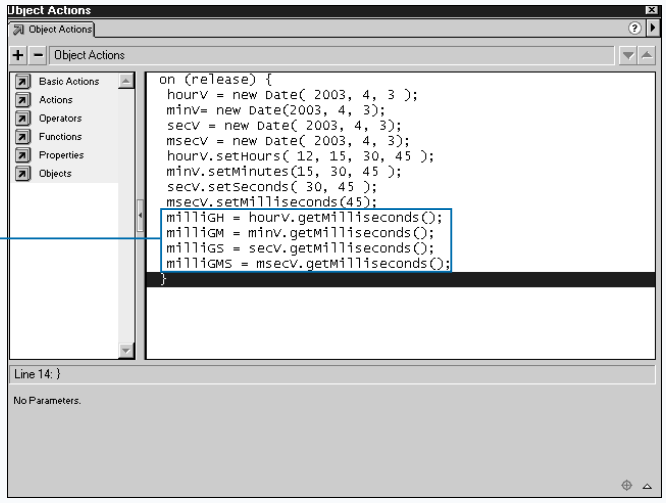
■ If you associate the variables with dynamic text boxes, you can display the values the variables return to the user.

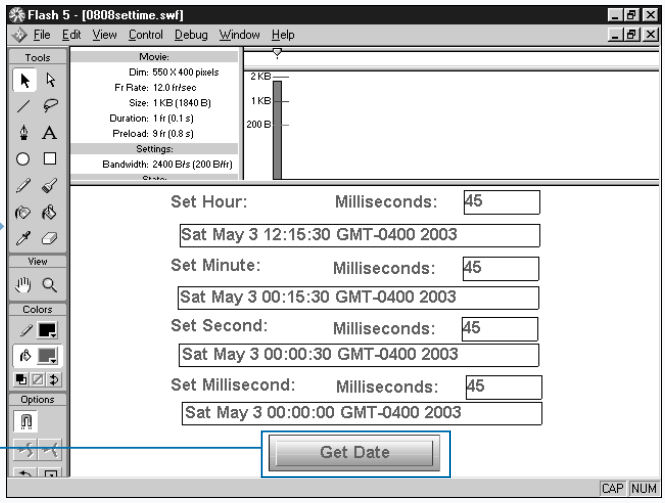
Extra

Use the `Date.UTC` method to retrieve the number of milliseconds between January 1, 1970, universal time and the date and time specified in the `Date.UTC` arguments. The syntax for the `Date.UTC` method is `Date.UTC(year,month,date,hour, minute,second,millisecond);`.

Use the `year` argument to specify a four-digit year. Use the `month` argument to indicate the month. Use 0 to represent January, 1 to represent February, and so forth. The `month` argument is optional. Use the `date` argument to specify the day of the month. Use an integer from 1 to 31. The `date` argument is optional. Use the `hour` argument to specify the hour. Use an integer from 0 to 23. Use 0 to represent midnight; use 23 to represent 11 p.m. The `hour` argument is optional. Use the `minute` argument to specify the minute. Use an integer between 0 and 59. The `minute` argument is optional. Use the second argument to specify the second. Again, use an integer between 0 and 59. The `second` argument is also optional.

When displaying dates in text boxes, make the box long enough to display the entire date. If Flash cannot display a date element in its entirety, it truncates.





5 Get the milliseconds.

6 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

7 Click the button to test your movie.

■ The screen displays the changes you made to the Date objects.

USING MATHEMATICAL FUNCTIONS

The Math object has methods that enable you to perform trigonometric functions. The functions supported include sine, cosine, tangent, arc sine, arc cosine, and arc tangent. The methods for these functions are `sin()`, `cos()`, `tangent()`, `asin()`, `acos()`, and `atan()`, respectively. The Math object also includes the method `atan2`. The method `atan2` computes the angle from the x-axis to the point. The `cos`, `sin`, and `tan` methods take the radian of an angle as an argument. Radians are used to measure an angle; 360 degrees are equal to 2 Pi radians. You can pass the functions of a measurement in radians, or you can you use the formula shown here to calculate radians: radian = Math.PI/180 * number of degrees.

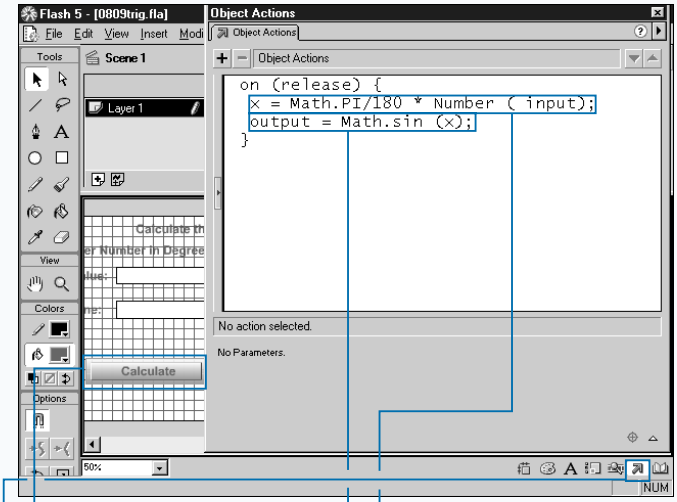
The Math object also includes functions for working with logarithms and exponentials based on Euler's constant. The `log()` method returns the natural logarithm of a number. The `exp` method returns Euler's constant raised to the power of the number specified.

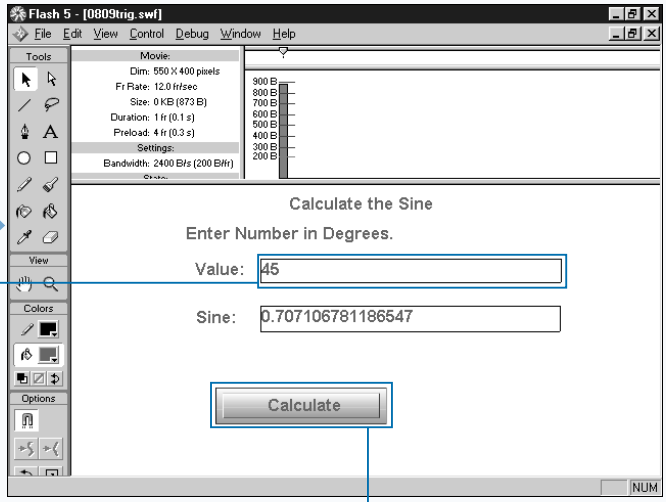
Extra

A constant is a value that does not change. The Math object includes several properties. All of the Math object properties are constants. You can use these constants to perform mathematical calculations.

PROPERTY	VALUE
E —Euler's constant	2.718
LN2 —The natural logarithm of 2	.0693
LOG2E —The base 2 logarithm of e	1.442
LN10 —The natural logarithm of 10	2.302
LOG10E —The base 10 logarithm of e	.434
PI —The ratio of the circumference of a circle to its diameter	3.14159
SQRT1_2 — The reciprocal of the square root of 1/2	.707
SQRT2 —The square root of 2	1.414

USING MATHEMATICAL FUNCTIONS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file trig.fla, which you can find on the CD that accompanies this book.

2 Click to open the Actions panel.

3 Convert degrees to radians.

4 Calculate the sine.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

6 Type a number.

7 Click the button to test your movie.

■ The screen displays the results of the calculation.

RAISE A POWER OR FIND A SQUARE ROOT

When performing mathematical calculations, you may need to raise a number to a power using the `Math.pow` method. The syntax for the `Math.pow` method is `Math.pow(x, y);`.

Use the `x` argument to specify the number you want to raise to a power. Use the `y` argument to specify the power to which you want to raise the number. The statement `Math.pow(3,2);` raises 3 to the second power. The statement returns 9.

If you want to find the square root of a number, use the `Math.sqrt` method. The syntax for the `Math.sqrt` method is `Math.sqrt(x);`.

Use the `x` argument to specify the value for which you want to find the square root. The value must be greater than or equal to zero. The statement `Math.sqrt(9);` finds the square root of 9. The statement returns 3.

Apply It

You can raise a number to a power using the script that follows. The variables `inputV` and `inputpowV` are input text boxes in which the user types the number and the power to which to raise the number. The variable `result` is a dynamic text box. It displays the results of the calculation on the screen. You attach this script to a button.

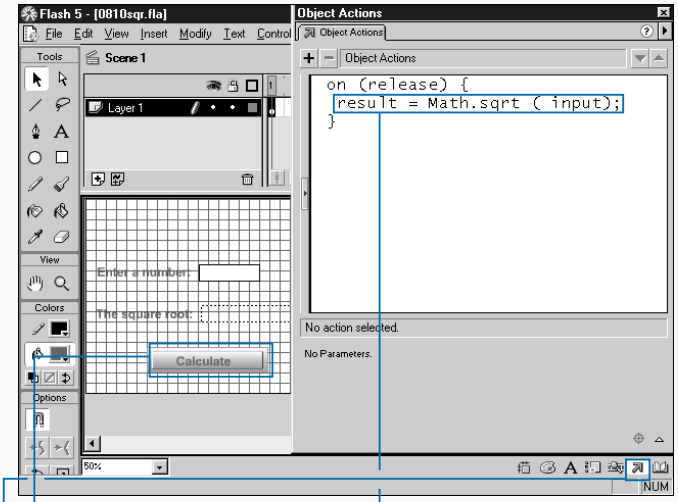
TYPE THIS:

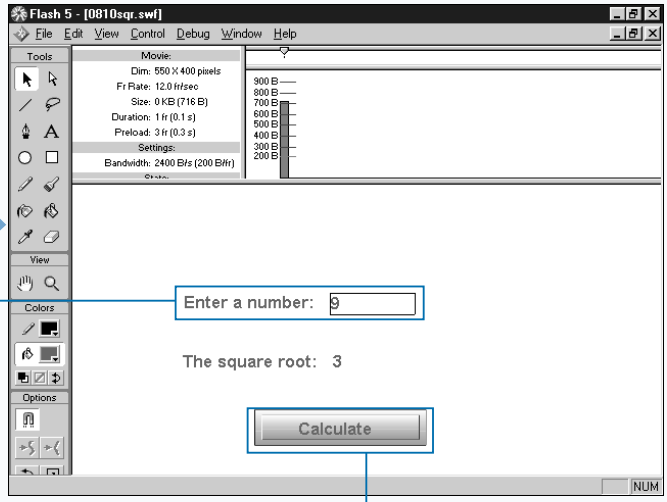
```
on (release) {
    result = Math.pow ( inputV, inputpowV);
}
```

RESULT:

ActionScript raises the number you entered to the power you specified.

WORK WITH POWERS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file sqr.fla, which you can find on the CD that accompanies this book.

2 Click to open the Actions panel.

3 Find the square root.

■ Associating the variable results with a dynamic text box enables you to display the value the variable returns to the user.

4 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

5 Type a value.

6 Click the button to test your movie.

■ The screen displays the square root.

ROUND NUMBERS

You can use the `Math.round` method to round a number to an integer. The syntax for the `Math.round` method is `Math.round(x);` . Use the argument `x` to specify the number you want to round. If the value after the decimal point is 0.5 or higher, the round method rounds up. If the value after the decimal point is lower than 0.5, the round method rounds down.

If you want to round up regardless of the value after the decimal point, use the `Math.ceil` method. The syntax for the `Math.ceil` method is `Math.ceil(x);` . Use the `x` argument to specify the number you want to round. The statement `ceilNumber = math.ceil(123.01);` returns 124.

If you want to round down regardless of the value after the decimal point, use the `Math.floor` method. The syntax for the `Math.floor` method is `Math.floor(x);` . Use the `x` argument to specify the number you want to round. The statement `floorNumber = math.floor(123.99);` returns 123.

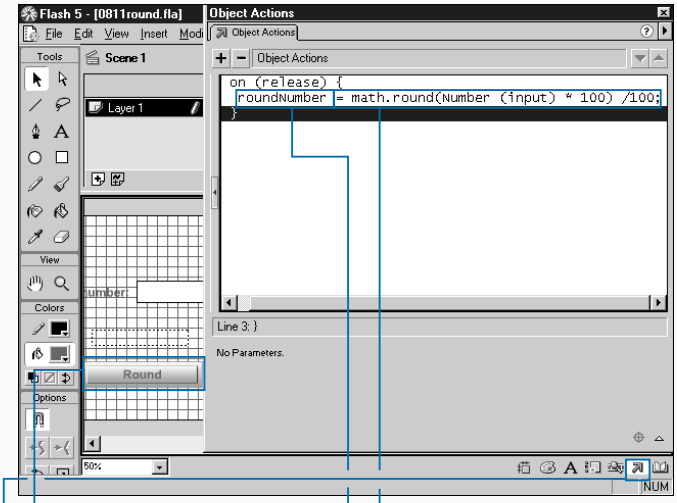
The `Math.round`, `Math.ceil`, and `Math.floor` methods all round your number to the nearest integer. You might want to round your number to one, two, three, or more decimal places. To round to one decimal place, multiply the number you want to round by 10, round the number, and then divide the result by 10. If you want two decimal places, multiply by 100 and divide by 100. If you want three decimal places, multiply by 1000 and divide by 1000.

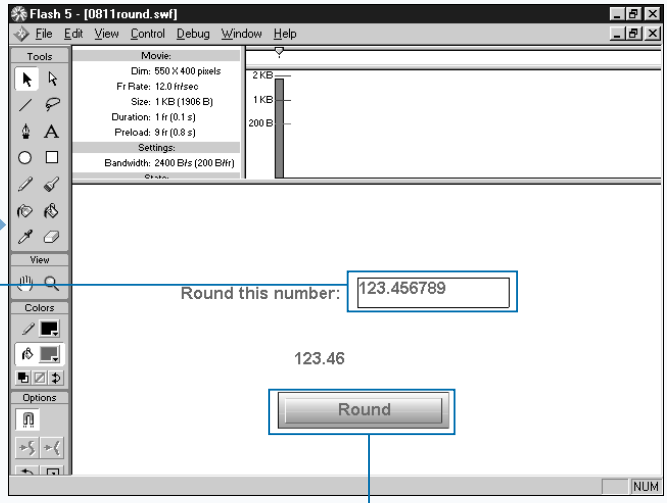
Extra

Math round returns the following values:

TYPE THIS:	RESULT:
`roundNumber = math.round(123.50);`	124
`roundNumber = math.round(123.49);`	123

ROUND NUMBERS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file round.fla, which you can find on the CD that accompanies this book.

2 Click to open the Actions panel.

3 Type the formula for rounding a number.

4 Assign the formula to a variable.

■ Associating the variable with a dynamic text box enables you to display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

6 Type a number with more than 2 digits after the decimal point.

7 Click the button to test your movie.

■ The screen displays the number rounded to two digits.

GENERATE RANDOM NUMBERS

You can use random numbers for a variety of purposes, such as displaying a movie clip at random locations on the Stage. You can use the `Math.random` method to generate random numbers. The syntax for the `Math.random` method is `Math.random();`. The `Math.random` method returns a random number between 0 and 1. If you want to generate a number between 0 and another value, multiply `Math.random` by the value you want to set as the limit. For example, if you want to generate a random number between 0 and 10, multiply `Math.random` by 10.

The number generated might not be a whole number. You can use the `Math.round` method to round the number to an integer. This example returns random integers between 1 and 10: `Math.round(Math.random() * 10);`.

You might want to generate random numbers between two numbers. This example generates integers between 50 and 60: `50 + Math.round(Math.random()*10);`.

Extra

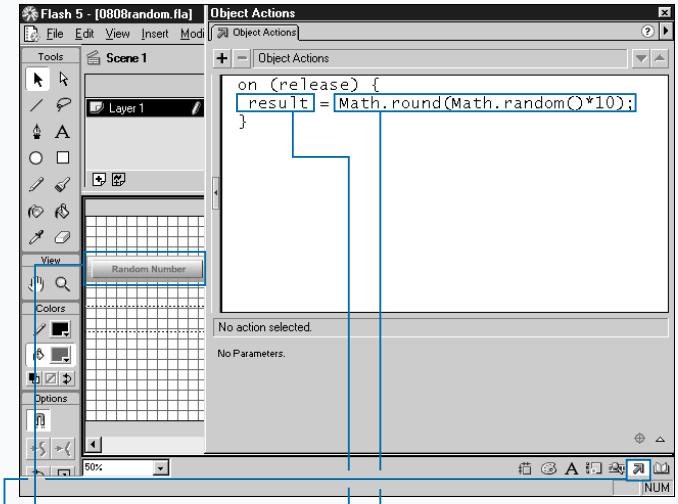
The random function provides you with an alternate way to generate a random number. The random function returns integers between 0 and a specified value. The syntax for the random function is `random(value);`.

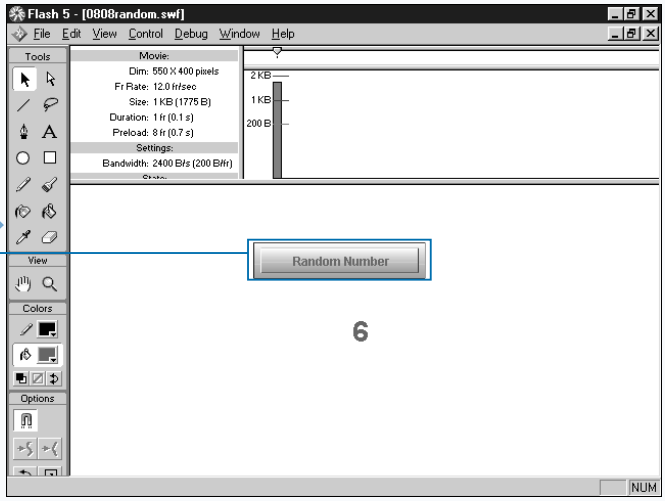
Use the `value` argument to set the highest value the random function should return. The random function will return any value between 0 and the value you specify minus 1.

TYPE THIS:	RESULT:
`Random(5);`	0, 1, 2, 3, or 4

Flash 5 deprecated the random function. If you are authoring for a Flash 5 environment, use `Math.random` instead.

GENERATE RANDOM NUMBERS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file random.fla, which you can find on the CD that accompanies this book.

2 Click to open the Actions panel.

3 Type the formula for generating a random number between 1 and 10.

4 Assign the result to a variable.

5 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

6 Click the button to test your movie.

■ The screen displays a random number between 1 and 10 each time you click the button.

FIND NUMERIC VALUES

The `Math.max` method enables you to compare two numbers to determine which is larger. The `Math.min` method enables you to compare two numbers to determine which is smaller. When an expression returns a number and you need to know the larger or the smaller of the two, use `Math.max` or `Math.min`.

The syntax for the `Math.max` method is `Math.max(x, y);`. The syntax for the `Math.min` method is `Math.min(x, y);`.

Use the `x` and `y` arguments to specify the values or expressions you want to compare. For example, `Math.max(23, 4);` compares 23 and 4 and returns 23; `Math.min(23, 4);` compares 23 and 4 and returns 4. If you send two equal values to `Math.max` and `Math.min`, the method returns the value sent. You cannot use `Math.max` and `Math.min` to compare non-numeric characters. If you send a non-numeric value to `Math.max` or `Math.min`, the method will return NaN, which means not a number.

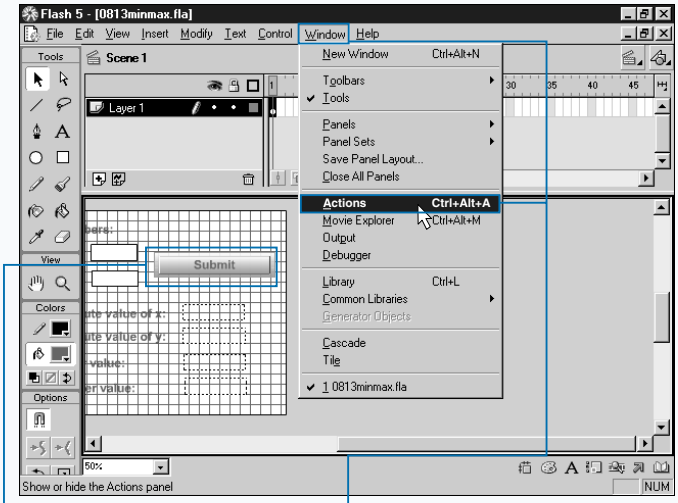
If you need to know the positive value of a number regardless of the sign of the value, use the `Math.abs` method. If you send a negative number to the `Math.abs` method, `Math.abs` returns a positve number. The `Math.abs` method always returns a positive number regardless of the value sent to it. The syntax for the `Math.abs` method is

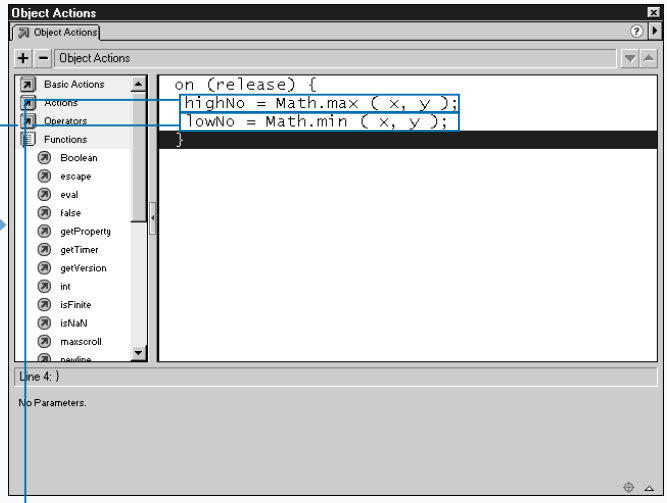
`Math.abs(x);`.

Use the `x` argument to specify the number or expression whose absolute value you want to find. The statement `Math.abs(-15)` returns 15.

You can use the `Math.abs` method with integers or floating-point numbers. If you assign a non-numeric value to `Math.abs`, `Math.abs` will return NaN.

FIND NUMERIC VALUES





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file maxmin.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Determine the maximum value.

4 Determine the minimum value.

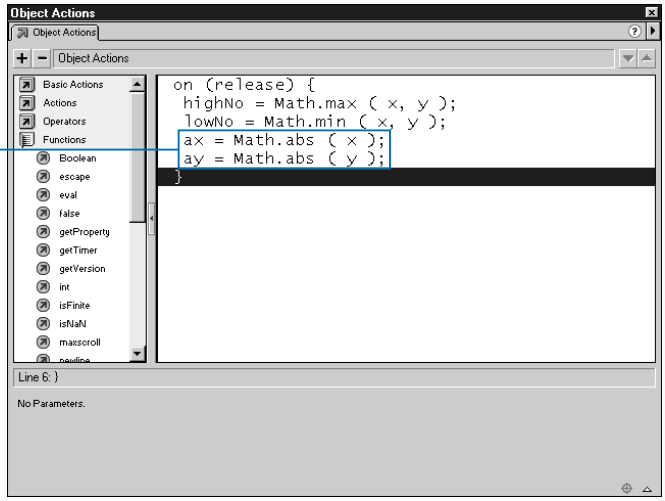
- Associating the variables `highNo` and `lowNo` with dynamic text boxes enables you to display the value the variable returns to the user.
- Associating the variables `x` and `y` with input text boxes allows you to store the values the user enters.

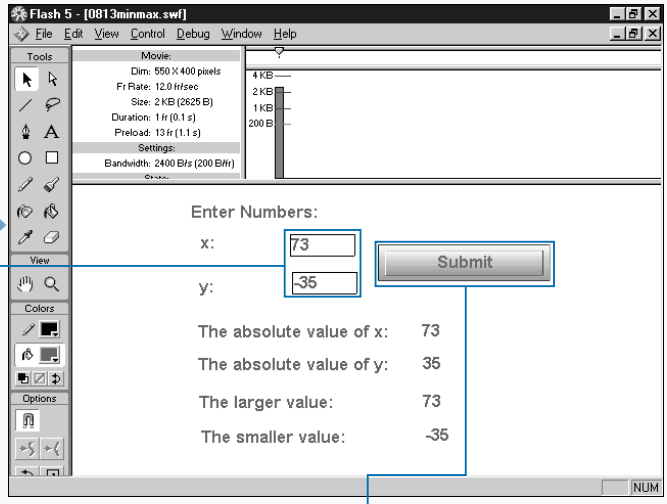
Extra

You can use this script to compare two values. The user enters numeric values in the input text boxes that are associated with variables x and y. The script compares the values. The variable `result` is a dynamic text box. It displays the results of the comparison. The script uses an `if` statement to determine if the numbers input are equal. If the numbers are equal, the script does not compare numbers. Instead, the script returns a message to the user informing the user that the numbers are equal.

Example:

```
on (release) {
    if (x != y) {
        highNo = Math.max ( x, y );
        lowNo = Math.min ( x, y );
        result = highNo + " > " + lowNo;
    } else {
        result = "x = y";
    }
}
```





5 Determine the absolute values.

■ Associating the variables `ax` and `ay` with dynamic text boxes allows you to display the value the variable returns to the user.

6 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

7 Type numbers in the text boxes.

8 Click the button to test your movie.

■ The minimum, maximum, and absolute values display.

USING THE KEY OBJECT

The methods of the Key object enable you to detect the last key pressed. You can use the Key object to create keyboard controls for Flash movies. The methods of the Key object are essential if you are programming games. Use the Key object to give the user the ability to maneuver objects around the screen using the keyboard. Use the Key object to give the user the ability to change the size or shape of objects using the keyboard. Alternatively, use the Key object to respond to the user based on the key pressed. You do not use a constructor to access the Key object.

ActionScript assigns a virtual key code to every physical key on the keyboard. Chapter 14 provides a list of virtual key codes. Virtual key codes are the same across all platforms and languages. You use virtual key codes to ensure that your movie key controls are the same across all platforms and languages.

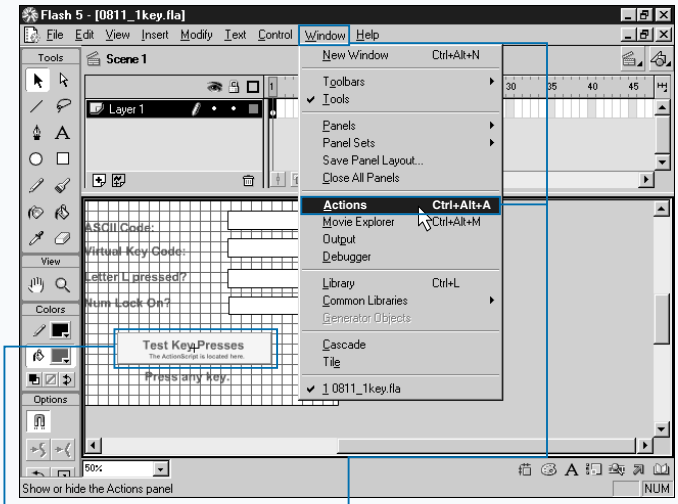
The `getCode` method returns the virtual key code of the last key pressed. The syntax for the `getCode` method is

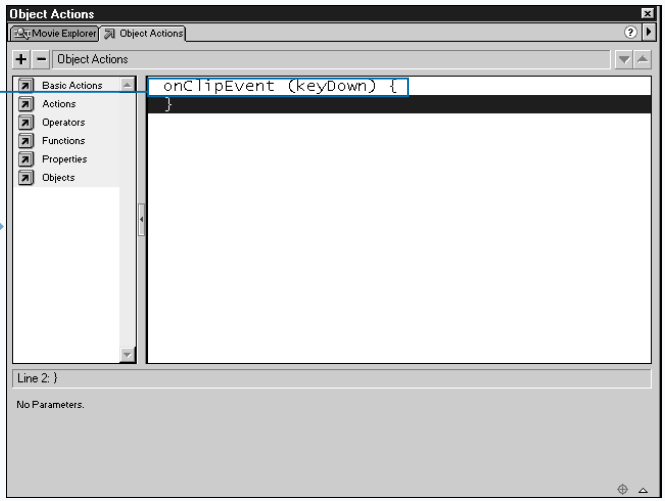
```
Key.getCode(); .
```

The `getCode` method does not take any arguments. The virtual key code for k is 75. If the user presses k, the `getCode` method returns 75. Using the `getCode` method, you can have your movie perform actions based on the key pressed.

The first 127 characters of every character set have ASCII values. The `getAscii` method returns the ASCII value for the last key pressed. The ASCII value for k is 107. If the user presses k, `getAscii` returns 107.

ASSIGN A KEY CODE





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file 1key.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Set the event to `KeyDown`.

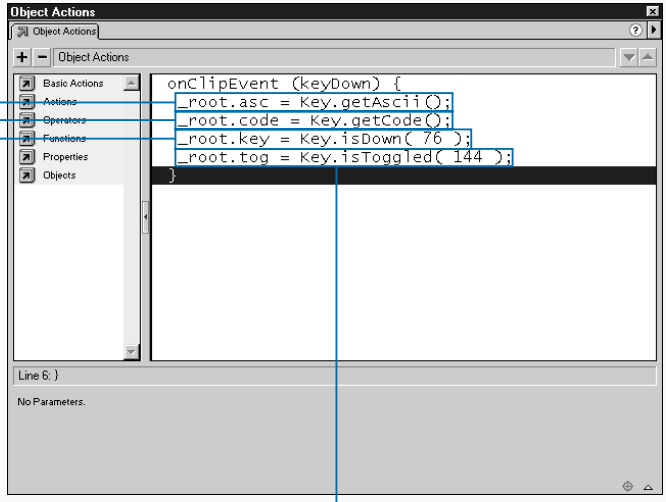
■ This causes the script to execute every time the user presses a key.

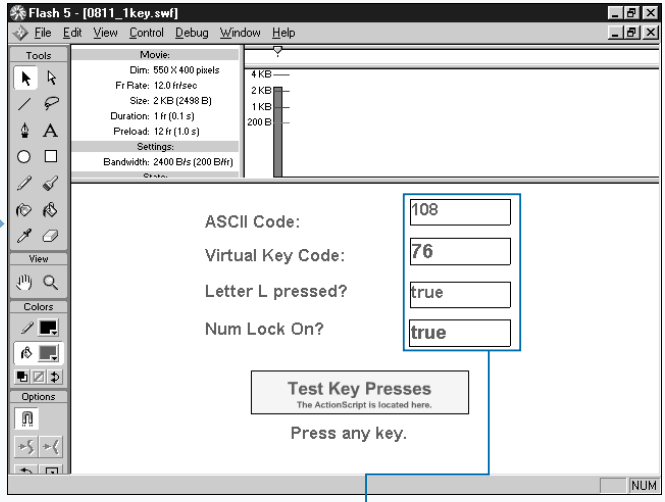
Extra

You can use arrow keys to move a movie clip around the Stage. This script uses the arrow keys to move a smiley face.

Example:

```
onClipEvent (keyDown) {
 if (Key.isDown(Key.RIGHT)) {
   _x = _x + 10;
 }
}
onClipEvent (keyDown) {
 if (Key.isDown(Key.LEFT)) {
   _x = _x - 10;
 }
}
onClipEvent (keyDown) {
 if (Key.isDown(Key.UP)) {
   _y = _y - 10;
 }
}
onClipEvent (keyDown) {
 if (Key.isDown(Key.Down)) {
   _y = _y + 10;
 }
}
```





4 Get the ASCII code.

5 Get the virtual key code.

6 Evaluate the key pressed.

■ The value 76 is the virtual key code for the letter L.

■ Associating the variables with dynamic text boxes enables you to display the values the variables return to the user.

7 Evaluate to see if the Num Lock key is on.

8 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

9 Test your movie by pressing keys on the keyboard.

■ The screen displays the ASCII code, the virtual key code, whether you pressed L, and the status of the Num Lock.

CONTINUED

USING THE KEY OBJECT (CONTINUED)

You may at times want to use ASCII values instead of virtual key codes because capital and small letters have different ASCII values. For example, the letter *A* and the letter *a* do not have the same ASCII value. The syntax for the `getAscii` method is

```
Key.getAscii();.
```

The `getAscii` method does not take any arguments.

You use the `Key.isDown` method to determine if the user pressed a specified key. The `Key.isDown` method returns `true` if the user pressed the key and `false` if the user did not press the key. The syntax for the `Key.isDown` method is

```
Key.isDown(keycode);.
```

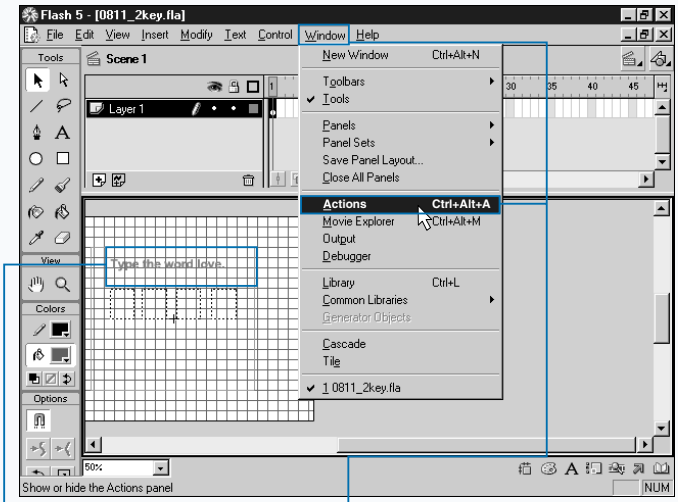
Use the `keycode` argument to specify the virtual key code value assigned to the value or the Key object property assigned to the key. The properties of the Key object represent the keys most commonly used to control games.

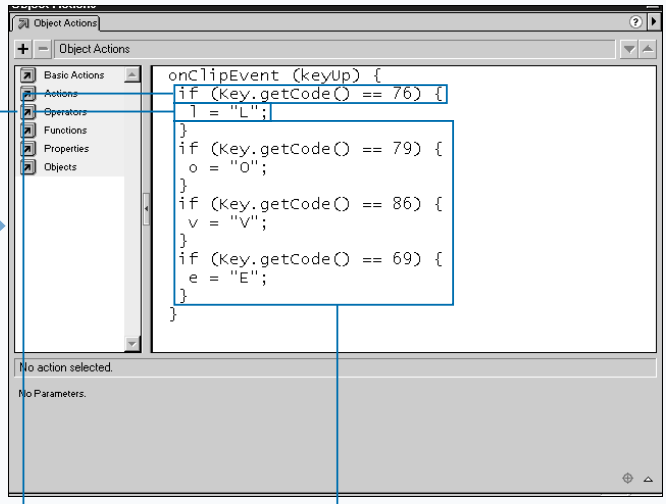
The virtual keycode for the Up arrow key is 38. The statement `Key.isDown(38);` returns true if the user presses the Up key. The Key object property for the Up key is `Key.UP`. The statement `Key.isDown(Key.UP);` returns true if the user presses the Up key.

Use the `Key.isToggled` method to determine whether the Caps Lock or Num Lock key is on or off. The syntax for the `Key.isToggled` method is `Key.isToggled(keycode)`. Use the keycode argument to specify the key you want to test. Use 20 for the Caps Lock key and 144 for the Num Lock key. The `Key.isToggle(keycode)` method returns `true` if the Caps Lock key or Num Lock key is on and `false` if the Caps Lock key or Num Lock key is off.

The `onClip Event (keyDown)` and the `onClip Event (keyUp)` events respond each time the user presses a key. When using the Key object, you may want to associate your script with one of these events.

USING THE KEY OBJECT (CONTINUED)





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file 2Key.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Use an `if` statement to set a condition.

■ If the virtual key code for the key pressed equals the key code specified, the condition is true.

4 Assign a value to a variable if the condition is true.

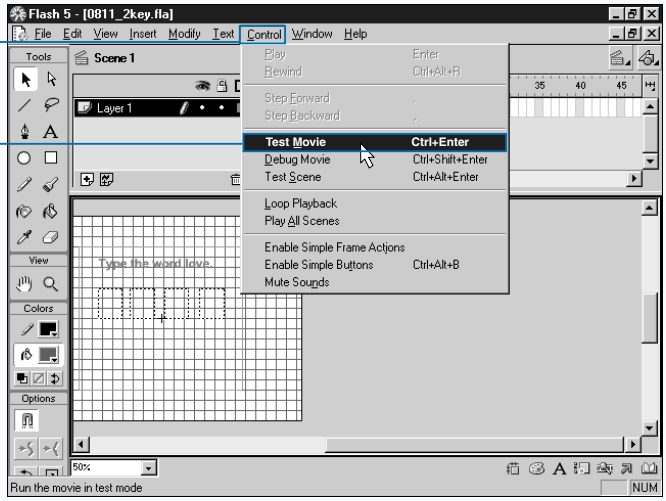
■ Associating the variable with a dynamic text box enables you to display the value the variable returns to the user.

5 Repeat steps 3 and 4 for each virtual key code you want to test.

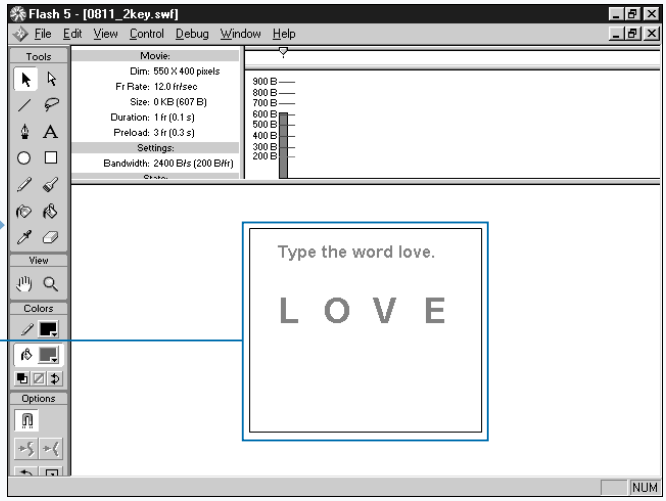
Extra

The properties of the Key object represent the keys most commonly used to control games. You can use properties of the `keycode` method as the `keycode` argument when using the `Key.isDown` method.

PROPERTY	REPRESENTS	VALUE	PROPERTY	REPRESENTS	VALUE
BACKSPACE	Backspace Key	9	INSERT	Insert Key	45
CAPSLOCK	Caps Lock Key	20	LEFT	Left Key	37
CONTROL	Control Key	17	PGDN	Page Down Key	34
DELETE KEY	Delete Key	46	PGUP	Page Up Key	33
DOWN	Down Arrow	40	RIGHT	Right Key	39
END	End Key	35	SHIFT	Shift Key	16
ENTER	Enter Key	13	SPACE	Space Key	32
ESCAPE	Esc Key	27	TAB	Tab Key	9
HOME	Home Key	36	UP	Up Key	38



6 Click Control ➪ Test Movie to move to the test environment.



7 Type a word to test your movie.

■ The word appears on screen.

USING THE SOUND OBJECT

You can use sound to create characters that talk, narrate a movie, or play background music. Sound methods enable you to turn sounds on and off, increase or decrease volume, or determine which speaker plays the sound. There are four steps to creating sound using ActionScript. You name the sound and set the sound to export, create a Sound object, attach the sound, and then start the sound.

You name a sound and set the sound to export using the Symbol Linkage Properties dialog box. You access the Symbol Linkage Properties dialog box through the Options menu of the Library. In the Symbol Linkage Properties dialog box, set the Linkage to Export this Symbol, and use the Identifier field to name the sound.

You use the constructor `new Sound` to create a Sound object. The syntax for the `new Sound` constructor is

```
soundName = new Sound (target);.
```

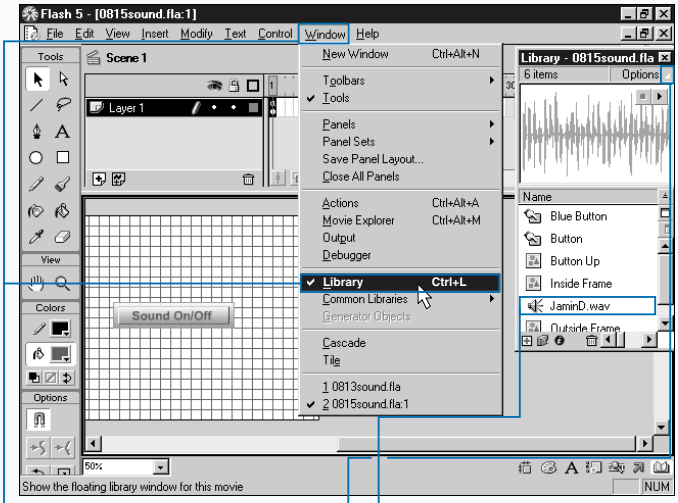
Use the `target` argument to specify the movie clip instance to which the Sound object applies. If you do not specify a target, the Sound object controls all of the sounds on the global Timeline. If you want to control each sound independently, place each sound in a separate movie clip. Use the `soundName` argument to specify the name of the Sound object.

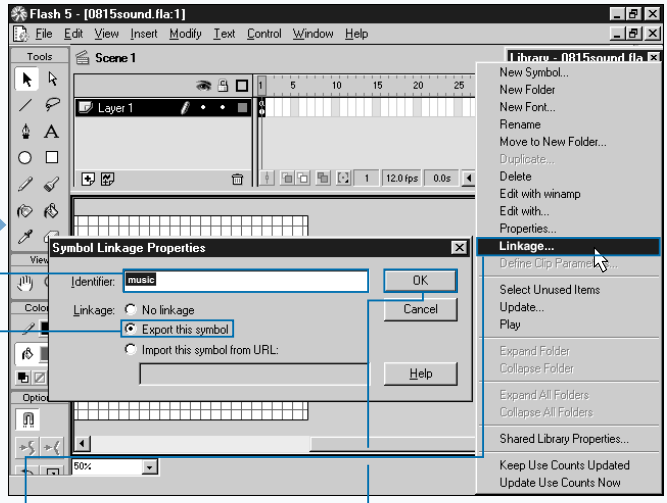
You use the `attachSound` method to create an instance of the sound. The syntax for the `attachSound` method is

```
soundName.attachSound("idName");.
```

Use the `soundName` argument to specify the name of the Sound object. Use the `idName` argument to specify the name of the new instance of the sound. Enclose the idName in quotes. The name you specify should be the same name you gave to the sound in the Identifier field of the Symbol Linkage Properties dialog box.

USING THE SOUND OBJECT





SET THE SYMBOL LINKAGE PROPERTY

1 Click Window ➪ Library to open the Library.

Note: This example uses file sound.fla, which you can find on the CD that accompanies this book.

2 Select the sound symbol.

3 Click ◪ to open the Options menu.

4 Click Linkage to open the Symbol Linkage Properties dialog box.

5 Click Export this symbol for the Linkage type (○ changes to ◉).

6 Type a name in the Identifier field.

7 Click OK.

8 Save your .FLA file.

■ After setting the Symbol Linkage Property, you must save your file.

Extra

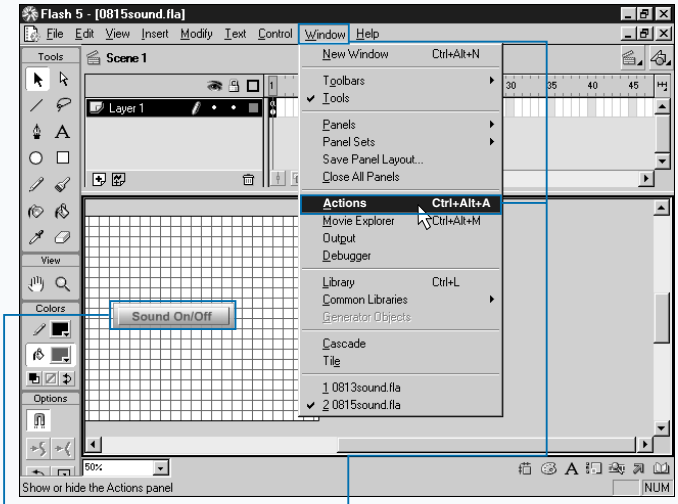
When the user presses a button, this script toggles the music on. When the user presses the button again, the script toggles the music off. The variable `onOff` is a dynamic text box that displays Click for Music when the music is off and Click to Stop when the music is on.

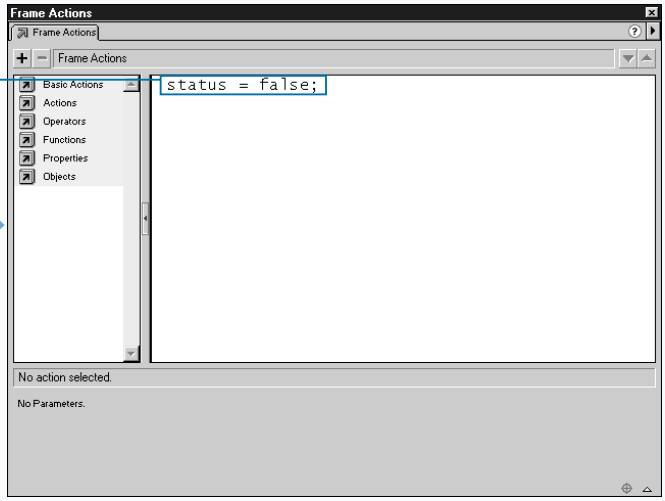
Frame 1

```
status = false;
```

Button

```
on (release) {
    if (status == false) {
        beat = new Sound();
        beat.attachSound("music");
        beat.start(0,10);
        status = true;
        onOff = "Click to Stop";
    } else {
        beat.stop();
        status = false;
        onOff = "Click for Music";
    }
}
```





ADD SOUND

1 Select the frame, button, or movie clip to which you want to add ActionScript.

2 Click Window ➪ Actions to open the Actions panel.

3 Set a variable to false.

■ You are initializing the variable. Frequently, you initialize variables in frame 1. Subsequent statements reset the value. In this case, false means no sound is playing.

CONTINUED

USING THE SOUND OBJECT (CONTINUED)

You use the `start` method to play your sound. The syntax for the `Sound.start` method is

```
soundName.start(secondOffset,loop);.
```

Use the `soundName` argument to specify the name of the Sound object. Use the `secondOffset` argument to specify the point at which you want the sound to start. If you have a 20-second sound and you specify a `secondOffset` of 10, the sound will start playing in the middle of the sound. Note that this argument does not delay the start of the sound 10 seconds, but instead starts the sound at the 10-second mark. The `secondOffset` argument is optional. Use the `loop` argument to specify the number of times the sound should loop. The `loop` argument is also optional.

Once you have started your sound, you use the `stop` argument to stop it. The syntax for the `stop` argument is

```
soundName.stop("idName");.
```

Use the `idName` argument to specify the name of the sound you want to stop. Enclose the `idName` in quotes. The `idName` argument is optional. If you do not specify an `idName`, all sounds will stop.

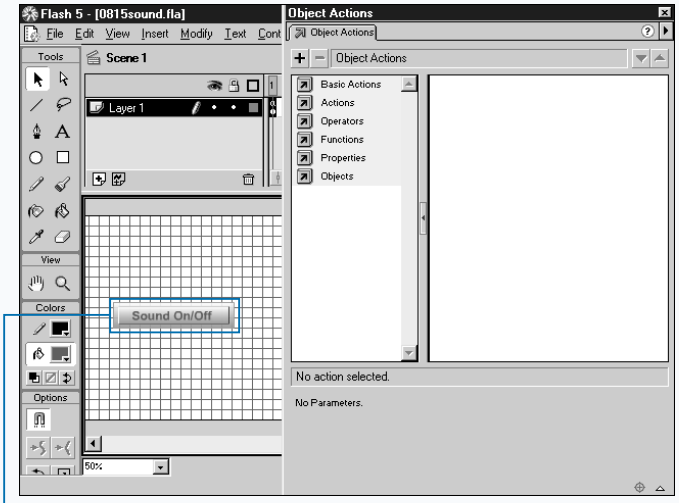
You can also use the `stopAllSounds` action to stop the sounds. Sounds set to streaming will resume playing when the playhead moves over them. The syntax for the `stopAllSounds` action is

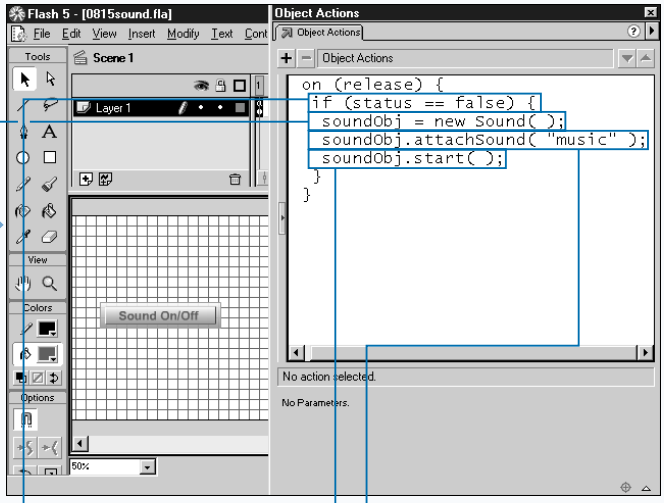
```
stopAllSound();.
```

The `stopAllSounds` action does not take any arguments.

In Flash, you can play multiple sounds at the same time. You can play background music, while a narrator narrates your movie. You can start sound in a frame or using any of the button or movie clip handlers. That means you can start sound with the press of a button, when the user rolls over an object, or upon entry into a frame.

USING THE SOUND OBJECT (CONTINUED)





4 Select the frame, button, or movie clip to which you want to add ActionScript.

5 Use an `if` statement to test the variable you set in frame 1.

■ If the variable is equal to false, the statements execute.

6 Create a Sound object.

7 Attach the sound.

■ You use the instance name you gave the sound in the Symbol Linkage Properties dialog box.

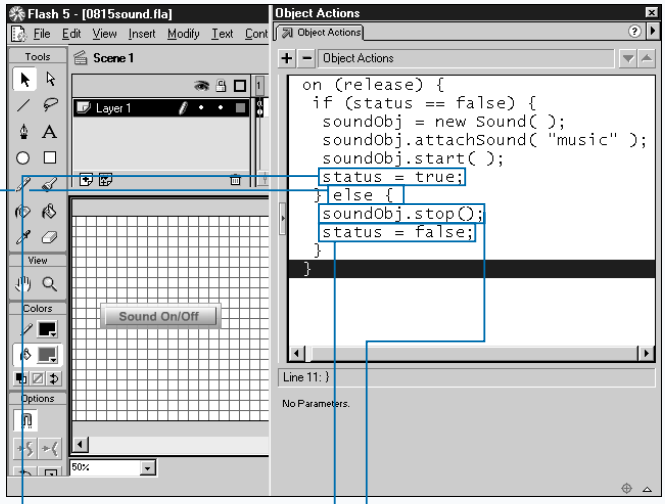
8 Start the sound.

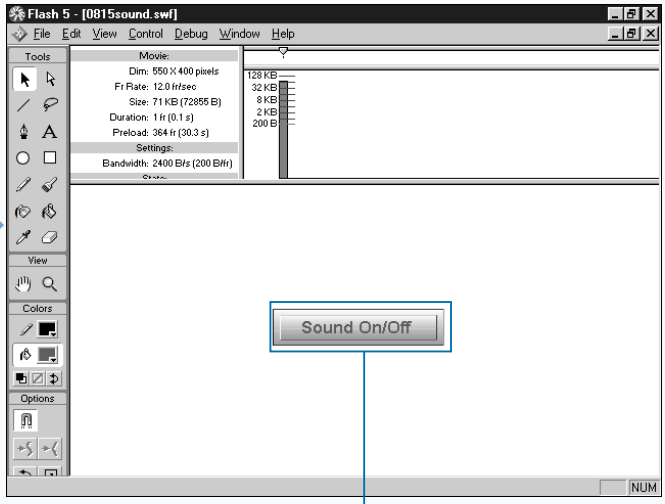
Extra

You can use a script similar to this one to increase the volume of a sound. In this script, if the volume is less than 95, the script increases the volume by 5 each time the user releases the mouse after clicking a button. If the volume is greater than or equal to 95, the script sets the volume to 100. You can create a separate button to decrease the volume in much the same way.

Button

```
on (release) {
    vol = beat.getVolume();
    if (vol < 95) {
        beat.setVolume(vol + 5);
        vol = beat.getVolume();
    } else {
        beat.setVolume(100);
        vol = beat.getVolume();
    }
```





9 Reset the value of the variable.

■ The sound is now playing so you set the status to true.

10 Create an `else` statement.

■ If the status is not equal to false, these statements will execute.

11 Stop the sound.

12 Reset the value of the variable.

■ The sound is no longer executing so you set the status to false.

13 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

14 Click the button to test your movie.

■ Clicking the button toggles sound on and off.

SET VOLUME AND PANNING

ActionScript provides methods that enable you to adjust sound volume and panning or to give your users the ability to adjust the volume and panning. You use the `setVolume` method to adjust sound volume. The syntax for the `setVolume` method is

```
soundName.setVolume(volume);.
```

Use the `soundName` argument to specify the name of the Sound object. Use the `volume` argument to specify an integer between 0 and 100 that represents the volume level. A value of 100 represents full volume. A value of 0 represents no volume. The default is 100.

To obtain the value of the current volume level, use the `getVolume` method. The syntax for the `getvolume` method is

```
soundName.getVolume();.
```

The `getVolume` method returns a value between 1 and 100 that represents the current volume level.

You use the `setPan` method to control how sound is played in each speaker. The syntax for the `setPan` method is

```
soundName.setPan(pan);.
```

Use the `soundName` argument to specify the name of the Sound object. Use the `pan` argument to specify an integer between –100 and 100. A pan of –100 only uses the left speaker. A pan of 100 only uses the right speaker. A value of 0 balances the sound equally between the two speakers. You can use `setPan` to fade sound from one speaker to the other.

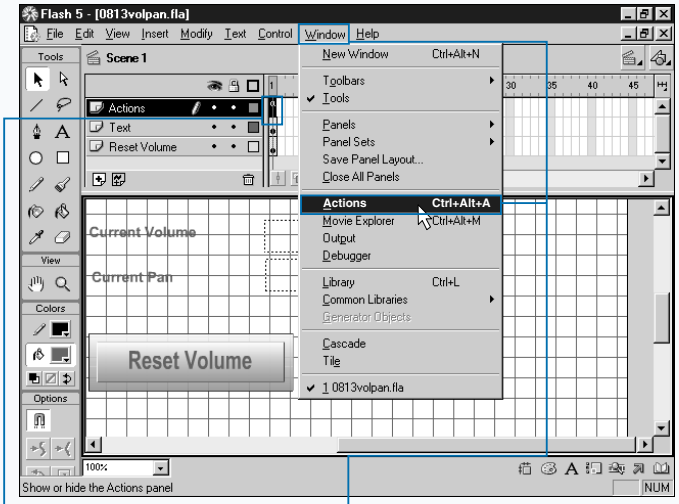
Use the `getPan` method to obtain the current pan value. The syntax for the `getPan` method is

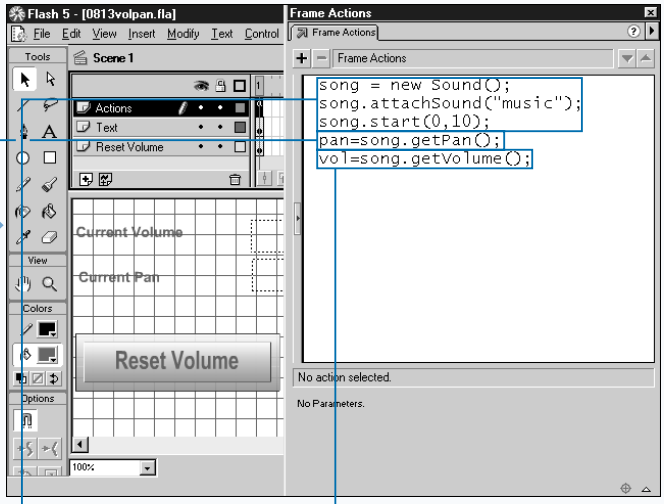
```
soundName.getPan().
```

The `getPan` method returns a value between –100 and 100, which represents the current pan level.

ActionScript includes a `_soundbuftime` action. Use the `_soundbuftime` action to establish the number of seconds of streaming sound to prebuffer. The default is 5 seconds. The syntax for the `_soundbuttime` action is `_soundbuftime = integer;`. Use the `integer` argument to specify the number of seconds to buffer.

SET VOLUME AND PANNING





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file volppan.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Create and start a sound.

4 Get the pan value.

■ Associating the variable with a dynamic text box, enables you to display the value the variable returns to the user.

5 Get the volume value.

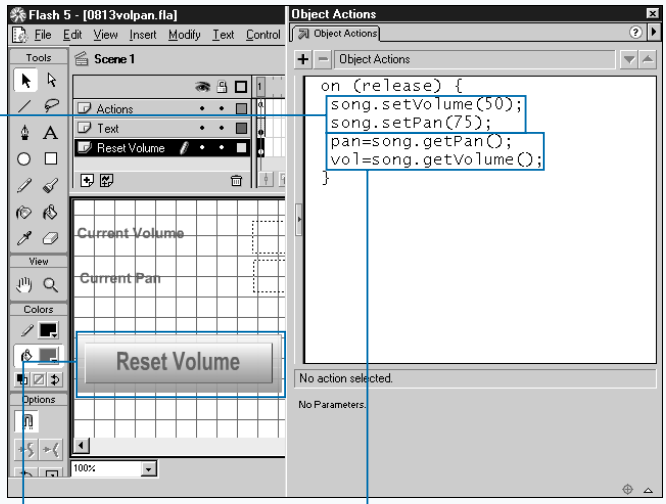
■ Associating the variable with a dynamic text box enables you to display the value the variable returns to the user.

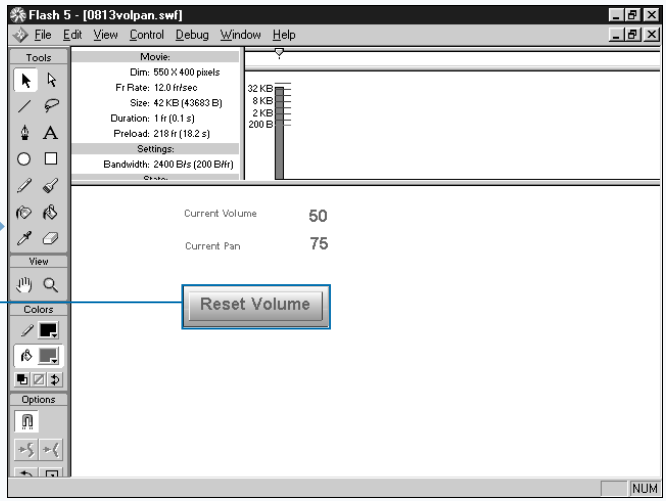
Extra

You can use a script similar to this one to set the pan. In this script, if the pan is less than 95, the script increases the pan by 5 each time the user releases the mouse after clicking on a button. If the pan is greater than or equal to 95, the script sets the pan to 100. You can create a separate button to decrease the pan in much the same way.

Button

```
on (release) {
    pan = beat.getpan();
    if (pan < 95) {
        beat.setPan(pan + 5);
        pan = beat.getPan();
    } else {
        beat.setPan(100);
        pan = beat.getPan();
    }
}
```





6 Select the frame, button, or movie clip to which you want to add ActionScript.

7 Reset the volume and pan values.

8 Get the new pan and volume values.

9 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

10 Click the button to test your movie.

■ Clicking the button resets the pan and volume values.

USING THE SELECTION OBJECT

If you make your dynamic text boxes or input text boxes selectable, the user can click and drag to select text located in the text boxes. You use the `selection` methods to obtain the beginning and ending points of user selections. This is useful when you want your movie to respond to or display information in response to a user selection. You use the `getBeginIndex` method and the `getEndIndex` to retrieve the index position of selected text. The syntax for `getBeginIndex` is

```
Selection.getBeginIndex();.
```

The syntax for `getEndIndex` is

```
Selection.getEndIndex();.
```

The `Selection.getBeginIndex` and the `Selection.getEndIndex` methods do not take arguments. The `Selection.getBeginIndex` method returns the index position of the beginning of the selection. The `selection.getEndIndex` method returns the index position of the end of the selection. The first position in a text box has an index position of 0, the second an index position of 1, and so forth. When the cursor is not located in a text box, the `Selection.getBeginIndex` and `Selection.getEndIndex` methods return –1.

When working with text boxes, if you need to know the location of the blinking cursor, use the `Selection.getCaretIndex` method. The syntax for the `Selection.getCaretIndex` method is

```
Selection.getCaretIndex();.
```

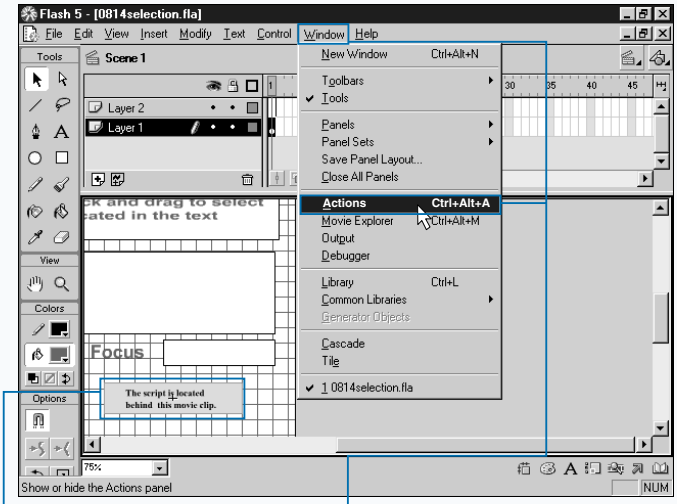
The `selection.getCaretIndex` method does not take any arguments. If the cursor is not located in a text box, `selection.getCaretIndex` returns a –1.

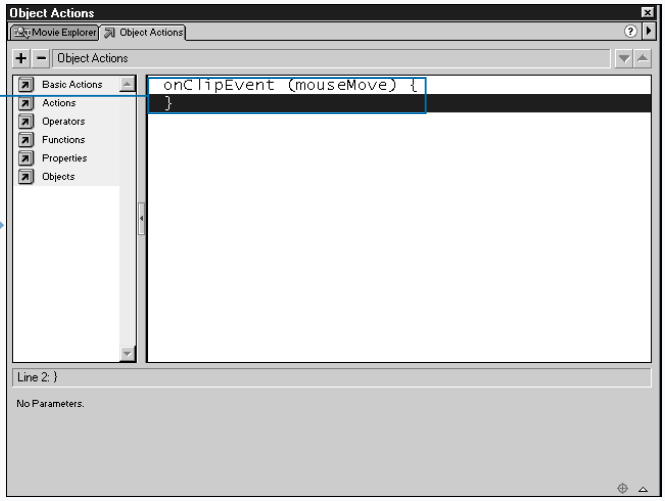
If you need the variable name of the field in which the cursor is currently located, use the `Selection.getFocus` method. The syntax for the `Selection.getFocus` method is

```
Selection.getFocus();.
```

The `Selection.getFocus` method does not take any arguments. It returns the name of the variable in which the cursor is currently located. If the cursor is not located in the text box, `selection.getFocus` returns null.

USING THE SELECTION OBJECT





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file selections.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Set the event to Mouse move.

■ This causes the script to execute every time the mouse moves.

Extra

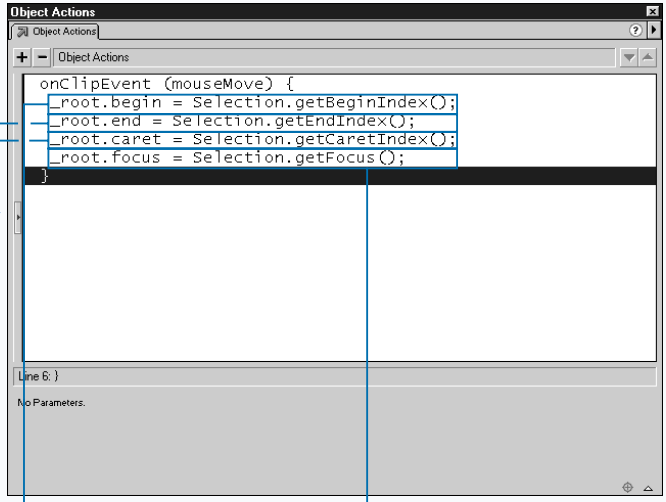
You can use the selection methods to determine what the user has selected. In this script, two words appear on the screen. Flash displays one word spelled correctly and one word spelled incorrectly. The user is instructed to select the word that is spelled incorrectly. If the user is successful, the script responds, correct. Otherwise, the script responds, wrong.

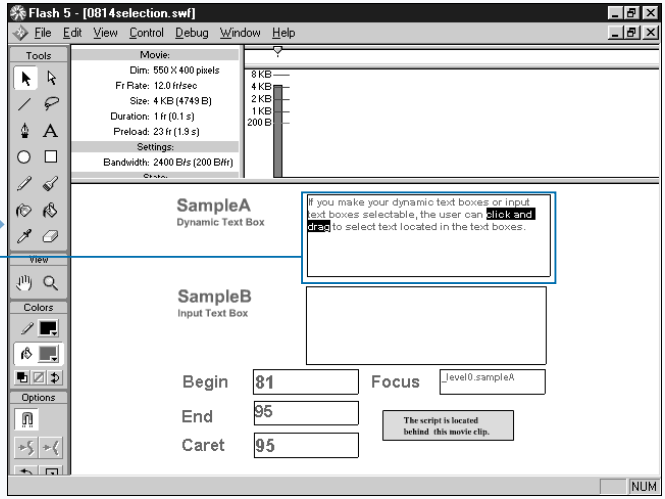
Frame 1

```
setProperty (codeMC, _visible, false);
```

MovieClip

```
onClipEvent (mouseMove) {
    _root.begin = Selection.getBeginIndex();
    _root.end = Selection.getEndIndex();
    if (_root.begin == 0 && _root.end == 9) {
        _root.response = "Correct!";
    } else if (_root.begin ==10 && _root.end ==19) {
        _root.response = "Wrong";
    } else {
        _root.response = "";
    }
}
```





4 Get the `BeginIndex` value.

5 Get the `EndIndex` value.

6 Get the `CaretIndex` value.

7 Get the `Focus` value.

8 Move to the test environment.

Note: See page 38 for instruction on how to test your movie.

9 Test your movie by clicking and dragging to select text.

■ The `BeginIndex`, `EndIndex`, `CaretIndex`, and `Focus` values display on-screen.

GET PROPERTIES

Movie clips have a number of properties, including height, width, location, and visibility. You can retrieve the properties of a movie clip using the `getProperty` function.

Retrieving movie clip properties enables you to perform actions based on the property value retrieved. For example, you can create a button that enables the user to toggle the size of an object between small, medium, and large. You retrieve the current `_height` and `_width` values and reset the values based on the value retrieved. You can retrieve the _x and _y properties of a movie clip to determine the location of the movie clip. The _x property gives you the x coordinate of a movie clip and the _y property gives the y coordinate of a movie clip. You can add 5 to the x coordinate to move the movie clip 5 pixels to the right. You can subtract 5 from the x coordinate to move a movie clip five pixels to the left.

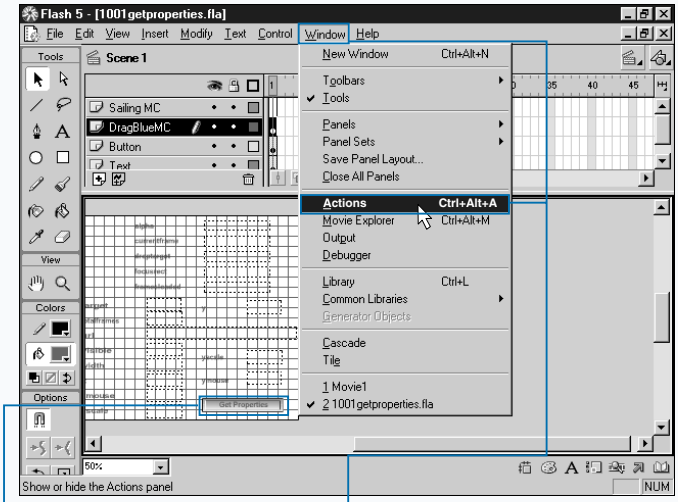
You can use the `getProperty` function to retrieve any movie clip property. The syntax for the `getProperty` function is `getProperty(instanceName, property);`.

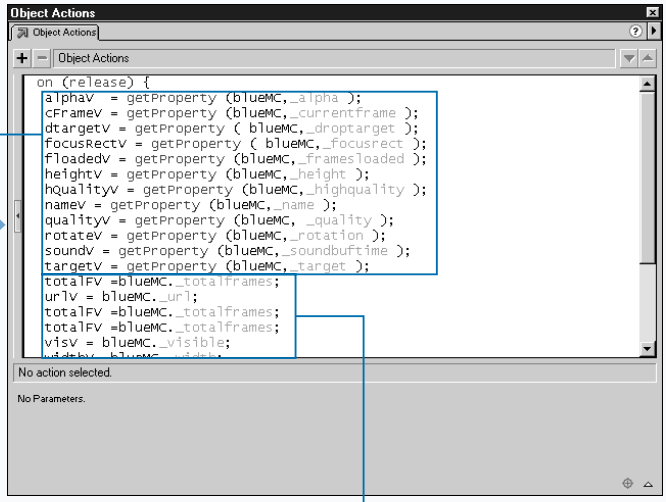
Use the `instanceName` argument to specify the instance for which you want to retrieve a property. Use the `property` argument to specify the property you want to retrieve. This example retrieves the height property of a movie clip named `sampleMC`: `getproperty(sampleMC, _height);`.

You can also retrieve movie clip properties by placing the instance name followed by a dot in front of the property. Here is the syntax: `instanceName._property;`.

Use the `instanceName` argument to specify the name of the instance for which you want to retrieve a property. Use the `property` argument to specify the property you want to retrieve. This example retrieves the height property of a movie clip named `sampleMC`: `sampleMC._height;`.

GET PROPERTIES





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file getproperties.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Get the movie clip property values using the `getProperty` function.

■ You can use the `getProperty` function to retrieve a property, or you can precede the property with the movie clip name followed by a dot.

Apply it

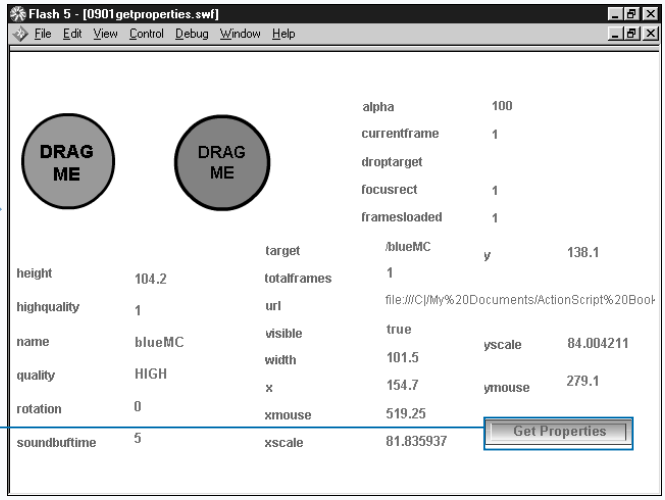
You can use the script that follows to toggle the size of a movie clip. The script is associated with a button.

TYPE THIS:

```
on (release) {
    if (getProperty (_root.blockMC, _height ) == 50  && getProperty (_root.blockMC, _width) == 50) {
        setProperty (_root.blockMC, _height, 100);
        setProperty (_root.blockMC, _width, 100);
    } else if (getProperty (_root.blockMC, _height ) == 100 && getProperty (_root.blockMC, _width) ==100) {
        setProperty (_root.blockMC, _width, 150);
        setProperty (_root.blockMC, _height, 150);
    } else if (getProperty (_root.blockMC, _height ) == 150  && getProperty (_root.blockMC, _width) ==150) {
        setProperty (_root.blockMC, _width, 50);
        setProperty (_root.blockMC, _height, 50);
    }
}
```

RESULT:

Each click of the button toggles the size of the movie clip blockMC from 50 pixels x 50 pixels, to 100 pixels x 100 pixels, to 150 pixels x 150 pixels, back to 50 pixels x 50 pixels.

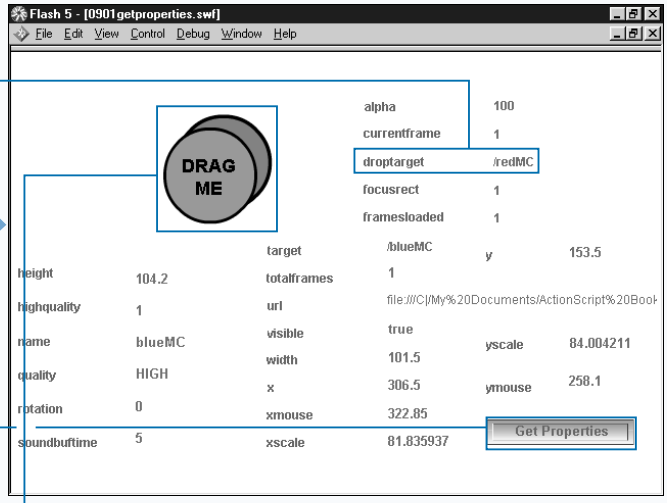


4 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

5 Click the button to test your movie.

■ Clicking the button displays the blueMC properties.



6 Drag the blueMC over the redMC.

7 Click the button to get the `dropTarget` property.

■ The screen displays the movie clip properties including `dropTarget`.

USING GET TIMER AND GET VERSION

You use the `getTimer` function to retrieve the number of milliseconds that have elapsed since a movie started playing. This function is useful when writing games. For example, you can write a game that gives the user 10 seconds to perform an action. You can retrieve the number of milliseconds that have elapsed when the user starts and you can stop the game 10 seconds later.

The syntax for the `getTimer` function is `getTimer();`. The `getTimer` function does not take any arguments.

You use the `getVersion` function to retrieve the version of Flash Player the user is using and the platform on which the user is working. The `getVersion` function only returns information if the user is using version 5 of Flash Player or higher. The `getVersion` function is useful when you want to display a different set of instructions to each user, dependent on their platform, or you want to ensure that the version of Flash Player the user is using is compatible with your movie.

The syntax for the `getVersion` function is `getVersion();`. The `getVersion` function does not take any arguments. The following is an example of what the `getVersion` function returns: WIN 5,0,17,0. The WIN indicates the user is using Windows. The 5,0,17,0 indicates the version of Flash Player the user is using.

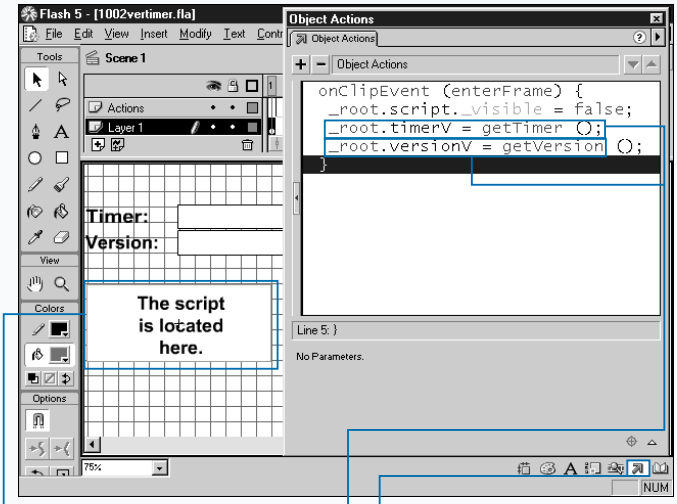
Extra

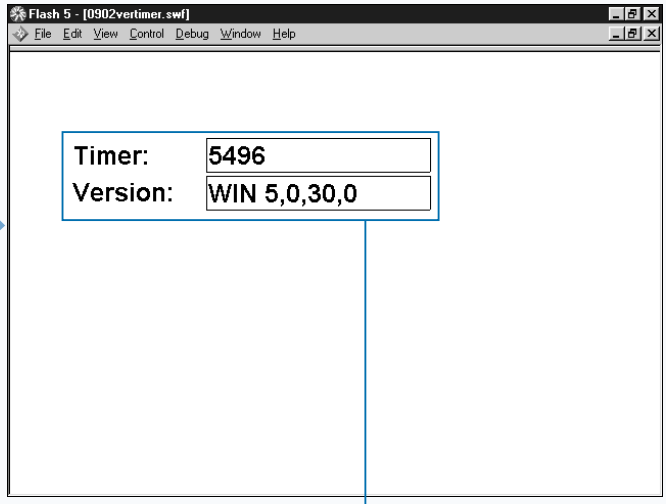
You can use `getTimer` to specify when a movie clip appears on the screen. The example that follows uses `getTimer` to display a movie clip on the Stage after 5 seconds have elapsed.

Example:

```
onClipEvent (enterFrame) {
    _root.timeV = getTimer ();
    if (_root.timeV >= 5000 ) {
        setProperty (_root.stopMC, _visible,
  true);
    }
}
```

USING GET TIMER AND GET VERSION





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file vertimer.fla, which you can find on the CD that accompanies this book.

2 Click to open the Actions panel.

3 Get the timer using the `getTimer` function.

4 Get the version using the `getVersion` function.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

■ The screen displays the timer and version.

USING EVAL

You use the ActionScript `eval` function to retrieve a variable, property, object, or movie clip from a string. This function is particularly useful when you have concatenated strings to form the variable, property, object, or movie clip name.

The syntax for the `eval` function is `eval(expression);`. Use the expression argument to specify the variable property, object, or movie clip you want to retrieve. If the expression argument represents a variable or property, `eval` returns a value. If the expression argument represents an object or movie clip, `eval` returns a reference to the object or movie clip. If ActionScript cannot find the expression the argument represents, `eval` returns `undefined`.

Flash 4 used the `eval` function to simulate arrays. If you are authoring for a Flash 5 environment, use the Array object to create arrays.

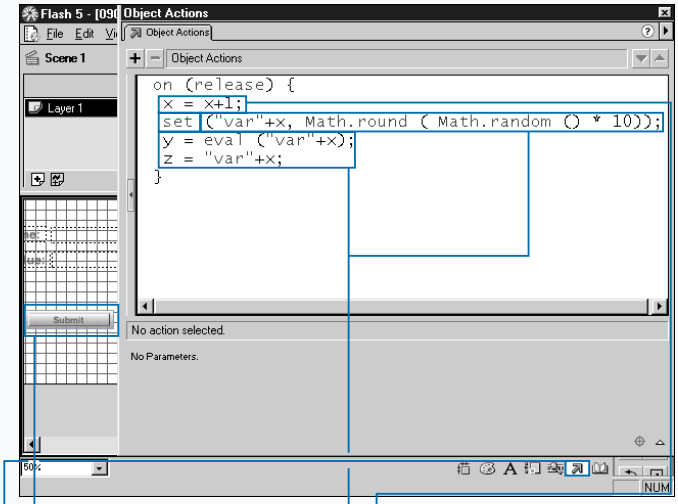
Extra

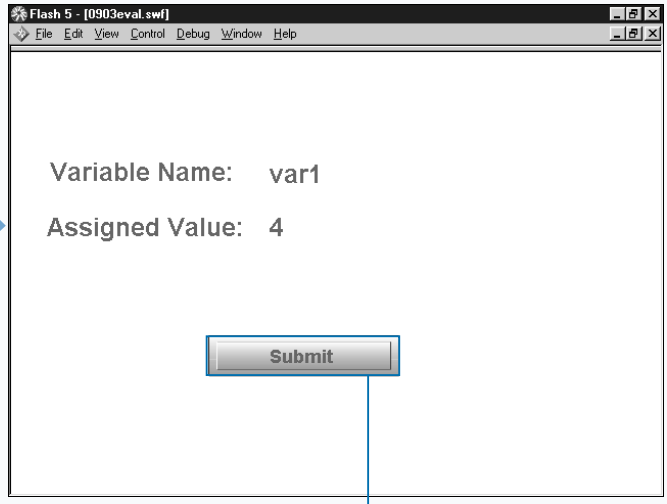
The table that follows illustrates the `eval` function.

EXAMPLE	RETURNS
`eval("sampleMC");`	`_level0.sampleMC`
`set ("x" + 1, 100);` `b = eval ( "x" + 1 );`	`100`
`eval("sampleMC._visible");`	`true`

`soundObj = new Sound( );`
`soundObj.attachSound( "music" );`
`soundObj.start( );`
`d = eval("soundObj");`
	`[object Object]`

USING EVAL





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file eval.fla, which you can find on the CD that accompanies this book.

2 Click to open the Actions panel.

3 Increment the value of x by one.

4 Assign a random number to a variable.

5 Retrieve the values.

6 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

7 Click the button to test your movie.

■ Clicking the button retrieves the value assigned to the variable and the variable name.

CONVERT A STRING TO A NUMBER

You cannot perform mathematical calculations using a string even if the value contained in the string is a number. If you need to perform a mathematical calculation using a value contained in a string, you must convert the string to a number.

You can use the `number` function to convert a string to a number. You can also use the `number` function to convert a Boolean to a number. The Boolean `true` returns 1 and the Boolean `false` returns 0. The syntax for the `number` function is `Number(expression);`.

Use the expression argument to specify the string or Boolean you want to convert. For more information on the `number` function, see Chapter 4.

You can also use the `parseFloat` function to convert a string to a number. The `parseFloat` function converts a string to a floating-point number. A floating-point number is a number that contains decimal places. If the string contains both numeric and non-numeric characters, the `parseFloat` function starts at the left and converts each character until it reaches the first non-numeric character. The `parseFloat` function ignores blank spaces preceding a string. If the first character in the string is not a number, `parseFloat` returns `NaN`, which stands for not a number.

The syntax for the `parseFloat` function is

`parseFloat(string);`.

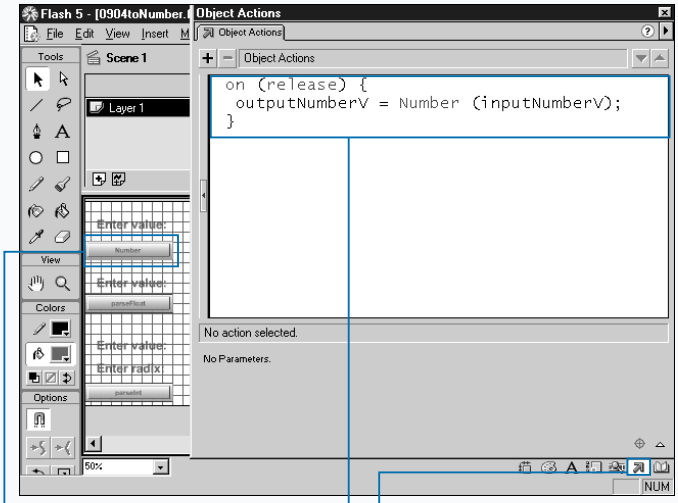
Use the `string` argument to specify the string you want to convert.

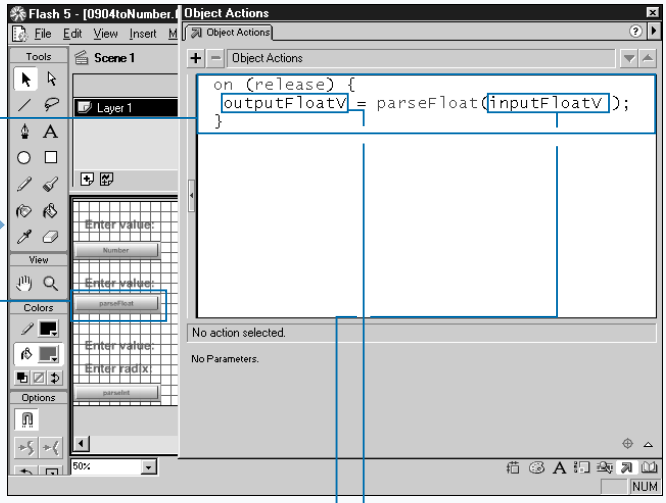
In addition to the `parseFloat` and `number` functions, you can also use the `parseInt` function to convert a string to a number. The `parseInt` function converts a string to an integer. An integer is a number that does not contain decimal places. The syntax for the `parseInt` function is

`parseInt(expression, radix);`.

Use the `expression` argument to specify the string you want to convert. Use the `radix` argument to specify the base from which you want to convert the string. Valid values are 2 to 36. The `parseFloat` function returns the value in base 10.

CONVERT A STRING TO A NUMBER





USING THE NUMBER FUNCTION

1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file tonumber.fla, which you can find on the CD that accompanies this book.

2 Click to open the Actions panel.

3 Convert a string to a number using the `number` function.

USING THE PARSEFLOAT FUNCTION

4 Select the frame, button, or movie clip to which you want to add ActionScript.

5 Convert a string to a number using the `parseFloat` function.

■ Associating the variable with a dynamic text box enables you to display the value the variable returns to the user.

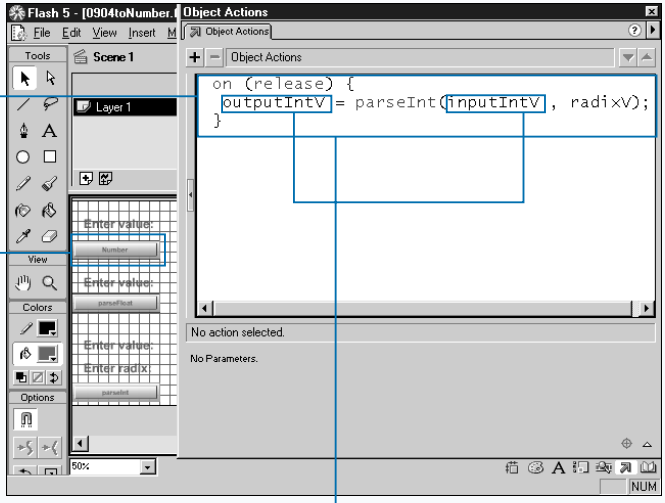
■ Associating the variable with an input text box enables you to accept user input.

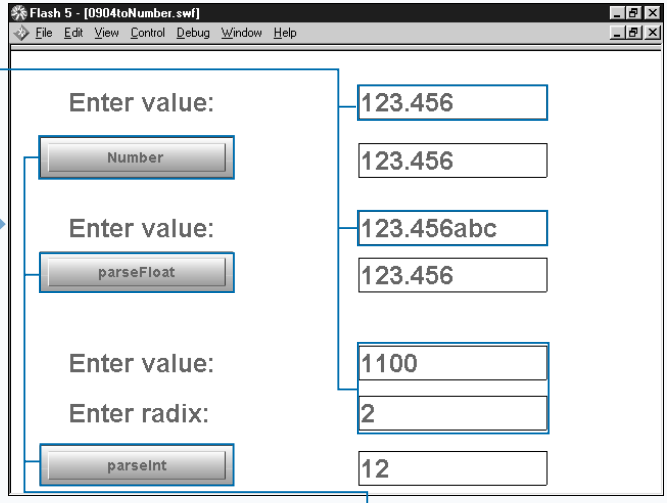
Extra

You normally count 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and then you add a new column to form the number 10. You count in base 10. You work with 10 digits. You can count in other bases. For example, you can count in base 2. When you count in base 2, you only have two digits with which to work — 0 and 1. You start a new column after you use the second digit. The table shown here illustrates:

BASE TEN	BASE TWO	COMMENTS
0	0	
1	1	
2	10	Using base two, you are out of digits. Form a new column.
3	11	
4	100	You are out of digits again. Form a new column.
5	101	
6	110	
7	111	

You can use the `parseInt` function to enter a number in a base other than 10 and return a base 10 value. Use the `radix` argument to specify the base of the number you are entering.





USING THE PARSEINT FUNCTION

6 Select the frame, button, or movie clip to which you want to add ActionScript.

7 Convert the string to a number using the `parseInt` function.

■ Associating the variables with a dynamic text box enables you to display the value the variable returns to the user.

■ Associating the variables with an input text box enables you to accept user input.

8 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

9 Type values in the text boxes.

10 Click the buttons to test your movie.

■ The script applies the `number`, `parseFloat`, and `parseInt` functions to your entries.

EVALUATE FOR MATHEMATICAL ERRORS

You cannot perform mathematical calculations using non-numeric values. If you try to perform a mathematical calculation using a non-numeric value, ActionScript will return an error message. When you request information from the user, the user may enter a non-numeric value when a numeric value is required. You can use the `isNAN` function to test whether a value is numeric or non-numeric. If a value entered by the user is non-numeric, when a numeric value is required, you can send a message requesting that the user enter a number.

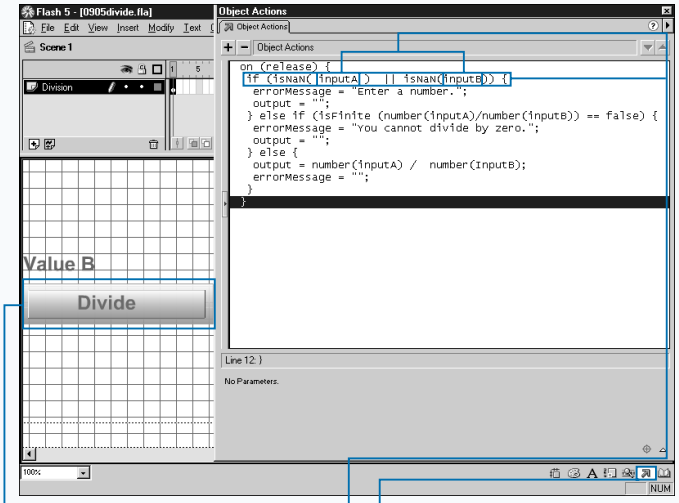
The syntax for the `isNAN` function is `isNaN(expression);`. Use the `expression` argument to enter the expression you want to evaluate to determine if it is a number. The `isNAN` function returns `true` if the expression is not a number and it returns `false` if the expression is a number.

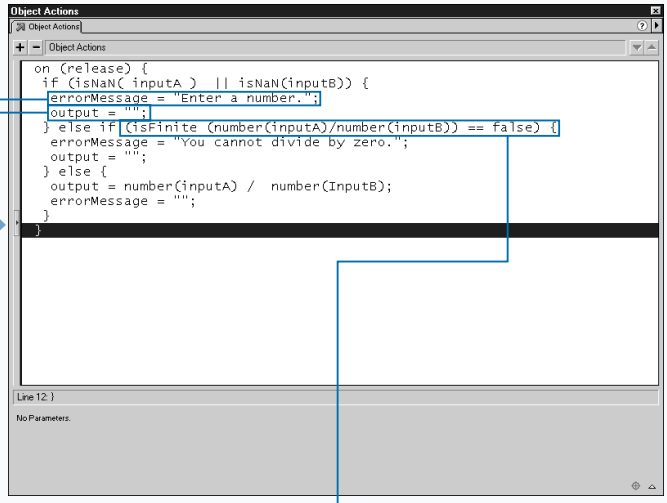
When an expression returns `infinity`, that can be an indication of a mathematical error. For example, when you divide by 0, the result is `infinity`. You can use the `isFinite` function to test whether an expression returns `infinity`.

The syntax for the `isfinite` function is `isFinite(expression)`. Use the `expression` argument to specify the expression you want to test. The `isFinite` function returns `true` if the expression is not equal to `infinity` and `false` if the expression is equal to `infinity`.

The `isFinite` expression also returns `false` if the expression is equal to negative infinity or positive infinity. ActionScript defines negative infinity as any value that is smaller than the smallest value ActionScript can represent. Positive infinity is any value that is larger than the largest value that ActionScript can represent. The smallest value ActionScript can represent is approximately `5e324`. The largest value ActionScript can represent is approximately `1.7976931348623158e+308`.

EVALUATE FOR MATHEMATICAL ERRORS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file tonumber.fla, which you can find on the CD that accompanies this book.

2 Click to open the Actions panel.

3 Determine if the values are numbers using the `isNAN` function.

■ Associating the variables with input text boxes enables you to accept user input.

4 Send an error message to the user if a value is not a number.

5 Set the results of the calculation to a blank space if a value is not a number.

■ Associating the variables with dynamic text boxes enables you to display the values the variables return to the user.

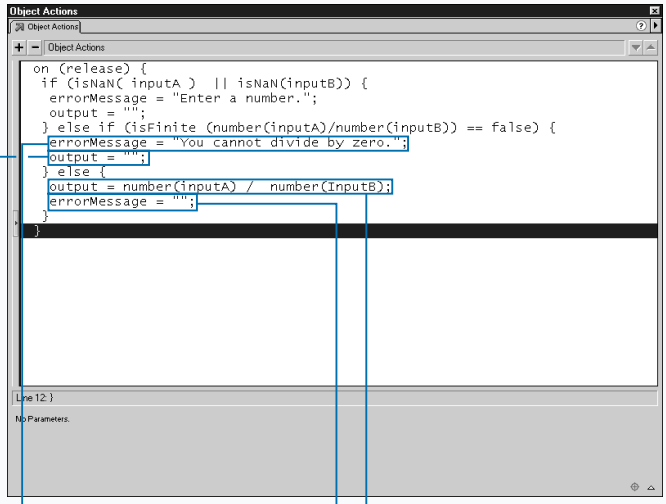
6 Test for infinity using the `isFinite` function.

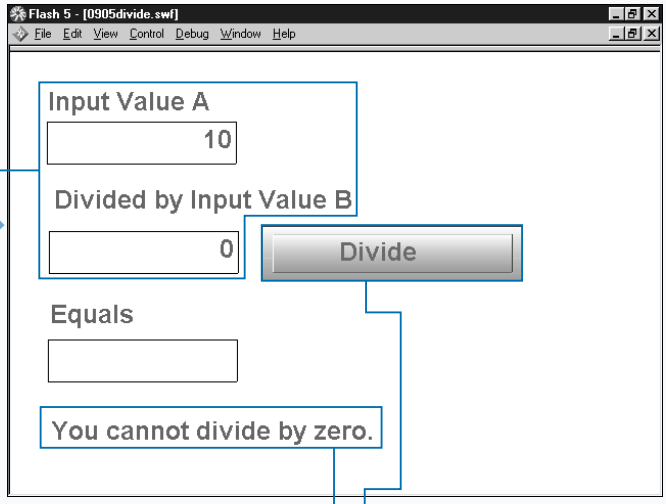
Extra

You can test user entries to determine if they are numbers. In the example that follows, if all entries are not numbers, the script returns an error message. If all entries are numbers, the script calculates an age based on the entries made.

Example:

```
on (release) {
    if (isNaN(monthV) || isNaN(yearV) ||isNaN( DayV)) {
        errorMessage = "Please enter numbers.";
        result = " ";
    } else {
        monthU = monthV-1;
        birth = new Date( yearV,MonthU, DayV );
        today = new Date();
        age = Math.floor ( (((((today-birth)/1000)/60)/60)/24)/365);
        result = "You are " + age + " years old.";
        errorMessage = "";
    }
}
```





7 Send an error message to the user if the expression returns `infinity`.

8 Set the results of the calculation to a blank space if the expression returns `infinity`.

9 Perform the calculations if there are no errors.

10 Set the error message to a blank space if there are no errors.

11 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

12 Type the value in the input text boxes.

13 Click the button to test your movie.

■ ActionScript will display an error message if your entry in not a number or if the calculation evaluates to `infinity`.

CREATE A SCROLLABLE TEXT BOX

When working with dynamic or input text boxes, you may have more text than will fit in the visible area of the text box. You can use ActionScript to create buttons or scroll bars that enable the user to scroll up or down the text box or move to a specified line.

ActionScript numbers each line in a text box sequentially. The first line is line number one. Variables that are associated with dynamic or input text boxes have a `scroll` and `maxscroll` property. The `scroll` property returns the topmost visible line number in the text box. You can set or retrieve this value. The `maxscroll` property returns the topmost visible line in the text box when the last line of text is visible. You can retrieve the `maxscroll` value, but you cannot set the `maxscroll` value.

The syntax for the `scroll` property is `variableName.scroll = x;` . Use the `variableName` argument to specify the variable name associated with the text box. Use the `x` argument to specify the line number you want to assign to the scroll property. For example, if you want the user to move to line one on the release of the mouse after pressing a button, use the syntax shown here:

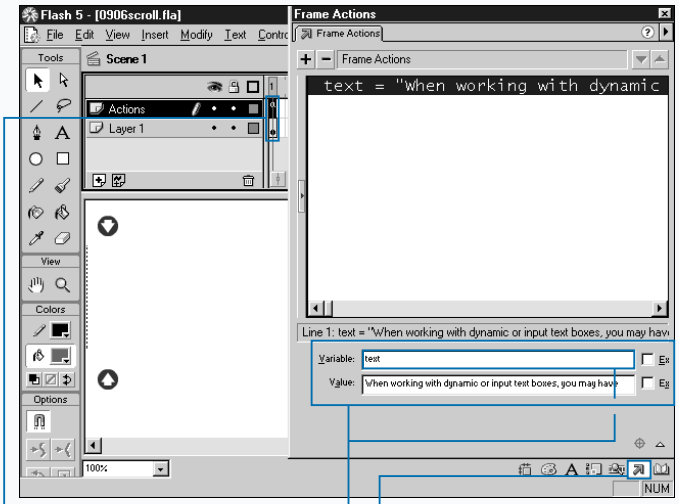
```
on (release) {
     text.scroll = 1;
}
```

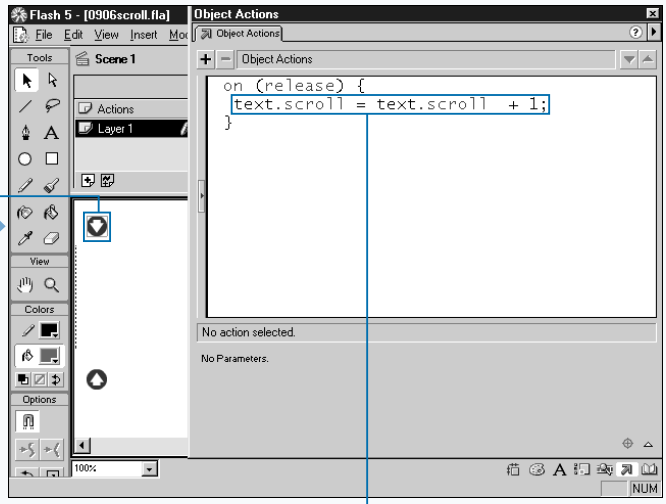
To retrieve the scroll value, use the syntax `scrollValue = variableName.scroll;` . As the user scrolls through a text box, the Flash Player updates the variable `scrollValue`.

The syntax for the `maxscroll` property is `variableName.maxscroll;` . Use the `variableName` argument to specify the variable name associated with the text box. The example shown here displays the last lines of text when the user releases the mouse after pressing a button:

```
on (release) {
     text.scroll = text.maxscroll;
}
```

CREATE A SCROLLABLE TEXT BOX





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file scroll.fla, which you can find on the CD that accompanies this book.

2 Click to open the Actions panel.

3 Assign text to an input or dynamic text field.

■ In this example, text is the variable name and the variable is associated with an input text box.

4 Select the frame, button, or movie clip to which you want to add ActionScript.

■ The Actions panel will become active.

■ This part of this example uses a button.

5 Set the scroll value to its current value plus one.

■ Each time the user releases the mouse after clicking the button, the text box will scroll down one line.

Extra

You can use the script shown here to create a continuous scroll button. When you press the button, the text box scrolls until you release the button. Start by creating a button and attach the script shown here.

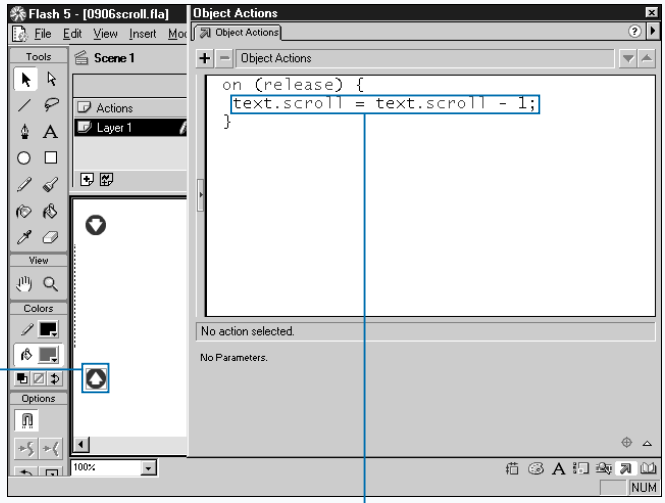
Example:

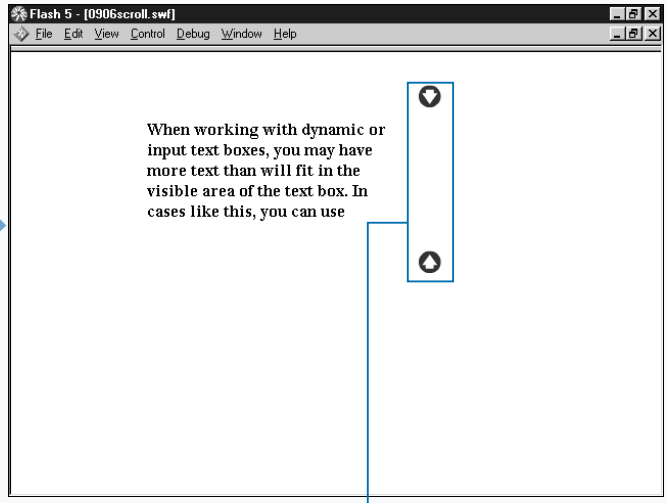
```
on (press) {
    var press = true;
}
on (release) {
    var press = false;
```

Convert the button to a movie clip. Select the button and click Insert ➪ Convert to Symbol to open the Symbol Properties dialog box. Type a symbol name in the Name field and select Movie Clip as the Behavior. Attach the script shown here to the movie clip. In this example, the variable text is the name of the text box.

Example:

```
onClipEvent (enterFrame) {
    if (press == true) {
        _root.text.scroll = _root.text.scroll + 1;
    }
}
```





6 Select the frame, button, or movie clip to which you want to add ActionScript.

■ The Actions panel will become active.

7 Set the scroll value to its current value minus one.

■ Each time the user releases the mouse after clicking the button, the text box will scroll up one line.

8 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

9 Click the buttons to test your movie.

■ When you click the down button the text scrolls down. When you click the up button the test scrolls up.

CREATE A CUSTOM FUNCTION

A function is a reusable block of code. You use arguments to pass values to a function. The function performs operations on the values passed to it and returns the results. ActionSctipt comes with several predefined functions. For example, the functions `getProperty`, `getTimer`, `random`, and `getVersion` are all predefined by ActionScript.

You can create your own custom functions using the `function` action. The syntax for the `function` action is

```
function functionName (a1, a2, ...aN){
statement(s)
};
```

Use the `functionName` argument to name your function. Use a1 to aN to specify the arguments to pass to the function. Arguments are optional. Use the `statement` argument to define the operation the function performs.

When creating a custom function, you use the `return` action to specify the value the function should return. The syntax for the return action is `return expression;` . Use the `expression` argument to define the value to return. The `expression` argument is optional.

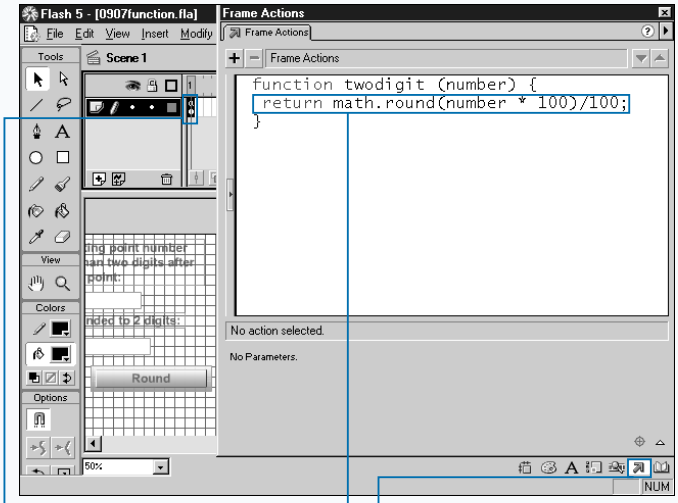
The following is an example of a custom function that you can use to round a floating-point number to two digits and return the value

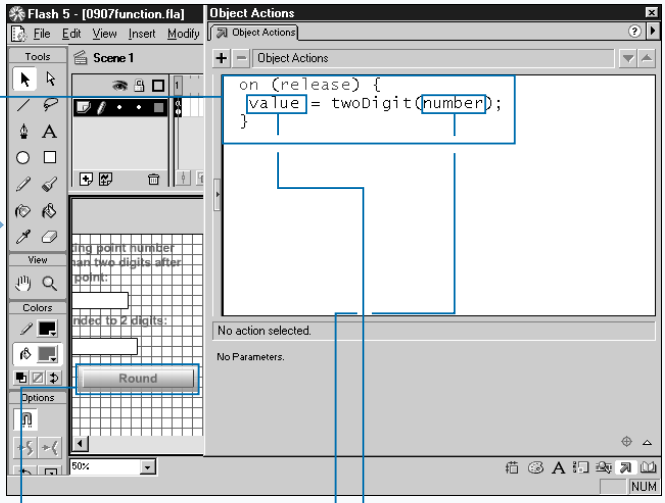
```
function twoDigit (number) {
    return math.round(number * 100)/100;
}
```

The name of the function is `twoDigit`. It takes one argument — number — and it returns the value of the number rounded to two digits after the decimal point.

When using Normal mode, you use the `evaluate` action to include a custom function to your script. The `evaluate` action creates an empty line with a semicolon.

CREATE A CUSTOM FUNCTION





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file function.fla, which you can find on the CD that accompanies this book.

2 Click to open the Actions panel.

3 Create a function.

4 Select the frame, button, or movie clip to which you want to add ActionScript.

5 Use the function you created.

■ Associating the variable with a dynamic text box enables you to display the value the variable returns to the user.

■ Associating the variable with an input text box enables you to receive input from the user.

Extra

You can call a function from any Timeline, including the Timelines of loaded movies. To call a function from another Timelime, precede the function with the target path using dot syntax.

Example:

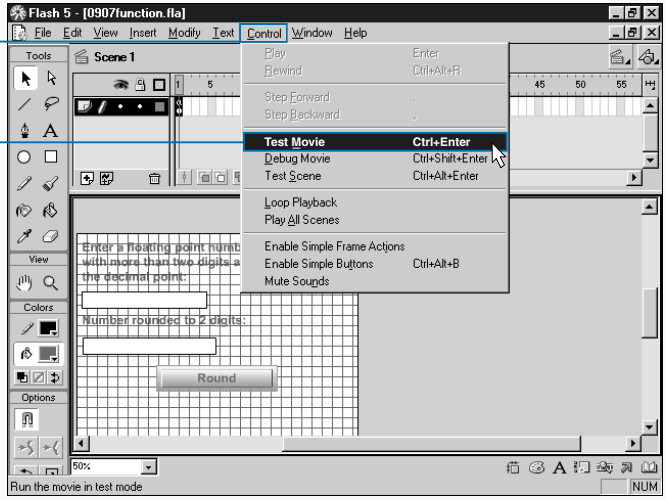
```
_root.sampleMC.twoDigit(123.456);
```

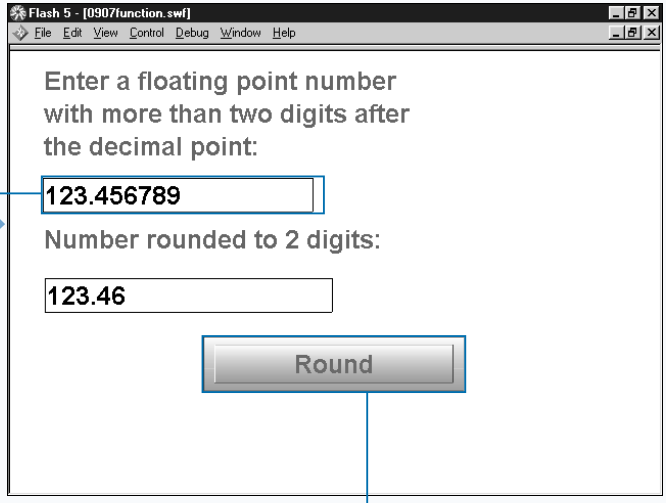
It is a good practice to document your functions with comments. Include the input, outputs, and purpose of the function in your comments.

When creating a function, it is a good idea to use local variables. The scope of a local variable is limited to the curly braces that enclose it. Using local variables in a function prevents other scripts in the movie from reading the variables. ActionScript treats arguments passed to a function as local variables.

If you omit arguments during a function call, ActionScript passes the argument to the function as `undefined`. This can cause errors when you export your movie. If you send extra arguments during a function call, ActionScript ignores them.

A function does not have to return a value. For example, you can create a function that initializes values.





6 Click Control ➪ Test Movie to move to the test environment.

7 Type a floating point number with more than two digits after the decimal point.

8 Click the button to test your movie.

■ The function rounds the number you entered to two digits.

VIEW THE HIERARCHY OF MULTIPLE MOVIES

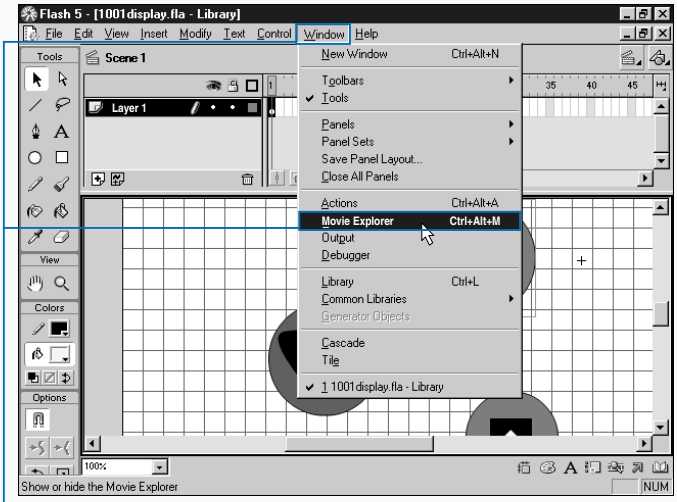
You create Flash movies on a Timeline. ActionScript refers to the main Timeline as `_level0`. You create other levels when you use the `loadMovie` command to load SWF files. You assign newly created levels a level number. You can use any level number you want, but only one movie can be on a given level at a time.

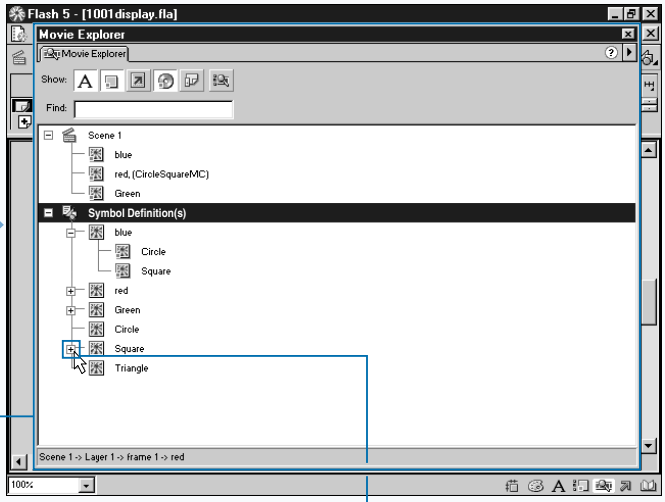
You can turn any Flash movie into a movie clip. You can embed a movie clip symbol within a Flash movie, and you can embed multiple instances of a movie clip symbol inside itself or other movie clips. Loaded movies can have movie clips on their Timeline. In addition, those movie clips can have movie clips on their Timelines.

Flash Player organizes the movies and movie clips in your movie into a hierarchy, called the *display list*. Movie Explorer enables you to view the display list while you are authoring a Flash movie. You can also view the display list in Debugger. The display list graphically represents the relationship between Timelines. If you make a change to a Timeline, your change will affect all of the Timelines on the Timeline that you change. Timelines can send messages to other Timelines. For example, one Timeline can tell another Timeline to stop playing, begin playing, or perform any other action.

When you place a movie or movie clip on the Timeline of another movie or movie clip, they form a relationship. The movie or movie clip that you place on the Timeline is the *child*. The Timeline on which you place the movie or movie clip is the *parent*. Any change that you make to the parent will affect the child.

VIEW THE HIERARCHY OF MULTIPLE MOVIES





1 Click Window ➪ Movie Explorer.

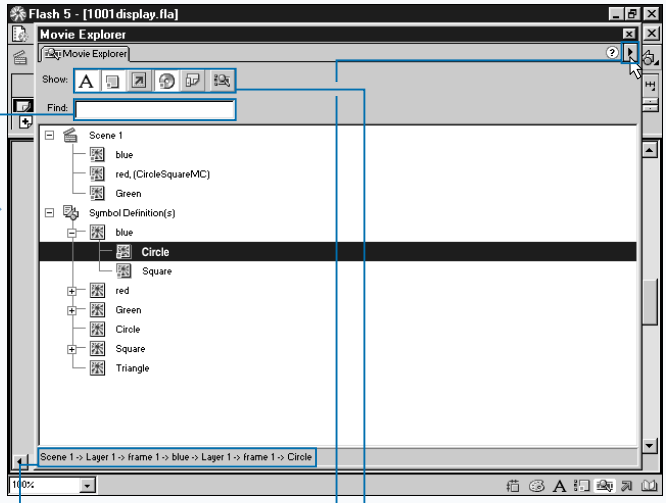
■ Movie Explorer opens, and the display list appears.

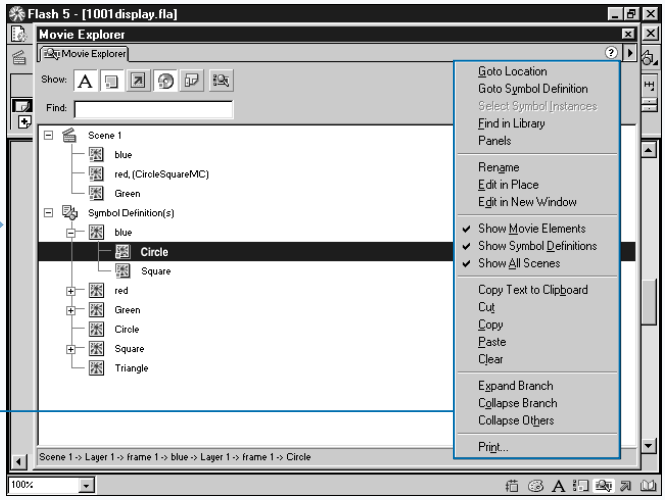
2 Click ⊞ to view nested Timelines.

Extra

The display list in Movie Explorer shows the hierarchy of the movie. You can use the display list to see a graphical representation of a movie and all the objects included in the movie. You will find Movie Explorer particularly useful when you are examining a movie developed by someone else. Movie Explorer provides you with a mechanisn that allows you to easily understand how a movie is put together.

When using Debugger, you can view the display list. When viewing the display list in Debugger, the addition and removal of movies and movie clips display instantly. The display list in Debugger also presents a hierarchical representation of the movie.





■ The path to the selected item appears on the Status bar.

■ You can use the Find text box to search for items.

■ You can use the Filtering buttons to limit the categories of items Movie Explorer displays.

3 Click ▶ to display the pop-up menu.

■ The pop-up menu appears.

Note: You can use the menu to navigate Movie Explorer, edit objects, or print the hierarchy.

ASSIGN TARGET PATHS

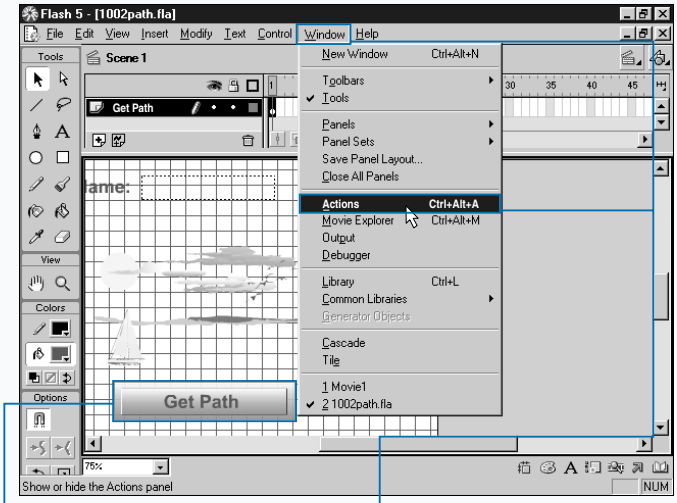
Flash defines a target path as the address of the Timeline you want to target. There are two types of target paths: absolute and relative. An *absolutepath* is a fixed address; it is always the same. A *relativepath* is not fixed. You determine the relative path based on the Timeline that calls the action.

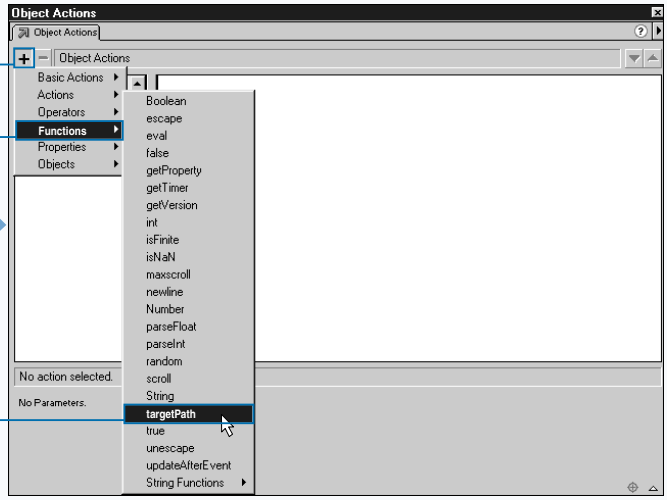
ActionScript bases both absolute and relative target paths on the movie hierarchy. You can view the movie hierarchy using the Movie Explorer display list. An absolute path address begins with the name of the level the targeted movie clip was loaded into and includes the name of each instance until it reaches the target instance. For example, if you place a movie clip with an instance name of `redMC` on the main Timeline, the absolute path to `redMC` would be `_level0.redMC`. If you add an instance named `triangleMC` to the Timeline of the movie clip `redMC`, the absolute path to `triangleMC` would be `_level0.redMC.triangleMC`.

There are two methods used to specify a target path: dot syntax and slash syntax. *Dot syntax* separates each item in the path with a dot — for example, `_level0.redMC.triangleMC`. *Slash syntax* separates each item in the path with a slash — for example, `_level0/redMC/triangleMC`.

ActionScipt bases a relative path on the relationship between the controller Timeline and the target Timeline. You can only use a relative path to target objects on the same level. Use the alias `_parent` to refer to the parent of the current Timeline. When using slash syntax, you use `..` to move up the hierarchy.

ASSIGN TARGET PATHS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file path.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Click ⊞ ➪ Functions ➪ targetPath to select the `targetPath` Action.

Extra

When using dot syntax, you can use the keyword `this` to refer to the current Timeline.

You can use the _target property to return the target path in slash notation. The syntax for the `_target` property is

```
instanceName._target;
```

Use the `instance name` argument to specify the movie clip for which you want the target path.

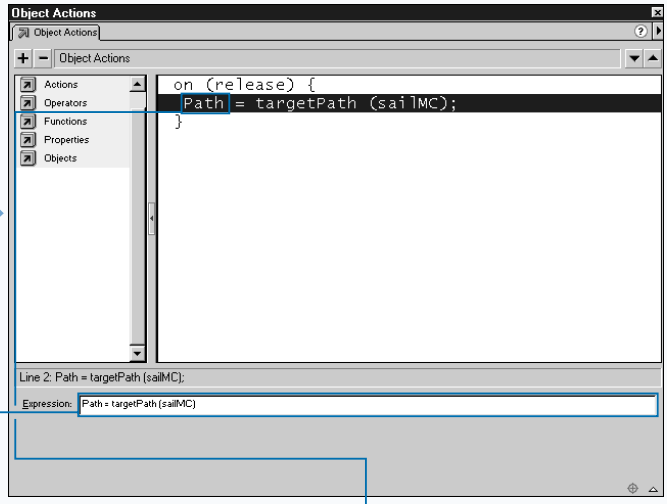
You can use the `targetPath` function to return the target path in dot notation. The syntax for the target path functions is

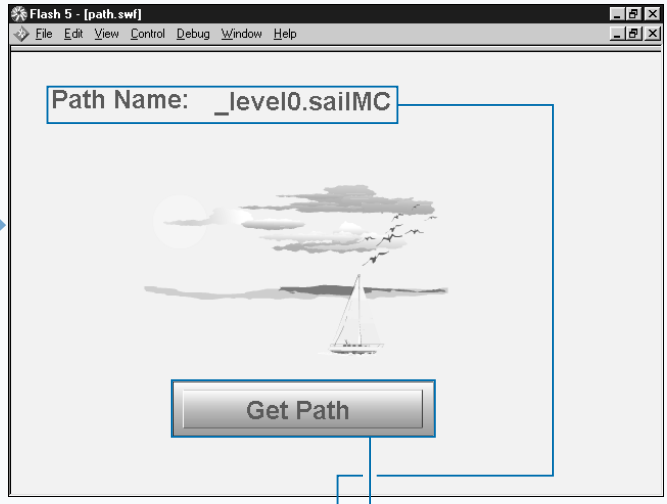
```
targetpath(instanceName);
```

Here also, use the `instanceName` argument to specify the movie clip for which you want the target path.

You use `_level` followed by the level number to refer to the main timeline for a level. If you want to reference the main Timeline for level 5, you use `_level5`. If you want to reference to the main Timeline for level 0, you use `_level0`.

You can also use `_root` to refer to the main Timeline. You use the `_root` property to refer to the main Timeline for the current level. For example, if you are on level 5 `_root` refers to the main Timeline for level 5. If you are on level 0, `_root` refers to the main Timelime for level 0.





4 Type your expression.

■ This expression retrieves the target path and assigns it to the variable path.

■ If you associate the variable with a dynamic text box, you can display the value the variable returns to the user.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button to test your movie.

■ The `targetPath` function retrieves the target path.

LOAD AND UNLOAD MOVIES AND MOVIE CLIPS

The `loadMovie` action enables you to play several movies or movie clips in concert or in sequence without closing Flash Player. Using the `loadMovie` action, you can easily change movies without opening a new Web page. The syntax for the `loadMovie` action is

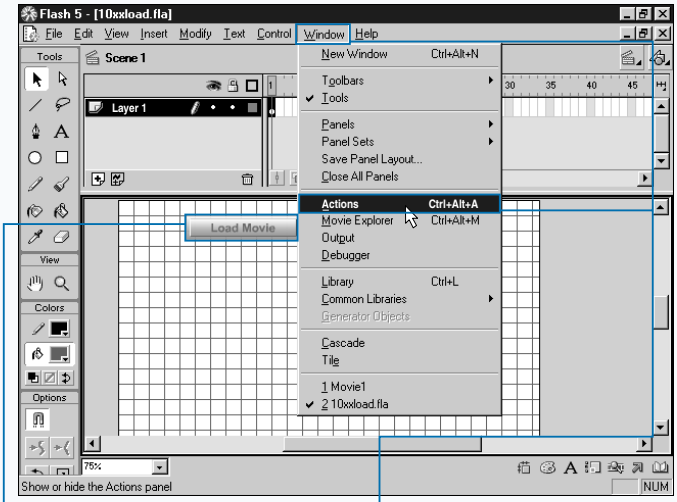
```
loadMovie(URL , target/location, variable);
```

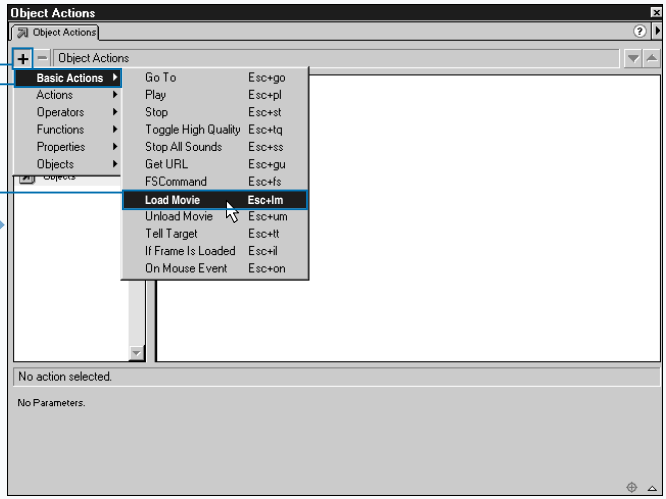
Use the `URL` argument to specify the absolute or relative URL of the SWF file that you want to load. Relative paths must be based on the location of the SWF file, and the URL must be in a subdomain of the current movie. When you are testing Flash movies, all SWF files must be stored in the same folder.

You can use the `loadMovie` action to replace a movie clip with a Flash movie. The targeted movie clip must have a unique instance name. The loaded movie will inherit the position, scale, and rotation properties of the targeted movie clip. If you want to replace a movie clip, use the `target` argument to specify the movie clip that the loaded movie clip replaces. The `target` argument is optional.

Flash organizes movies into a hierarchy called *levels*. You use the `_level` property to specify the level. The syntax for the `_level` property is `_levelN;` . Use the `N` argument to specify the level number. The movie loaded at `_level0` sets the frame rate, background color, and frame size for all other movies.

LOAD AND UNLOAD MOVIES AND MOVIE CLIPS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file load.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Click ⊞ ➪ Basic Actions ➪ Load Movie to select the `loadMovie` Action.

Extra

The `loadMovie` action enables you to break a large project down into its component parts and load each movie when needed. Several small movies play faster and use memory more efficiently than one large movie.

You can use this script to replace a movie clip with a movie when the user releases the mouse after clicking a button. Note the location at which the loaded movie appears. Also note that if you rotate or scale the movie clip, the loaded movie inherits the properties.

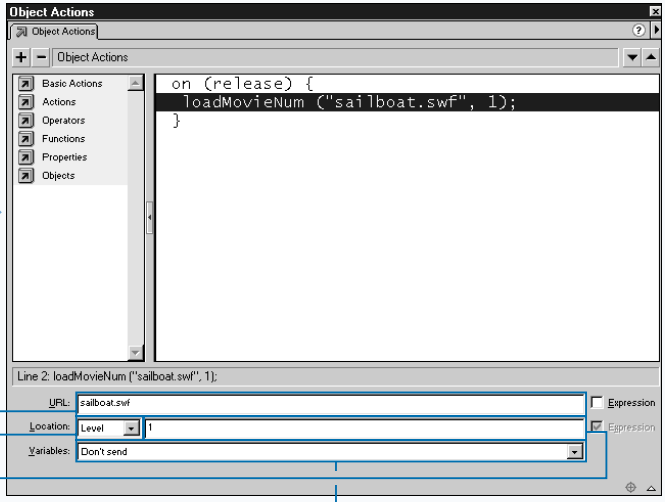
Example:

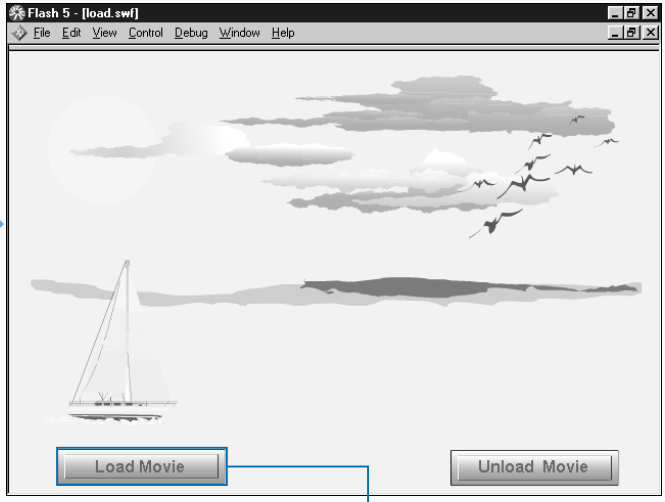
```
on (release) {
    loadMovie ("redstar.swf", "blueStarMC");
}
```

You can use this script to load a movie and send variables using the `post` method.

Example:

```
loadMovieNum ("sample.swf", 1, "POST");
```





4 Type the URL of the movie that you want to load.

5 Click ▾ and select the location where the movie will load.

■ To load a movie, select Level. To load a movie clip, select Target.

6 Type a level number.

■ If you are replacing a movie clip, type the target path instead of performing step 6.

7 Click ▾ and select how you want to send variables.

■ You can choose from Don't Send, Send Using Get, or Send Using Post.

8 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

9 Click the button to load your movie.

■ Your movie loads.

CONTINUED

LOAD AND UNLOAD MOVIES AND MOVIE CLIPS (CONTINUED)

The main Timeline for every movie is located on `_level0`. When loading a movie, you can specify a level. You can use any level number that you want, but only one movie can be on a given level at a time. When loading a movie, use the `location` argument to specify the level into which you want to load the movie. When you load a movie into level 0, the new movie will replace the current movie, and all other levels will be unloaded. When you load a movie into a level occupied by another movie, the new movie will replace the movie at that level. When you load a movie into an unoccupied level, all existing movies will remain, and the new movie will be loaded.

You can send variables to the CGI scripts using the `loadMovie` action. The `Variable` parameter in Normal Mode presents you with three choices: Send Using Get, Send Using Post, and Don't Send. Use Send Using Get to append a small number of variables to the end of the URL. Use Send Using Post to send variables separate from the URL. Send Using Post enables you to send a larger number of variables. Use Don't Send if you do not want to send any variables. Use the `variable` argument to specify the method that you want to use to send associated variables.

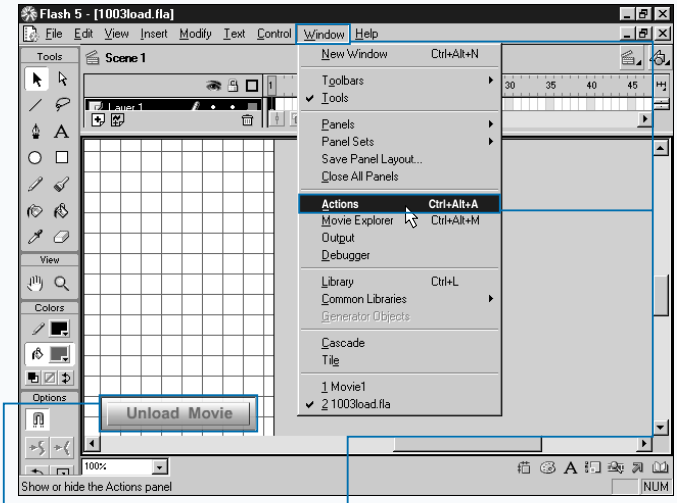
You use the `unloadMovie` action to remove a movie loaded using the `loadMovie` action. The syntax for the `unloadMovie` action is

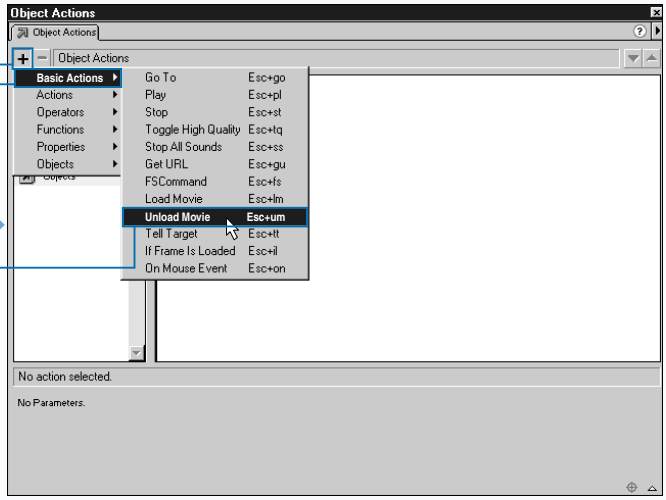
```
unloadMove(location);
```

Use the `location` argument to specify the level of the target movie that you want to unload.

When you select `loadMovie` in Normal Mode, Flash may substitute `LoadMovieNum`. For an explantion of `LoadMovie` and `LoadMovieNum`, see the appendix.

LOAD AND UNLOAD MOVIES AND MOVIE CLIPS (CONTINUED)





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file load.fla, which you can find on the CD that accompanies this book.

■ This example uses a button.

2 Click Window ➪ Actions to open the Actions panel.

3 Click ⊞ ➪ Basic Actions ➪ Unload Movie to select the `unloadMovie` Action.

Extra

The following example loads a movie.

Example:

```
loadMovieNum ("sample.swf", 1);
```

The following example unloads a previously loaded movie.

Example:

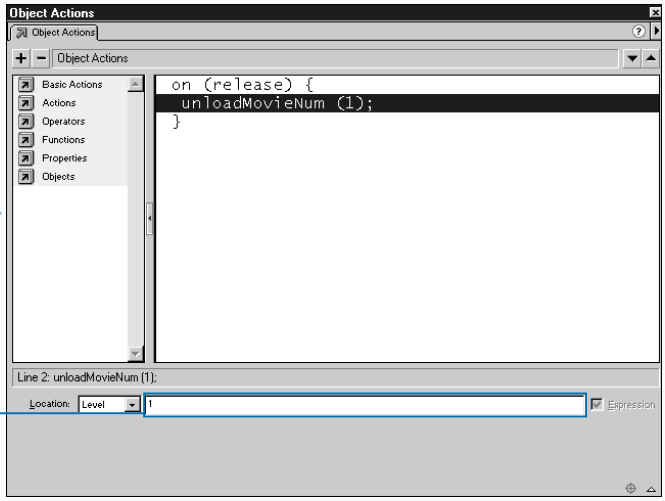
```
unloadMovieNum (1);
```

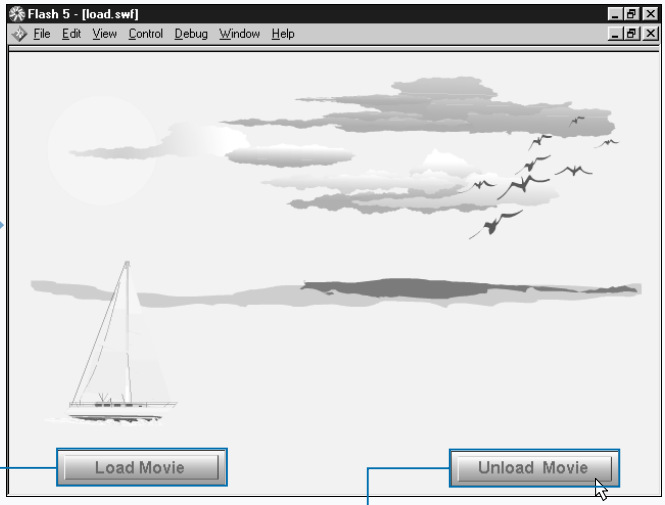
The following example replaces a movie clip with a movie.

Example:

```
loadMovie ("sample.swf", "sampleMC");
```

When you load a movie on another movie, the loaded move is aligned with the upper left corner of the movie into which it is loaded. To ensure proper alignment, it is a good idea to make both movies the same size.





4 Type a Level number.

5 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

6 Click the button to load your movie.

■ Your movie loads.

7 Click the button to unload your movie

■ Your movie unloads.

USING TELL TARGET

You can use the `tellTarget` action to send statements to a movie clip. You can also use `tellTarget` to send statements to a movie that was loaded using the `loadMovie` action. You assign the actions that you want the target movie or movie clip to perform to a frame, button, or movie clip. The frame, button, or movie clip to which you assign the actions is the controller. The movie or movie clip that receives the actions is the target. A targeted movie clip must have a unique instance name and must be on the Timeline of the movie on which you create the controller. The syntax for the `tellTarget` action is

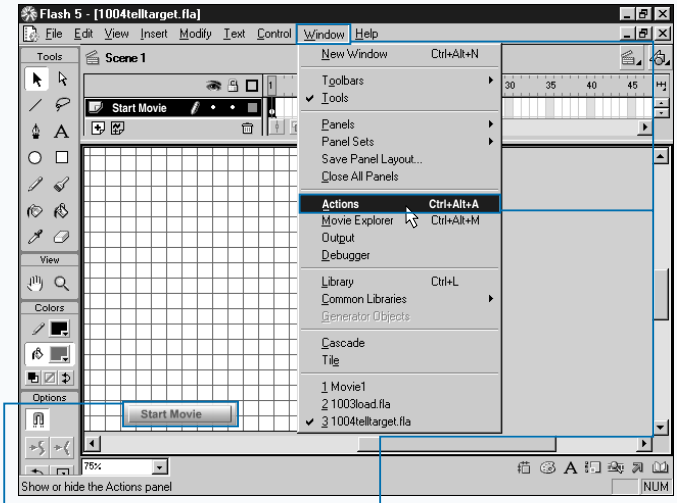
```
tellTarget(target){
statement;
}
```

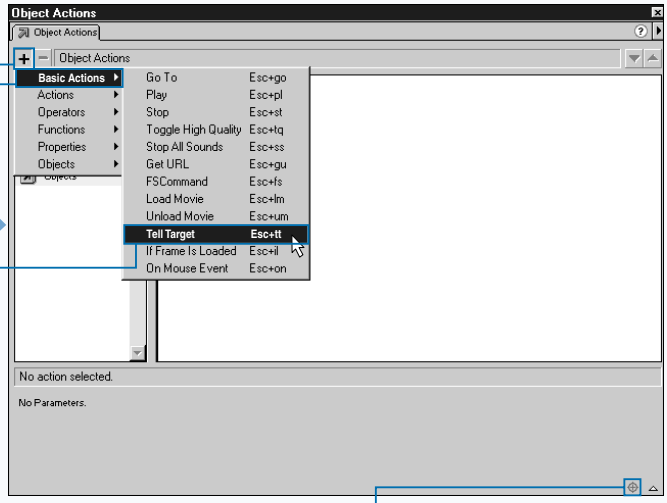
Use the `target` argument to specify the target path to be controlled. Use the `statement` argument to specify the statements that you want to send. You can use the Insert Target Path dialog box to specify the target movie clip. To open the Insert Target Path dialog box, click the Insert Target Path button in the lower-right corner of the Actions panel.

When using the Target Path dialog box, use the Notation radio buttons to specify the type of notation that you want to use. You can choose from dot and slash. The default notation type is dot. You use the Mode radio buttons to specify the mode; you can choose Relative or Absolute. The default mode is Relative. If you choose the Relative mode, the tree view displays only movie and movie clip instances that are located on the controller Timeline and their children. You can use the keyword `this` to refer to the current Timeline.

If you choose the Absolute mode, the tree view displays all the loaded movies and movie clips and uses `_level` or `_root` to designate their location.

USING TELL TARGET





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file telltarget.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

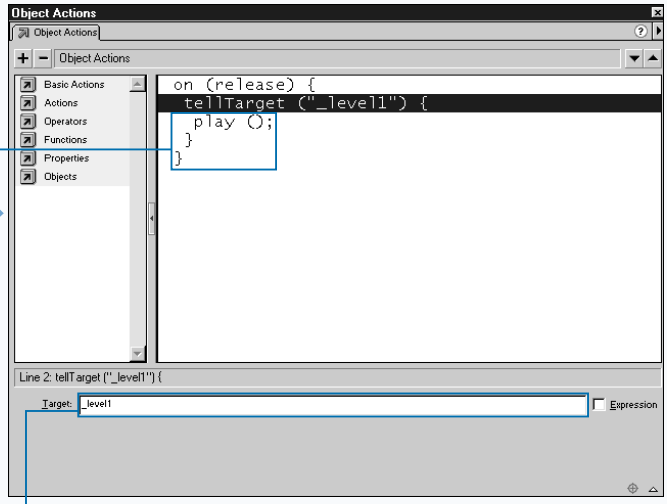
3 Click ⊞ ➪ Basic Actions ➪ Tell Target to select the `tellTarget` Action.

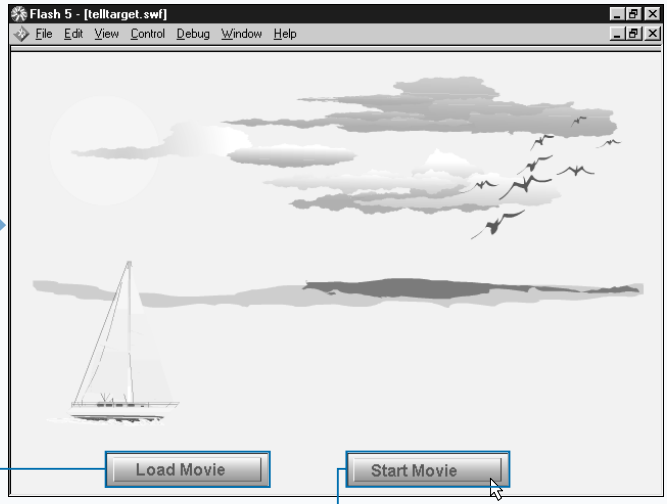
■ If you want to open the Target Path dialog box, click the Insert Target Path dialog box.

Extra

Before you can tartget a a Timeline, the Timeline must be in Flash Player. You cannot target the Timeline of a movie or movie clip that is not loaded. If you are targeting a movie clip, the playhead must be located in one of the frames of the movie clip. If the movie clip is in frames 1 to 10 and the playhead is in frame 11, you cannot target the movie clip.

Flash 5 deprecated the `tellTarget` action. You should use the `with` action when you are authoring for a Flash 5 environment.





4 Type a Level number.

5 Type your statements.

■ Type the actions you want to execute.

6 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

7 Click the button to load your movie.

■ Your movie will load.

8 Click the button to start your movie

■ Your movie will start.

USING THE WITH ACTION

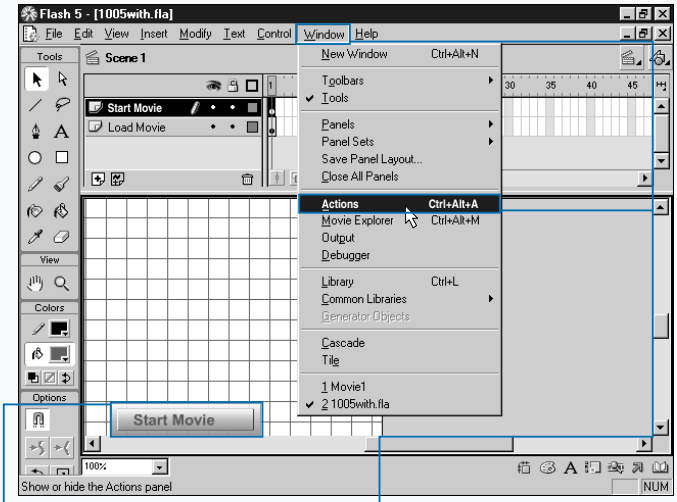
You can use the `with` action instead of the `tellTarget` action to send statements to movie clips and to movies that have been loaded with the `loadMovie` command. As when using the `tellTarget` action, you assign the actions that you want the target movie to perform to a frame, button, or movie clip. The frame, button, or movie clip to which you assign the actions is the controller. The movie or movie clip that receives the actions is the target. The `with` action is preferable to the `tellTarget` action because Flash 5 deprecated the `tellTarget` action. In addition, the `with` action has a few advantages over the `tellTarget` action. For example, the `with` action can take a movie clip or other object as a target. When you use the `tellTarget` action, you must specify a target path, and the `tellTarget` action cannot target objects. As with the `tellTarget` action, you can use the `with` action to perform multiple actions on the same target. The syntax for the `with` action is

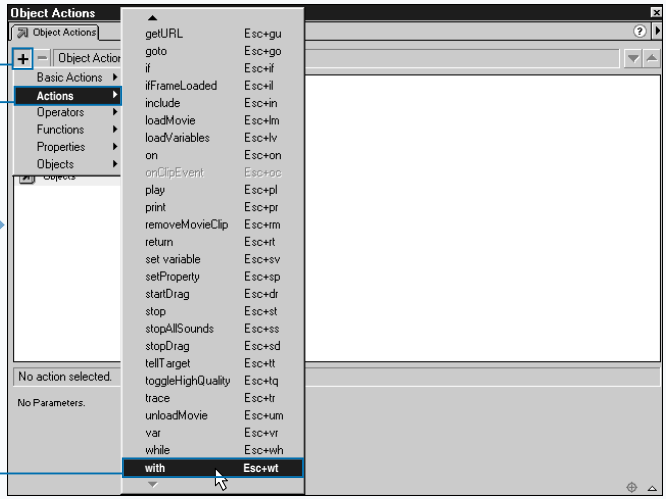
```
with(object) {
statement(s);
}
```

Use the `object` argument to specify the movie, movie clip, or object on which you want to execute the statements. Use the `statement` argument to specify the statements that you want to execute.

ActionScript uses the following order to determine the scope of the `with` action: 1) the objects under the innermost `with` action, 2) the objects under the outermost `with` action, 3) the activation object (the activation object is an object that ActionScript creates automatically when you call a function; it holds the local variable called by the function), 4) the movie clip that contains the script that you are executing, and 5) global objects such as `Math` and `String`.

USING THE WITH ACTION





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file with.fla, which you can find on the CD that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Click ⊞ ➪ Actions ➪ with to select the `with` Action.

Extra

Using the `with` action, the following script adjusts the `_xscale`, `yscale`, and `_alpha` properties of the block movie clip when the user releases the mouse after clicking a button.

Example:

```
on (release) {
    with (block) {
        _xscale = 50;
        _yscale = 75;
        _alpha = 40;
    }
}
```

Using the `tellTarget` action, the following script adjusts the `_xscale`, `yscale`, and `_alpha` properties of the block movie clip when the user releases the mouse after clicking a button:

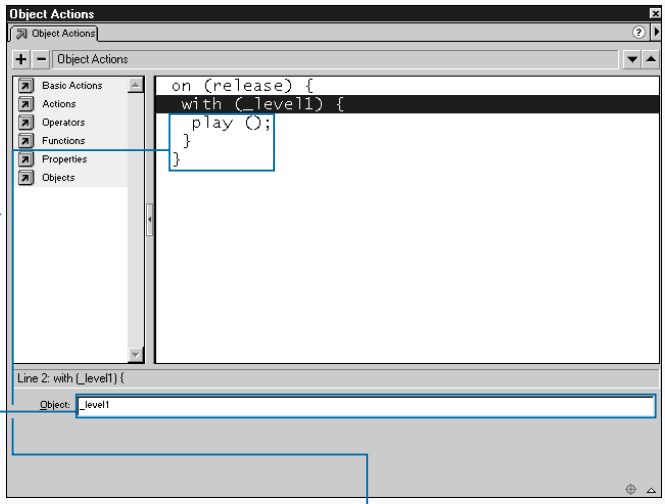
Example:

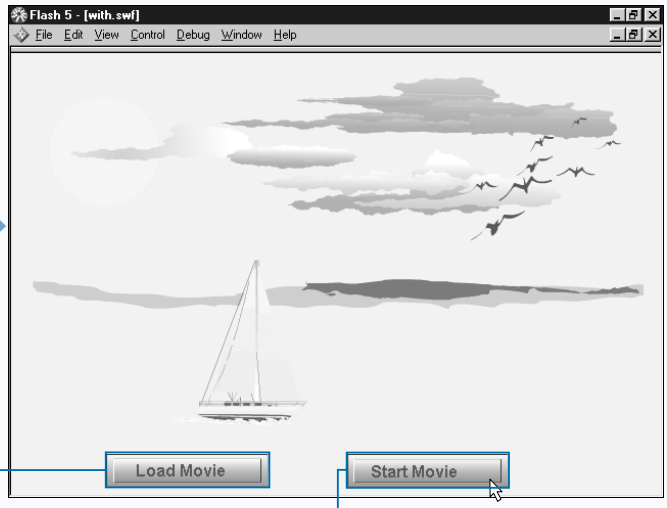
```
on (release) {
    tellTarget ("block") {
        _yscale = 75;
        _xscale = 50;
        _alpha = 40;
    }
}
```

Using standard syntax, the following script adjusts the `_xscale`, `yscale`, and `_alpha` properties of the block movie clip when the user releases the mouse after clicking a button.

Example:

```
on (release) {
    block._xscale = 50;
    block._yscale = 75;
    block._alpha = 40;
}
```





4 Type a level number if you are targeting a movie loaded with the `loadMovie` action or type a target path if you are targeting a movie clip.

5 Type your statement(s).

- Type the statements you want to execute.

6 Move to the test environment

Note: See page 38 for instructions on how to test your movie.

7 Click the button to load your movie.

- The movie will load.

8 Click the button to start your movie

- The movie will start.

DUPLICATE MOVIE CLIPS

If you want to make a copy of a movie clip that is currently on the Stage, use the `duplicateMovieClip` action. The `duplicateMovieClip` action enables you to create a new instance of a movie clip while the movie is playing. The syntax for the `duplicateMovieClip` action is

```
duplicateMovieClip(target, newName, depth);
```

Use the `target` argument to specify the name of the instance that you want to duplicate. Use the `newName` argument to specify the name that you want to give to the new instance. The `depth` argument is used to specify stacking order. Stacking order determines how objects appear on the Stage when objects overlap. Objects with a higher depth number appear to be in front of objects with a lower number. Assign each object that you duplicate a depth number. If you place a new instance on the same depth level as an existing instance, the new instance replaces the existing instance.

Duplicated movie clips always begin playing in frame 1.

ActionScript does not copy the variables in the parent movie clip to the child movie clip. If you delete the parent movie clip, ActionScript will automatically delete the child.

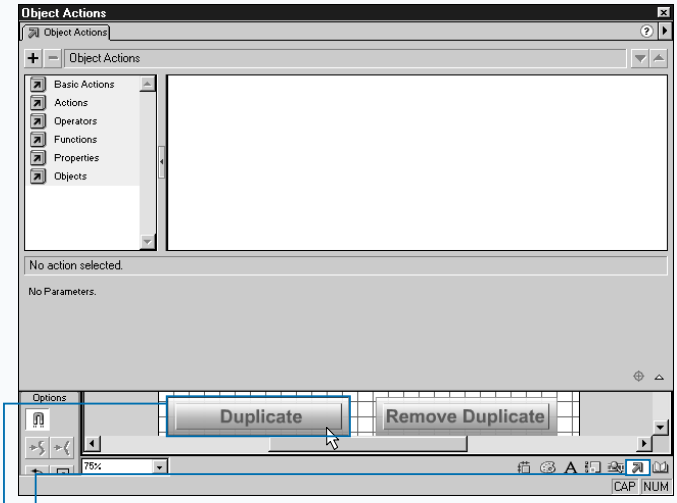
ActionScript places the child movie clip directly on top of the parent. You can use the `_x` and/or `_y` properties to change the location of the duplicated movie clip. Each instance of the duplicated movie clip must have a unique name. You can append an incrementing number to the end of the movie clip name to make each instance unique.

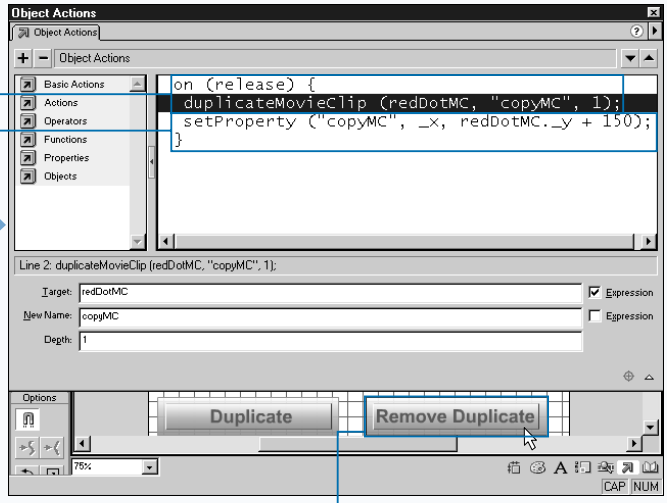
You use the `removeMovieClip` action to remove instances created with the `duplicateMovieClip` action. The syntax for the `removeMovieClip` action is

```
removeMovieClip(target);
```

Use the `target` argument to specify the movie clip that you want to remove. You can also use the `removeMovieClip` action to remove movie clips created with the `attachMovie` and `duplicateMovie` methods.

DUPLICATE MOVIE CLIPS





1 Click to open the Action panel.

2 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file duplicate.fla, which you can find on the CD that accompanies this book.

■ This part of this example uses a button.

3 Duplicate the movie clip.

4 Reset the **_x** property for the new instance so that the new instance does not appear on top of the existing instance.

■ Each time the user clicks the Duplicate button, the new duplicate instance replaces any existing duplicate instance.

5 Select the frame, button, or movie clip to which you want to add ActionScript.

Extra

Using the following script, you make the movie clip that you are going to duplicate draggable. You start by attaching the script shown here to a button to make the button draggable, and then you click Insert➪Convert to Symbol to convert the button to a movie clip. You name the movie clip `redDotMC`.

Example:

```
on (press) {
     startDrag (redDotMC);
}
on (release) {
     stopDrag ();
}
```

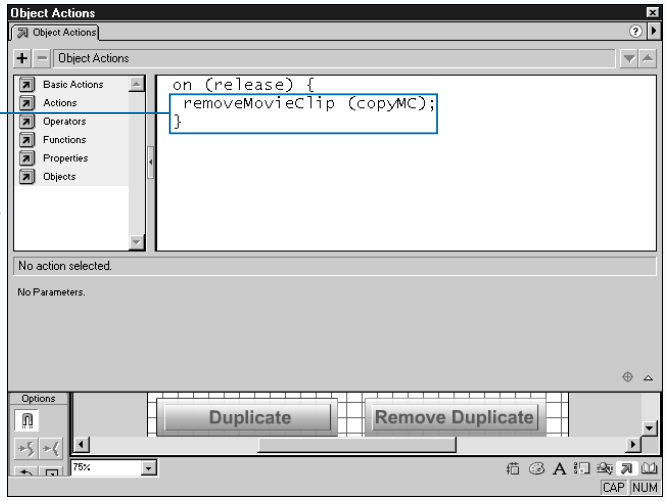
You can use the script that follows to duplicate a movie clip. Assign the following code to frame 1 to initialize `i` to 1. You use `i` to set the depth and to ensure that each instance of the duplicated movie clip has a unique name:

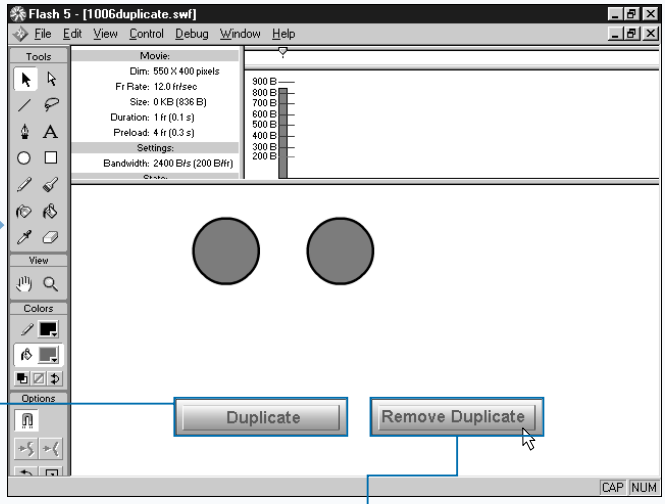
```
i = 1;
```

Assign the following code to a button. The statement `copyMC = "newDot" + i;` concatenates the current value of `i` with `newDot` and assigns the result to the variable `copyMC`. The statement `i = i + 1` increments the value of `i` by 1. This ensures that each instance of the duplicated movie clip has a unique name. You use the `set` property action to place the duplicate next to the original. Unless the user drags the duplicate to a new location, each new copy will be stacked on the previous copy.

Example:

```
on (release) {
     copyMC = "newDot" + i;
     duplicateMovieClip (redDotMC, copyMC, i);
     setProperty (copyMC, _x, redDotMC._x +100);
     i = i + 1;
}
```





6 Remove the movie clip.

7 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

8 Click the button to duplicate your movie clip.

■ A duplicate of your movie clip will be created.

9 Click the button to remove the duplicate movie clip.

■ The duplicate of your movie clip will be removed.

CREATE SMART CLIPS

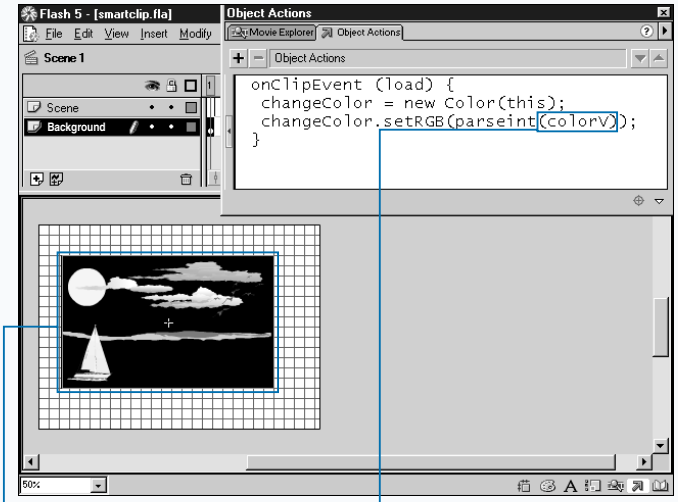
A smart clip is ActionScript that designers can reprogram without using the Action panel. Smart clips enable you to create ActionScript that people with no programming ability can modify. You define a list of options that affect the movie clip look or behavior. Designers select from those options.

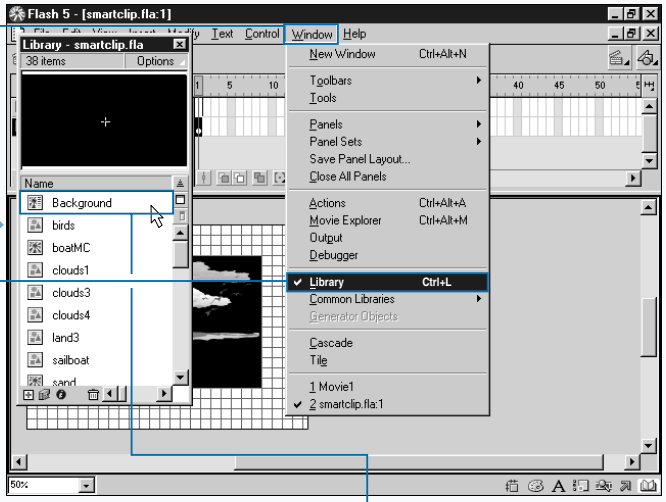
You start the creation of a smart clip by creating a movie clip. Include any values you want the smart clip to pass to the movie clip as variables in the movie clip.

You define smart clips in the Define Clip Parameters dialog box, which you can find by selecting Define Clip Parameters from the Options menu of the Library. In the Define Clip Parameters dialog box, you use the + button to add new parameters and values. You use the − button to remove parameters and values. You use the ▼ and ▲ buttons to change the order of parameters.

The Define Clip Parameters dialog box enables you to add the values from which the designer can choose. You specify a value type of Default, Array ,List, or Object. Default enables you to default the value the designer can select to a string or number. Array enables you to provide the designer with a list of values from which he can choose. The designer can add or remove values from the list. List enables you to create a list of values that the designer can choose from but cannot modify. The Object type enables you to declare several related elements and specify the names and values. For example, you can use the Object type to specify the elements in an array.

CREATE SMART CLIPS





1 Create a movie clip.

Note: This example uses file smartclip.fla, which you can find on the CD that accompanies this book.

2 Include variables that the user can set in your script.

Note: A parameter is a variable.

3 Click Window ➪ Library to open the Library panel.

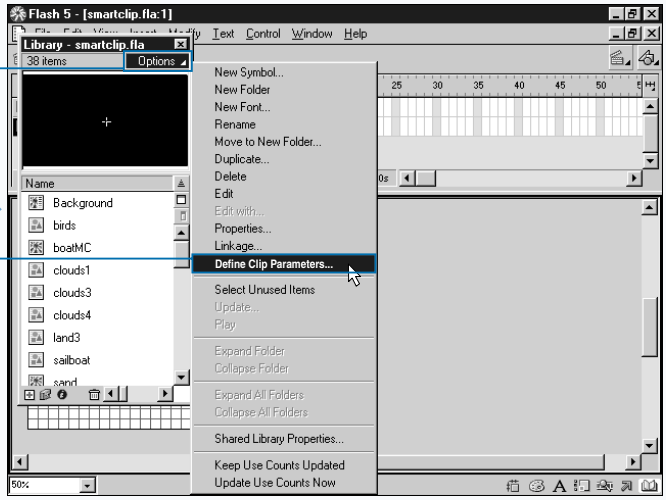
4 Click to select the movie clip in the Library.

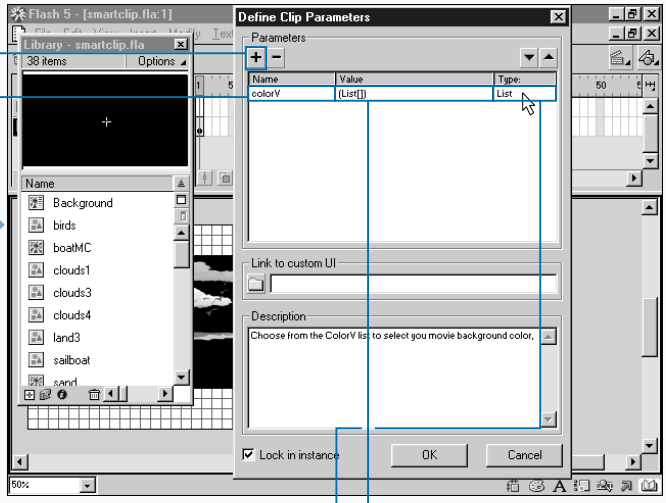
Extra

When you place a movie clip in the Library, it is available only to that movie. You can use the common library to make a movie clip available to all movies. Because smart clips are reusable code, it is a good idea to place them in the common library.

To add a file to the common library, copy the file to the Libraries folder under the Flash application folder. To use the file, click Window ➪ Common Libraries and select the file from the submenu. Drag the file from the common library onto the Stage.

You use smart clip parameters to change the behavior or appearance of a movie clip. Smart clip parameters are passed to the movie clip when the movie clip loads. Smart clips display a unique icon in the Library window.





5 Click Options ➪ Define Clip Parameters to open the Define Clip Parameters dialog box.

6 Click ⊞ to add a parameter.

7 Double-click here and type a variable name.

8 Double-click Value to open the Values panel.

9 Double-click here and select a type.

CONTINUED

CREATE SMART CLIPS (CONTINUED)

You enter notes in the Description field of the Clip Parameters panel. You can enter any information you want in this field. The Clip Parameters panel has a Lock in Instance box. The Lock in Instance field prevents the user from renaming parameters. It is a good idea to check this box.

A smart clip parameter is a variable. When you create or modify your movie, you define the variables. The designer uses the Clip Parameters panel to select parameter values. Selecting a parameter value assigns a value to the variable.

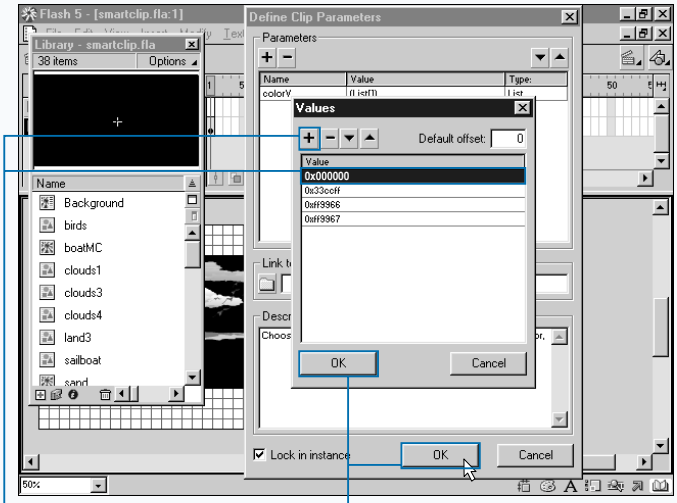
You can use smart clips for any repetitive programming tasks. For example, you can create menus, buttons, list boxes, logos, games, or other types of movie clips and use smart clips to change or modify them.

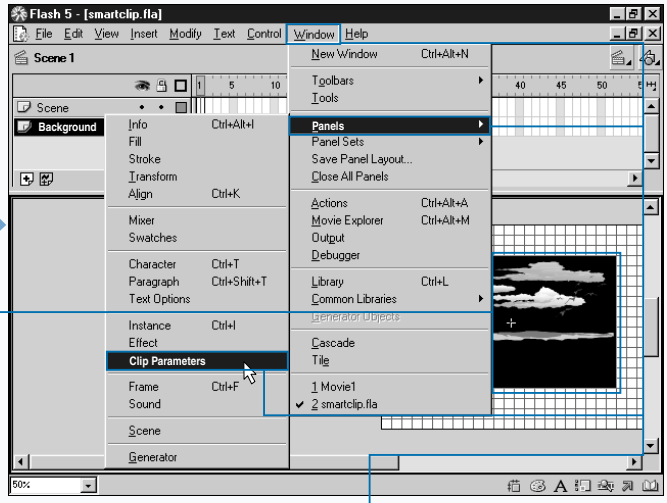
Let us say you have a movie clip. Sometimes the designer wants the movie clip to play with a blue background. At other times, he wants the movie clip to play with an orange or black background. You can create a smart clip that allows the designer to select the hexadecimal code for the background color he wants to use.

You start by creating ActionScript that will change the background color, including a variable that defines the color. Enter the name of the variable that defines the color in the Name field of the Define Clip Parameters box. Use the Value field to list the possible values.

When the designer opens the Clip Parameters panel, he will be able to select the hexadecimal code he wants to use from the list you provide.

CREATE A SMART CLIP (CONTINUED)





10 Click ⊞ to add a value.

11 Double-click here and type a value.

12 Click OK.

13 Click OK.

14 Select the movie clip.

■ Select the black background.

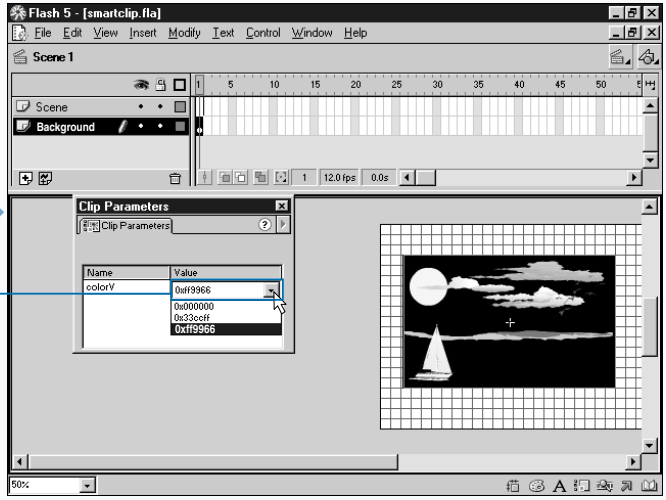
15 Click Window ➪ Panels ➪ Clip Parameters to open the Clip Parameters panel.

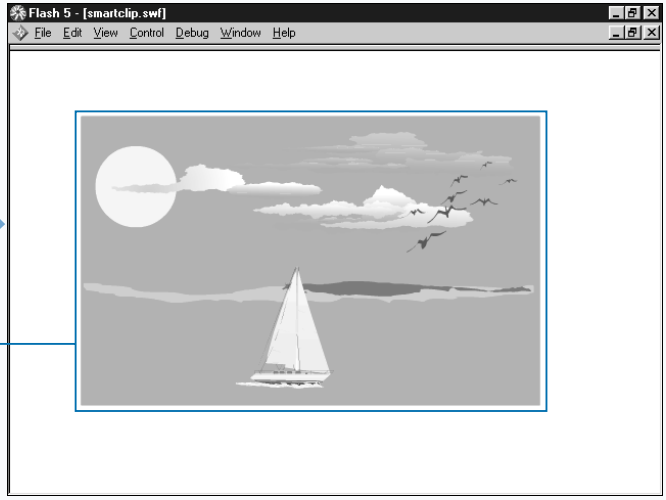
Extra

A smart clip custom interface enables you to pass values to a movie clip using a movie instead of the Clip Parameters panel. If you create a custom interface, when the designer selects Window ➪ Panels ➪ Clip Parameters from the menu, Flash presents the custom interface instead on the Clip Parameters panel. The designer makes selections from the custom interface.

It is a good practice to save the custom interface SWF file and the FLA for the smart clip that uses it in the same directory. If you use your smart clip in multiple files, the SWF file for the custom interface and the FLA for the smart clip should always be in the same relative location.

If you place your custom smart clip in the common library, you must copy the SWF to the Libraries folder with the same relative path that you specified in the Define Clip Parameters dialog box.





16 Click ▾ to select a value.

■ You can use your smart clip to select a background color for your movie.

17 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

18 Your movie plays automatically when you enter the test environment.

■ The background of the movie appears in the color you selected in the Clip Parameters panel.

CREATE A SMART CLIP CUSTOM INTERFACE

A smart clip enables you to create ActionScript that a designer can reprogram without using the Actions panel. The designer uses the Clip Parameters panel to select from a list of options you set up. You can create a custom interface for a smart clip. A custom interface enables you to pass values to a movie clip using a movie instead of the Clip Parameters panel. Using a custom interface, you can create smart clips that are user friendly and aesthetically pleasing.

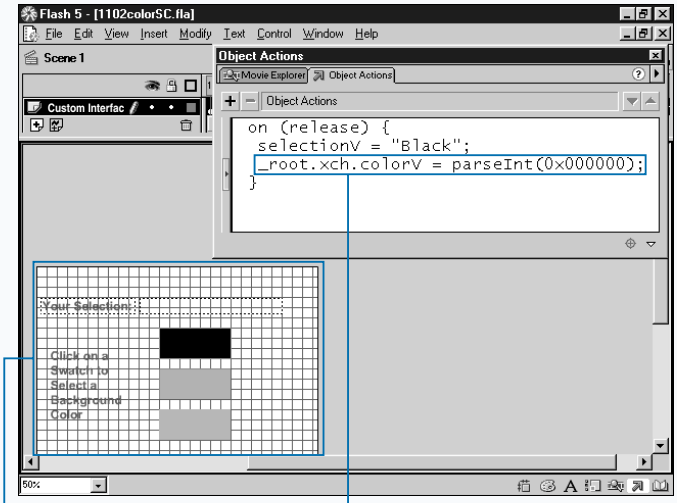
You start by creating a movie that will act as the interface. The movie should provide the user with options that can be used to select the values the user wants to set.

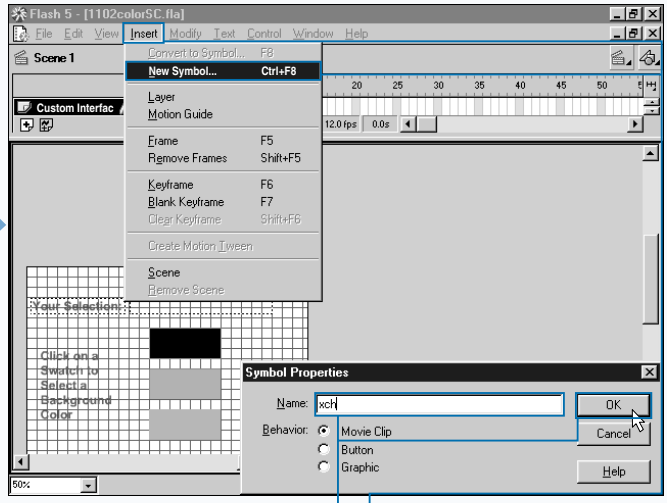
You pass the values to the smart clip using an intermediary, or exchange, movie clip. An intermediary movie clip contains nothing. It is completely blank. You do not place any graphics, buttons, movie clips, or ActionScript in an intermediary movie clip.

You use the Symbol Properties dialog box to create an intermediary movie clip. Simply specify a Name in the name field and select movie clip as the behavior.

Intermediary movie clips have an instance name of `xch`. You must place an instance of the exchange movie clip on the main Timeline of the custom interface movie. Create a layer and name the layer Exchange Clip. See Chapter 1 for more information on creating layers. Place the intermediary movie clip in frame 1 of the Exchange Clip layer. The intermediary movie clip appears on the Stage as a small dot. Use the Instance panel to name the movie clip instance `xch`. The `xch` movie clip instance must always be loaded. You can pass arrays and objects through the `xch` movie clip, but you cannot pass nested arrays or objects.

CREATE A SMART CLIP CUSTOM INTERFACE





1 Create a movie to use as a custom interface.

Note: This example uses file colorsc.fla, which you can find on the CD that accompanies this book.

2 Add statements that pass parameters to the **xch** movie clip.

■ You must create the intermediary movie clip with an instance name of **xch**. Do not place any graphics, buttons, movie clips, or ActionScript in the intermediary movie clip.

3 Click Insert ➪ New Symbol to open the Symbol Properties dialog box.

4 Type a name.

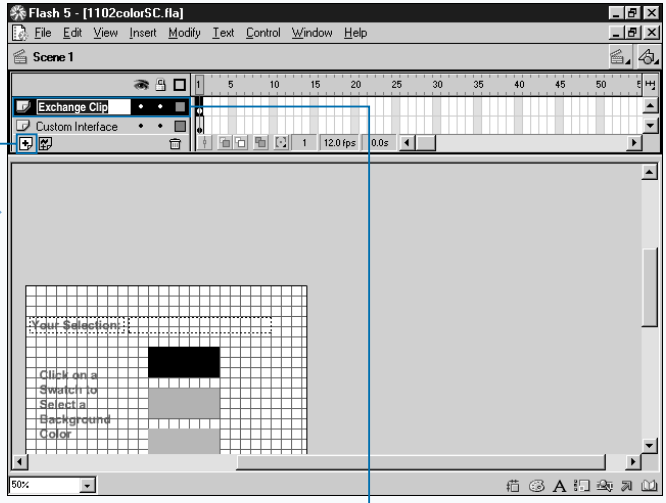
5 Click Movie Clip to select it (○ changes to ◉).

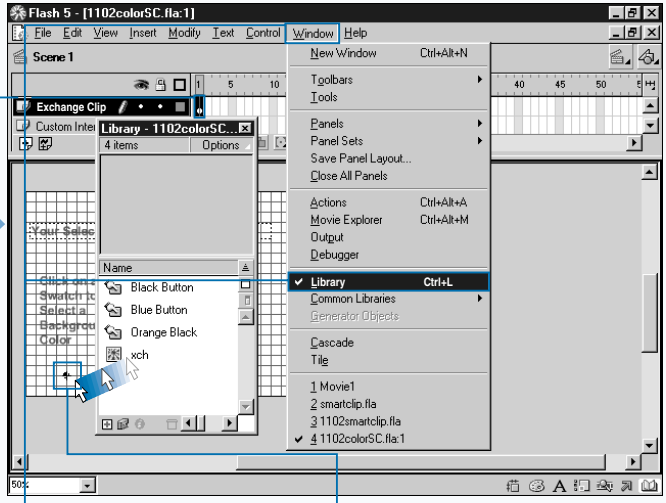
6 Click OK.

Extra

Flash ships with smart clips that you can use to create interactive learning applications. To find these smart clips, click Window ➪ Common Libraries ➪ Learning Interactions. The Learning Interactions smart clips enable you to create true/false, multiple choice, fill-in-the-blank, drag-and-drop, hot spot, and hot object learning applications.

With the true/false smart clip, you can create applications with an either/or option. The multiple-choice smart clip enables you to create applications that allow the user to select from a list of options. The fill-in-the-blank smart clip enables you to create applications that allow the user to type in one or more responses. You can use the drag-and-drop smart clip to create applications that require the user to drag objects to a defined area. Use the hot-spot smart clip when you want the user to select one or more of the defined areas. Use hot object when you want the user to select one or more predefined objects.





7 Click to create a new layer.

8 Double-click and type to name the layer Exchange Clip.

Note: See page 4 for more information on layers.

9 Click Window ➪ Library to open the Library.

10 Click to select frame 1 of the Exchange Clip layer.

11 Drag the intermediary movie clip from the Library onto the Stage.

12 Select the instance that you just dragged onto the Stage.

CONTINUED

CREATE A SMART CLIP CUSTOM INTERFACE (CONTINUED)

Parameters are variables that are included in statements in your custom interface movie. You use parameters to pass values to the `xch` movie clip. In Normal mode, you can use the set variable action to set the variable. The following is an example:

```
_root.xch.colorV = parseInt(0x000000);
```

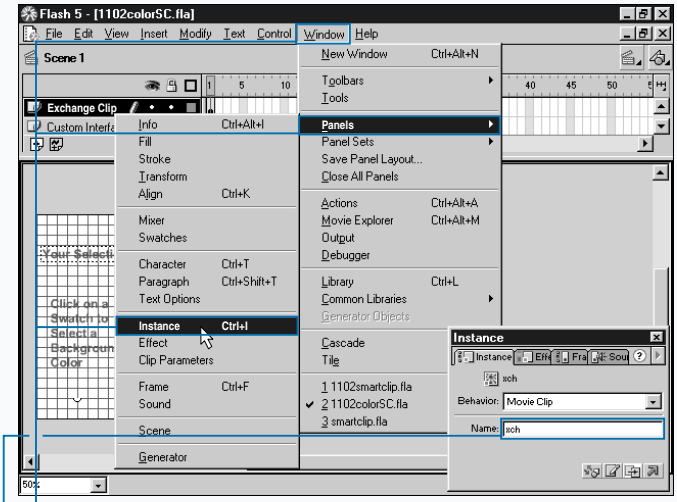
The example sets the value of a variable named `colorV`. Note that the intermediary movie clip is included in the path name. Place your statements that pass parameters in the custom interface movie. When you complete your custom interface movie, publish it as an SWF file.

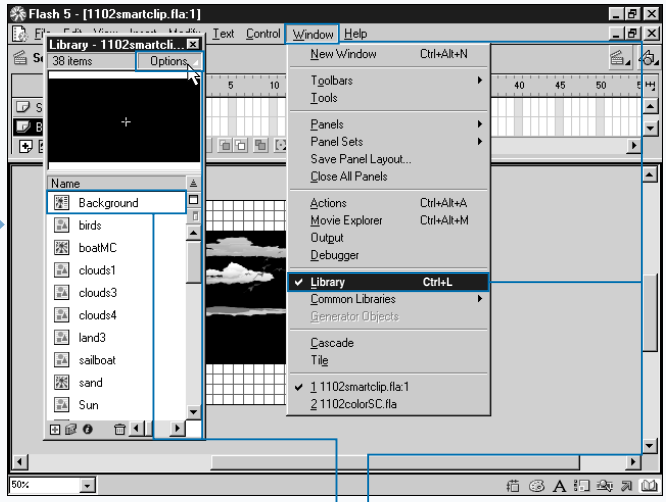
In the movie that uses the custom interface, you link the smart clip to the movie using the Define Clip Parameters dialog box. Place the path to the smart clip in the Link to Custom UI field.

If you want a smart clip that can change the background color of a movie clip, you can create a smart clip that allows the user to select the hexadecimal code for the background color from a list in the Clip Parameters panel. Unfortunately, hexadecimal codes are not user friendly. You can create a custom smart clip that enables the user to click on the background color they want to use.

Start by creating a movie that will act as the custom interface. The movie should include options that allow the user to select a color. Use a variable to pass the selection to the `xch` movie clip. Place the exchange movie clip on the Stage. In the movie that uses the custom smart clip, use the UI field of the Define Clip Parameters dialog box to link to the custom smart clip.

CREATE A SMART CLIP CUSTOM INTERFACE (CONTINUED)





13 Click Window ➪ Panels ➪ Instance to open the Instance panel.

14 Type **xch** to name the instance.

15 Publish your movie as an SWF file.

Note: See page 32 for information on how to publish your movie.

16 Open the movie with which you want to associate the smart clip.

Note: This example uses smartclip2.fla, which you can find on the CD that accompanies this book.

17 Click Window ➪ Library to open the Library.

18 Click to select the movie with which you want to associate the custom movie clip.

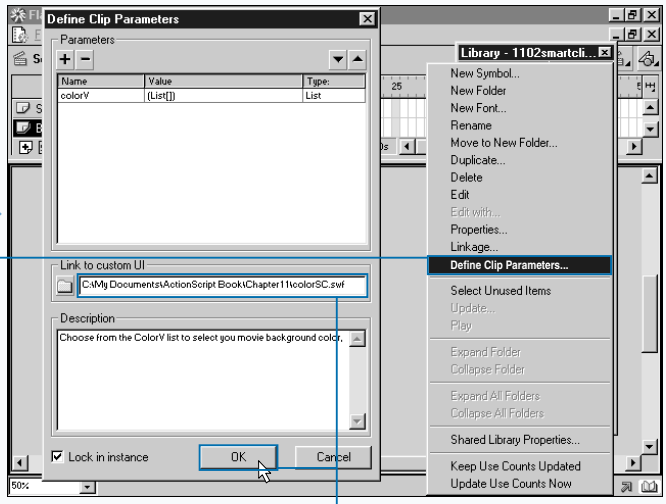
19 Click Options.

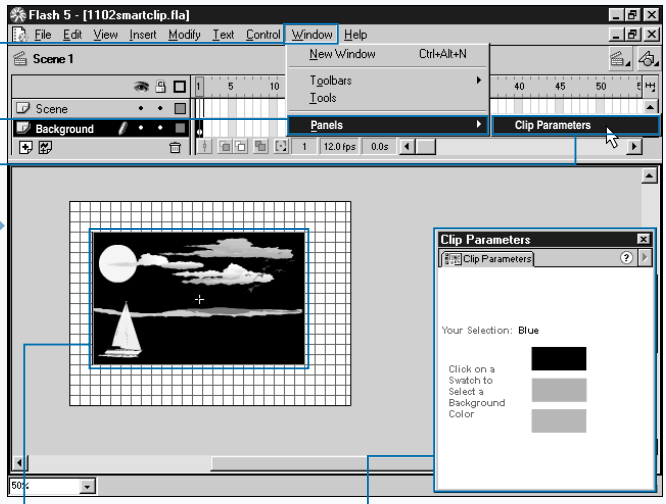
Extra

Learning Interactions includes a feature called Knowledge Track. Knowledge Track allows you to send data to a learning management system. You can use Lotus LearningSpace, or another tracking system. Knowledge Track uses AICC (Aviation Industry CBT Committee) specification version 2.0, an industry standard protocol for courseware-to-tracking system communications. You can set the following values: `interactionID`, `ObjectiveID`, and `Weighting`.

To include Learning Interactions Library smart clips in your movie, select a keyframe. Then click Window ➪ Common Libraries ➪ Learning Interactions and drag a Learning Interactions smart clip onto the Stage.

Then click Window ➪ Panels ➪ Clip Parameters to open the clip parameters panel. If you select TrueFalse, you can use the clip parameters panel to define your question, select the correct answer, and specify the feedback you want to send to the user. A navigation tab enables you to specify how you want Flash to navigate.





20 Click Define Clip Parameters to open the Define Clip Parameters dialog box.

21 Associate the smart clip with the movie.

22 Click OK.

23 Select the movie clip with which you associated the custom smart clip.

24 Click Window ➪ Panels ➪ Clip Parameter to open the Clip Parameters dialog box.

25 Click a color to select an option.

■ When the Clip Parameters panel opens, you can select options using the custom smart clip.

VALIDATE A STRING

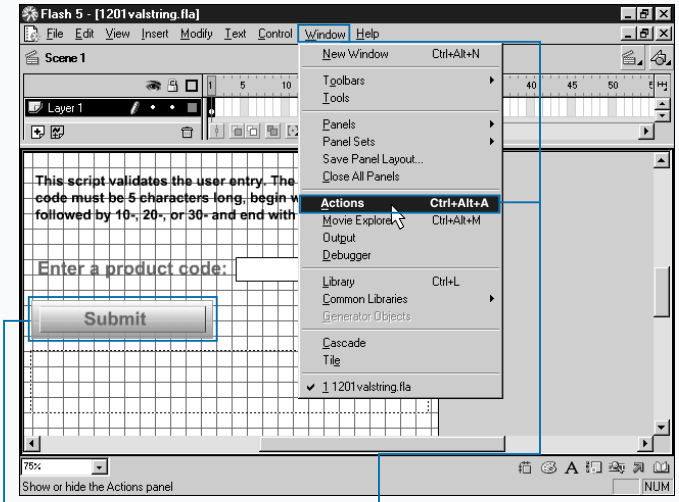
An important part of collecting information from the user is validating the data. Strings may need to be of a specific length or contain specific characters. For example, say that you require a user to enter a product code. In your organization, all product codes begin with the letter p, followed by a 10, 20, or 30, a dash, and a digit. The following is a valid product code: p20-1.

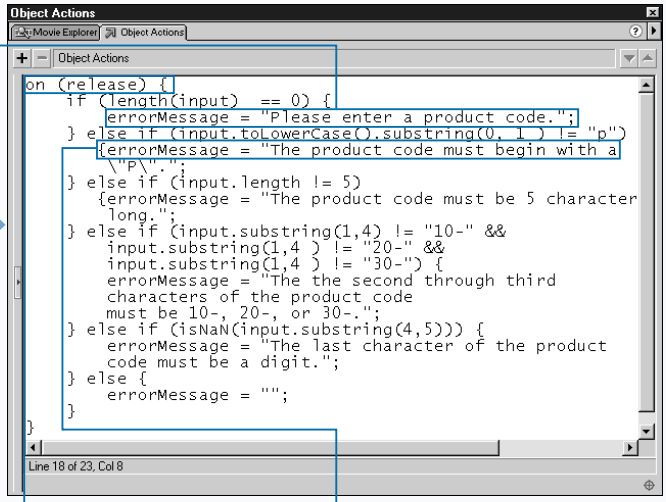
In this situation, if the user submits an invalid product code, you have to contact the user to determine the correct code. You can instead validate user entries at entry to ensure that you do not receive any invalid codes. Many of the string functions are useful when validating data, and you can use them in coordination with one another to validate user entries.

The product code in the preceding example must be exactly five characters long. You can use the `String length` method to determine the length of the string. The first character of the product code must be a `p`. You can use the `substring` method to retrieve the value to determine if the user entered the letter `p`. The next four characters must be `10`, `20`, or `30`, a dash, and a digit. Again, you can use the `substring` method this time used with an `or` to determine if the user entry is valid. You can retrieve the last character of the entry and use the `isNaN` function to determine if it is a number.

After you evaluate whether an entry is valid, if the user enters an invalid entry, you can use a dynamic text box to send a message that tells the user what the valid entries are and instructs the user to enter a valid entry.

VALIDATE A STRING





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file valstring.fla, which you can find on the CD-ROM that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Use the `length` function to determine if the user has made an entry.

4 Check the value of a substring, convert the input to lowercase, and compare it to a lowercase value.

■ This makes the input check insensitive to case.

5 Check the length of the input.

Extra

When validating data, you may need to determine whether the user made an entry. You can make this determination by checking the length of the input field. If the length of the field is `0`, the user has not made an entry.

Example:

```
if (length(input)  == 0) {
     errorMessage = "Please enter a product
code.";
```

If you do not want to take case into consideration when you evaluate an entry made by the user, use the `toLowerCase` or `toUpperCase` function to convert the input value before you make your comparison. The example shown here converts the input to lowercase and then compares the input to a lowercase value.

Example:

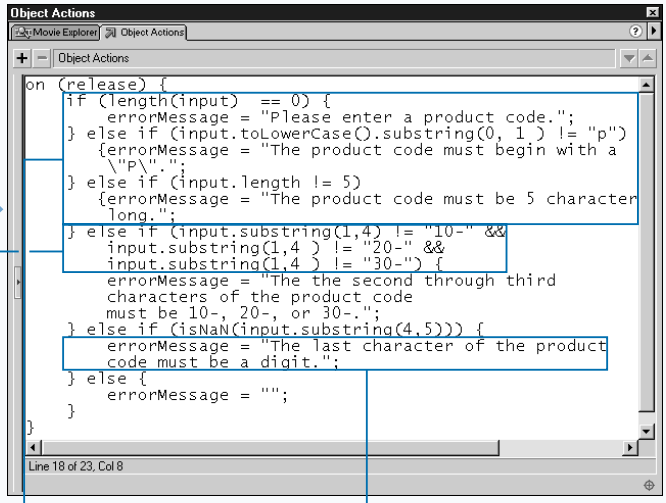
```
if (input.toLowerCase().substring(0, 1 ) != "p")
```

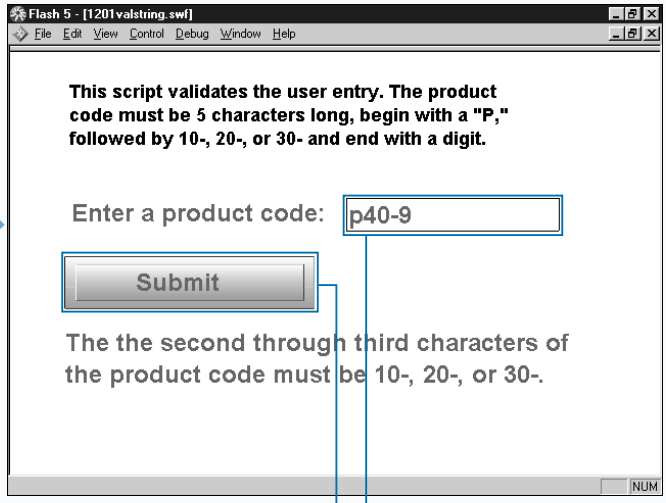
If the user is required to enter a code with a delimiter such as a dash, for example, product code `p10-9`, you can have the user make the entry without the delimiter, and you can use code similar to the following to add the delimiter later.

Example:

```
input = input.substring(0,3) + "-" +
input.substring( 3,4);
```

When validating a list of values, you may want to use a custom function. See Chapter 9 for more information on custom functions.





6 Check a substring to ensure that the input value is in a valid list of values.

7 Check a substring to ensure that the value is a number.

8 Send error messages to the user.

■ Associate the variable with a dynamic text box. This enables you to display the value that the variable returns to the user.

9 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

10 Test your movie by first entering a product code.

11 Click the Submit button.

■ The script validates user entries.

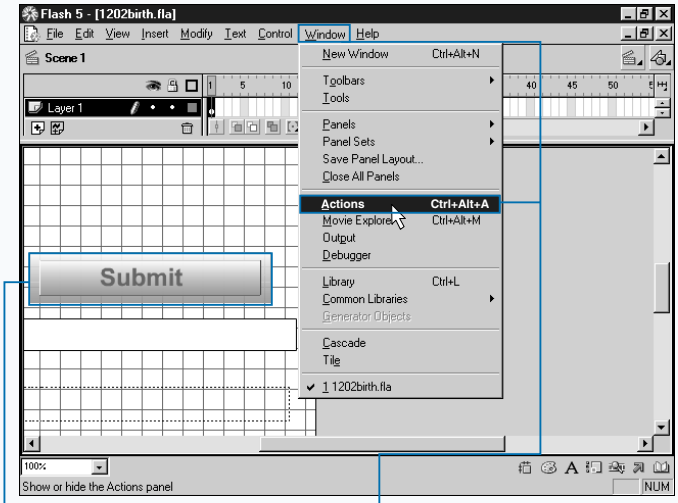
VALIDATE A DATE

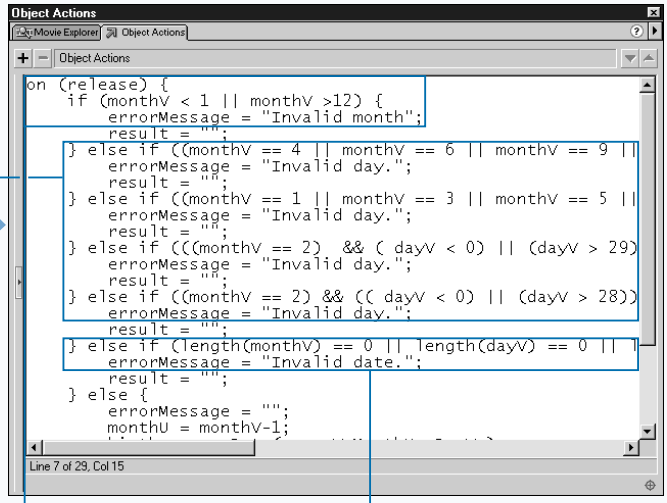
The 12 months of the year are usually numbered from 1 through 12, starting with January. With ActionScript, if you enter `20` as the month, it will accept your entry as a valid entry. It will also accept an invalid number for the number of days in a month, and years are not limited to four digits. You can enter more than the 24 hours in a day, the 60 minutes in an hour, the 60 seconds in a minute, and so on. Therefore, when a user enters a date or time value, you may want to validate the value to ensure that the user has entered valid data.

You want to make sure that the value the user enters for the month is a number between 1 and 12 and that the number of days entered for the months April, June, September, and November do not exceed 30. February is limited to 28 days, and you want to account for Leap Year. The remaining months cannot have more that 31 days. If the user is entering a time, you want to do similar time validations.

These validations require conditional checking. If the month is equal to 1 (January), the day must be greater than 0 but less than 32, but if the month is equal to 4 (April), the day must be greater than 0 but less than 31. It is even more complicated for February; the day cannot be greater than 28 unless the year is a Leap Year. You can use the modulo operator to help you determine whether the year is a Leap Year.

VALIDATE A DATE





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file birth.fla, which you can find on the CD-ROM that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Validate the month.

4 Validate the day.

5 Check whether the user made an entry.

Extra

When validating dates, you must keep the following in mind: A Leap Year occurs whenever the year is evenly divisible by four, unless the year is also divisible by 100. However, if the year is evenly divisible by 100 *and* evenly divisible by 400, it is a Leap Year.

The modulo operator returns the remainder of one number divided by another number. Because you begin determining whether a year is a Leap Year by finding out if the year is evenly divided by four, you should use the modulo function. Your expression will look similar to the one shown here.

Example:

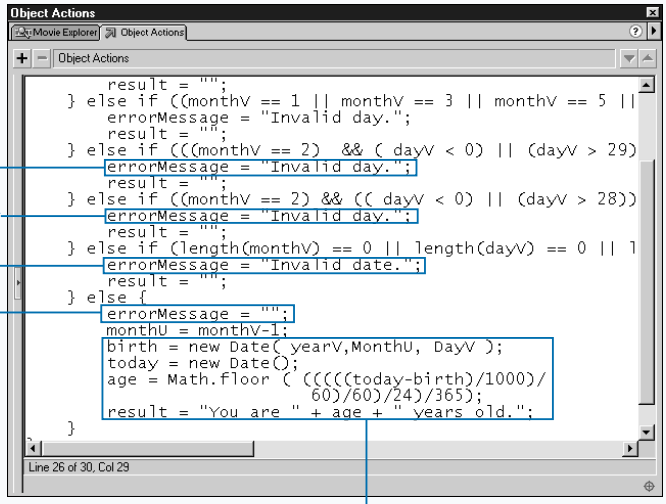
```
year % 4 ==0
```

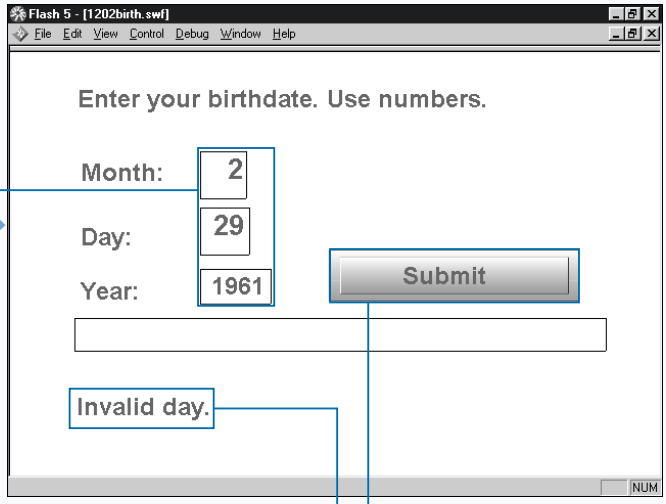
The operator divides the number 4 into the value in the variable `year` and returns the remainder. The expression shown tests to determine if the remainder is equal to 0.

If you do a lot of date checking, you will definitely want to develop custom date-checking functions, such as the following. The function returns `true` if the month argument is between 1 and 12 and `false` if it is not. See Chapter 9 for more information on custom functions.

Example:

```
function validateMonth (month) {
  if (month < 1 || month > 12 ) {
    return false;
  } else {
    return true;
  }
}
```





6 Send error messages.

7 Perform a calculation.

8 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

9 Test your movie by entering your birthday.

10 Click the Submit button.

■ The script will validate your entry before calculating your age.

SEARCH A TEXT BOX

You can use the `Selection.setFocus` method with the `Selection.setSelection` method, `String.indexOf` method, and `length` property to add a search feature to a text box. A search feature enables the user to search a text box for a word or phrase. The user types the word or phrase into an input text box and clicks a button, and ActionScript finds and selects the word or phrase entered, or it sends a message to the user stating the entry cannot be found. You can search a text box from top to bottom or from bottom to top.

The `Selection.setFocus` method sets the focus to a specified text box. You must set the focus before you can use the `Selection.setSelection` method. The syntax for the `Selection.setFocus` method is

```
Selection.setFocus(string)
```

Use the `string` argument to specify a string that names the variable associated with the text box to which you want to set the focus.

The `Selection.setSelection` method selects an area in the text box that currently has focus. The syntax for the `Selection.setSelection` method is

```
Selection.setSelection(start,end);
```

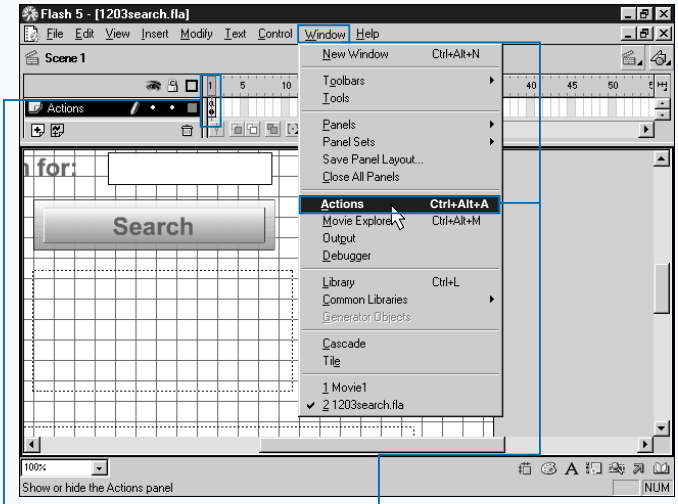
Use the `start` argument to indicate the index position at which you want to start the selection. Use the `end` argument to indicate the index position at which you want to end the selection. The first position in the text box has an index position of `0`.

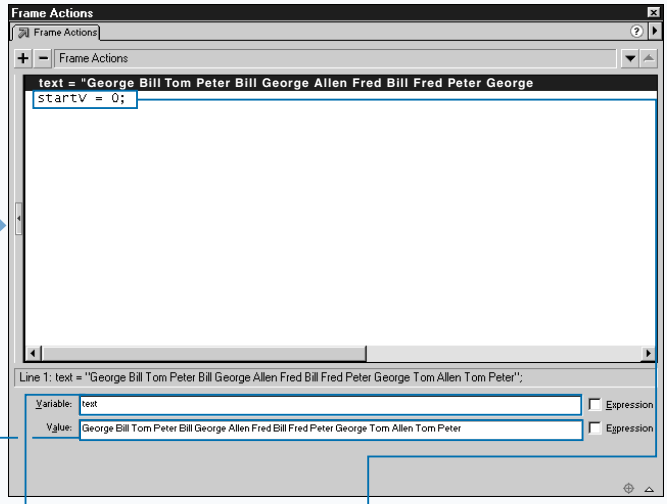
The `String.indexOf` method searches a string for a specified value and returns the index position for that value. The syntax for the `String.indexOf` method is

```
string.indexOf(value, start)
```

Use the `string` argument to specify the string you want to search. Use the `value` argument to specify the value for which you want to search. Use the `start` argument to specify the index position at which you want your search to begin. If the `String.indexOf` method does not find the value specified, it returns `-1`.

SEARCH A TEXT BOX





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file search.fla, which you can find on the CD-ROM that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Assign text to a variable.

4 Associate the variable with a text box.

5 Initialize the variable.

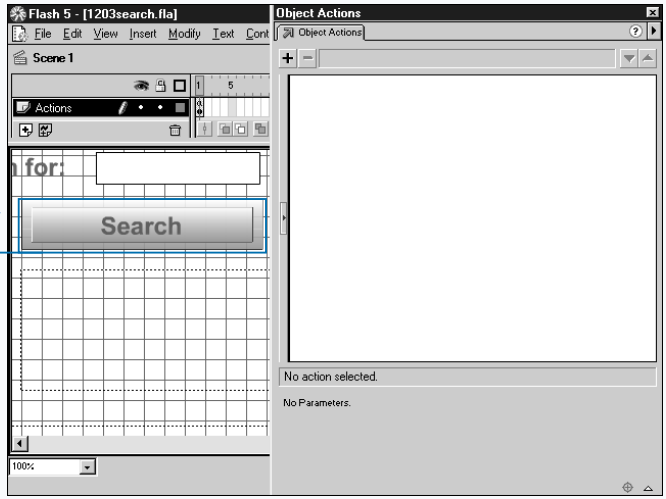
■ This example uses the `startV` variable to specify where the search should start. The variable `StartV` is used as the `indexOf` start argument.

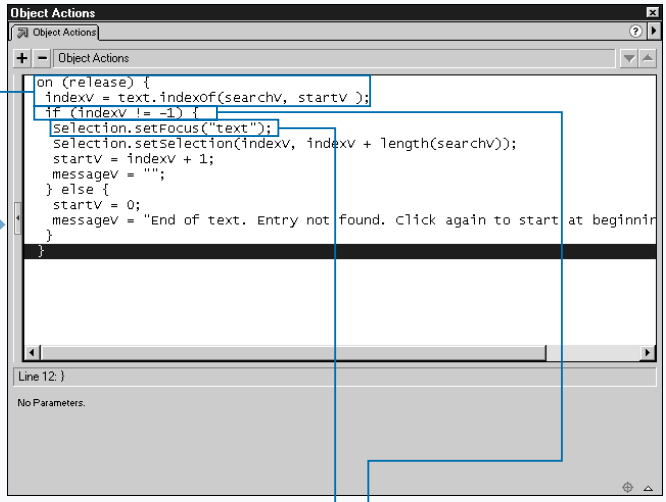
Apply It

If you do not want your search to be case sensitive, use a script similar to the one shown here. Apply the `String.toLowerCase` methods to the value for which you are searching and to the text you are searching.

Example:

```
on (release) {
   valueV =
text.toLowerCase().indexOf(searchV.toLowerCase(),
startV ) ;
   if (ValueV != -1) {
      Selection.setFocus("text");
      Selection.setSelection(valueV,valueV +
length(searchV.toLowerCase()));
      startV = valueV + 1;
      messageV = "";
   } else {
      startV = 0;
      messageV = "End of text. Entry not found. Click
again to start at beginning.";
   }
}
```





6 Select the frame, button, or movie clip to which you want to add ActionScript.

7 Get the `indexOf` search value and associate the variable with an input text box.

■ Use `searchv` to get the value for which the user is searching.

8 Check to determine if the value of `searchv` is found.

9 If ActionScript finds the search value, set the focus.

CONTINUED

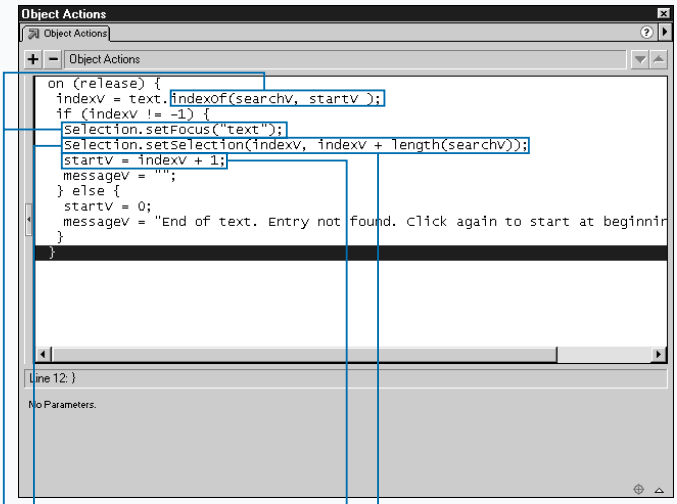
SEARCH A TEXT BOX (CONTINUED)

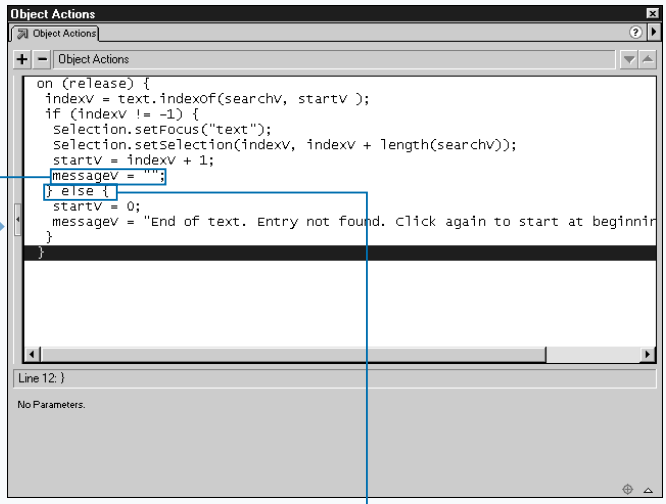
If you use the `String.indexOf` method as the `start` argument for the `Selection.setSelection` method, ActionScript will find the value that you specify and start a selection beginning where the value you are searching for begins. If you set the `end` argument to the `start` position plus the length of the value for which you are searching, ActionScript will select the value for which you are searching.

You can use an `if` statement to determine whether the search will find a value. If the search will find a value, execute the search. Start the next search at the value returned by `indexOf` +1. This causes your script to continue searching the text box until it does not find the value. If the search does not return a value, you can send a message to the user stating value not found. You then set the `indexOf` start value to `0`. This will cause the next search to begin at the beginning of the text box.

The search is case sensitive. If you type bill, the search will not find `Bill`. If you do not want your search to be case sensitive, either apply the `String.toLowerCase` or `String.toUpperCase` methods to the value for which you are searching and to the text you are searching. This forces the comparison of two values of the same case, regardless of what the user enters.

SEARCH A TEXT BOX (CONTINUED)





10 Define the selection criteria.

11 Use `IndexV` as the start argument for `Selection.setSelection`.

12 Use `indexV` plus the length of the search criteria as the end argument for `Selection.setSelection`.

13 Add 1 to the `indexV` value.

■ This causes the next search to begin one character to the right of the last value found.

14 Clear any messages previously issued.

15 If ActionScript cannot find the search value, reset the variable.

■ This causes `startV` to begin the next search at the top of the text box.

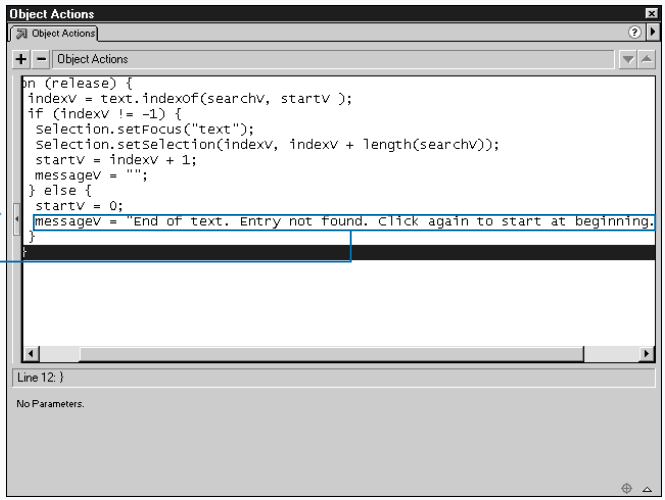
Extra

The `String.lastIndexOf` method searches a string for a specified value and returns the index position for the last occurrence of that value. You can use the `lastIndexOf` method to search a text box starting from the bottom, as shown in the following.

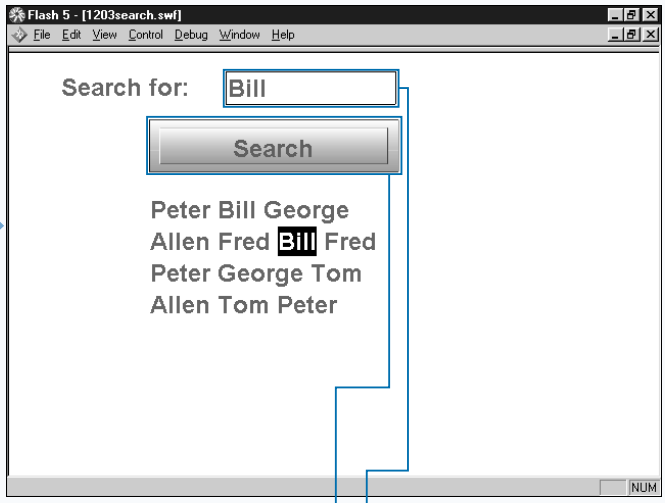
Example:

```
on (release) {
   if (text.lastIndexOf(searchV, startV ) != -1)
{
      Selection.setFocus("text");

Selection.setSelection(text.lastIndexOf(search
V, startV ),text.lastIndexOf(searchV, startV ) +
length(searchV));
      startV = text.lastIndexOf(searchV, startV )
- 1;
      messageV = "";
   } else {
      startV = 0;
      startV = length(text);
      messageV = "Begining of text. Entry not
found. Click button to start again.";
   }
}
```



16 Send a message to the user.



17 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

18 Test your movie by first entering a name.

19 Click the Search button.

■ ActionScript searches for the value you entered.

CONVERT SYMBOLS

Buttons have several handlers that you can use to trigger actions. Movie clips also have handlers that you can use to trigger actions. However, there are times when you need to use both movie clip and button handlers. In such cases, you must convert a button to a movie clip or vice versa.

Creating a button that executes a continuous action is a good example of when you need to convert a button to a movie clip. If you want to create a button that executes a continuous action, such as rotating an object when the user clicks the object and continuing to rotate the object as long as the user holds down the mouse, you need to use the button `on(press)` and `on(release)` handlers and the movie clip `onClipEvent(enterFrame)` handler.

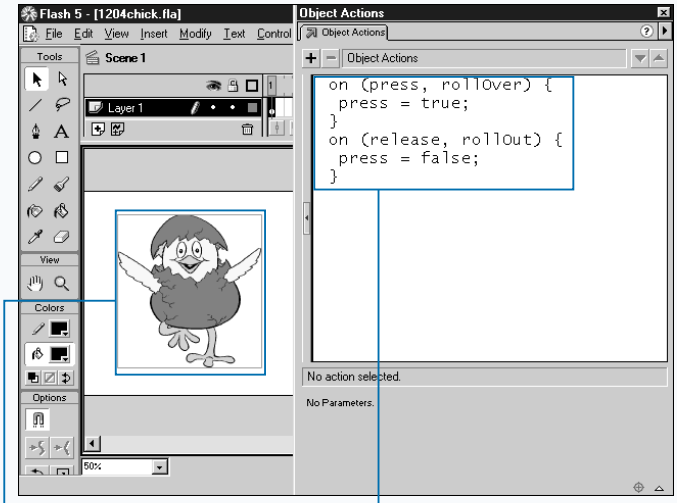
The `on(press)` handler triggers actions when the user clicks the object. The `on(release)` handler triggers actions when the user releases the mouse. You use `on(press)` to start a continuous action and `on(release)` to stop a continuous action.

A handy technique for creating continuous play actions is to create a one-frame movie or movie clip with the `onClipEvent(enterFrame)` handler. When you use the `onClipEvent(enterFrame)` handler with a one-frame movie, the actions in the frame execute continuously, thereby creating a continuous action.

To use these actions in combination with one another, you must create the button that starts and stops the action, and then you must convert the button to a movie clip. You use the Symbol Properties dialog box to convert a button to a movie clip.

Creating an object that users can click and drag is another common example of when you need to convert a button into a movie clip. For more information on creating a draggable object, see Chapter 2.

CONVERT SYMBOLS

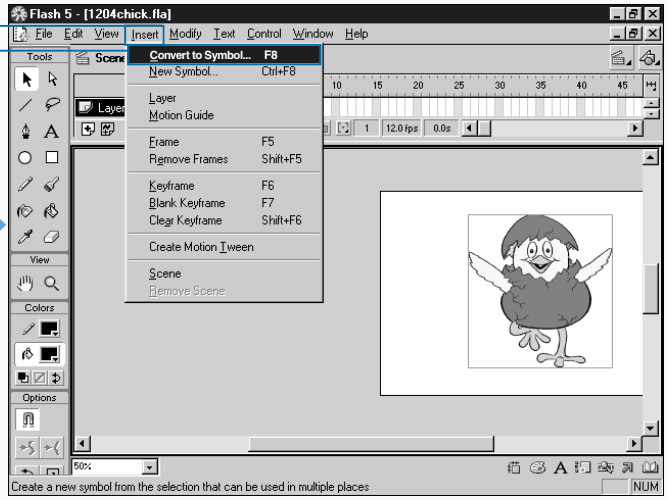


1 Create a button.

Note: This example uses file chick.fla, which you can find on the CD-ROM that accompanies this book.

2 Add ActionScript.

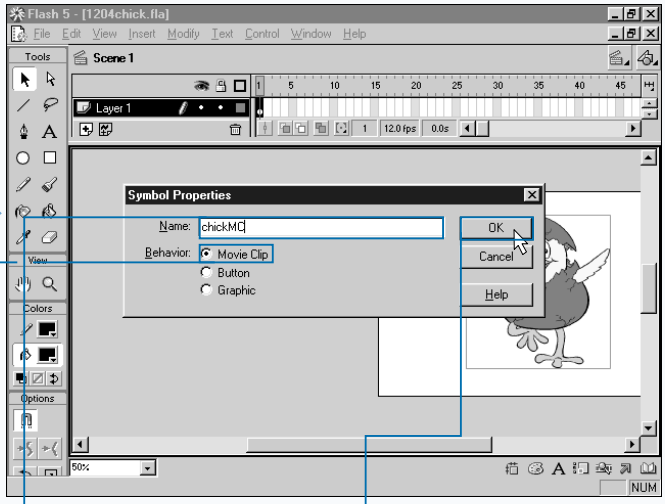
Note: See page 12 for information on creating a button.

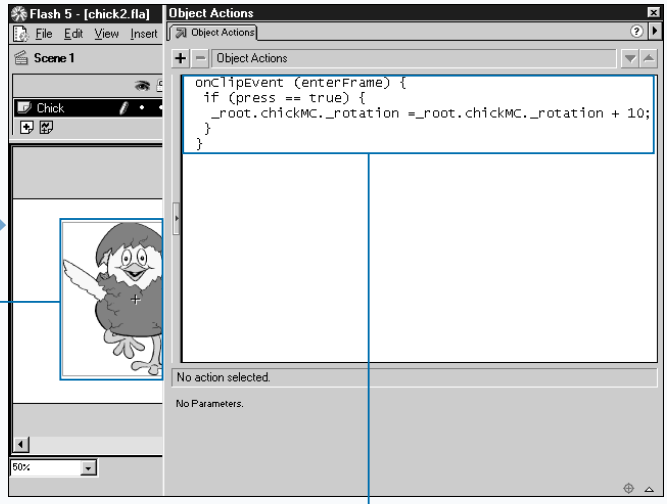


3 Click Insert ➪ Convert to Symbol to open the Symbol Properties dialog box.

Extra

After you convert a button to a movie clip, you must select the movie clip, click Edit ➪ Edit Symbols, and open the Actions panels to edit the script associated with the button. After editing the script, click Edit ➪ Edit Symbols to return to the main Timeline. You can also edit the script by double-clicking the movie clip and then opening the Actions panel.





4 Type a symbol name.

5 Click Movie Clip.

6 Click OK.

■ The button is now a movie clip.

7 Select the movie clip.

8 Add ActionScript.

CREATE ROTATION EFFECTS

It is obvious that you can use movie clip properties to create special effects. What may not be as obvious is how easy it is to create certain effects. For example, you can use the `_rotation` property to set the rotation value of an object. When you place an object on the Stage, it has a rotation value of `0`. To rotate the object clockwise, increase the value. To rotate the object counterclockwise, decrease the value.

You can create many interesting effects using the expression `_rotation = _rotation + x`, where `x` is equal to the speed at which you want the object to rotate. The higher the value of `x`, the faster the object will rotate. Negative `x` values rotate the object counterclockwise.

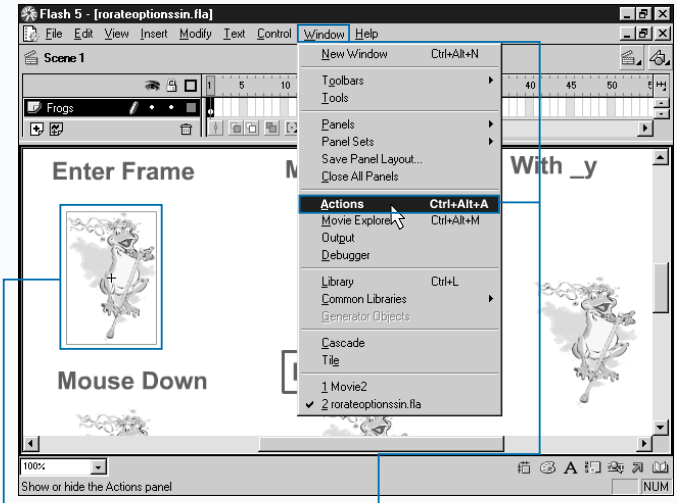
Use `_rotation = _rotation + x` with `onClipEvent(enterFrame)` to create an object that rotates continually. Use it with `mouse move` to rotate the object when you move your mouse. Use it with `mouse down` to rotate the object when you click the mouse. In fact, you should try it with each of the movie clip and button states.

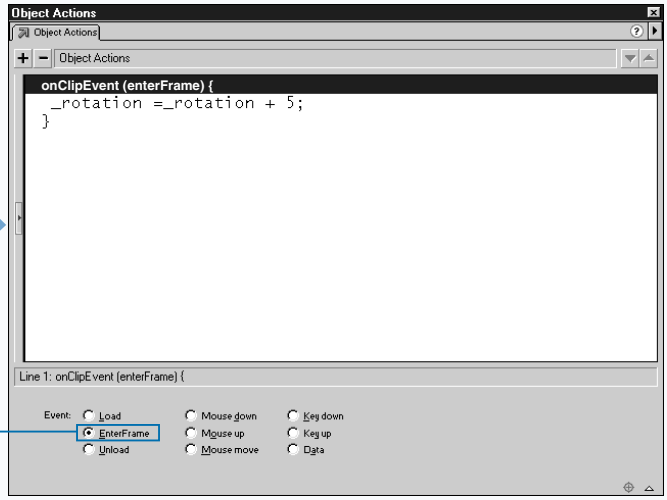
If you add some additional script to the `_rotation = _rotation + x` expression, you can create an object that rotates when you roll over it and stops rotating when you release the mouse.

If you use the `_rotation = _rotation + x` expression along with the `_x` property or the `_y` property, you can rotate an object and move it at the same time. However, you will have to add some additional code to limit the area in which the object can move, or your object may move off the Stage. You can also use the expression `_rotation = _xmouse` to create an interesting effect.

Keep in mind that a rotating object rotates around its registration point. You can change the registration point to change the rotation angle.

CREATE ROTATION EFFECTS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file rotate.fla, which you can find on the CD-ROM that accompanies this book.

2 Click Window ➪ Actions to open the Actions panel.

3 Select the event handler.

Extra

To make a button rotate continuously, you can attach the script shown here to it. When you roll over the object, it rotates continuously until you roll out.

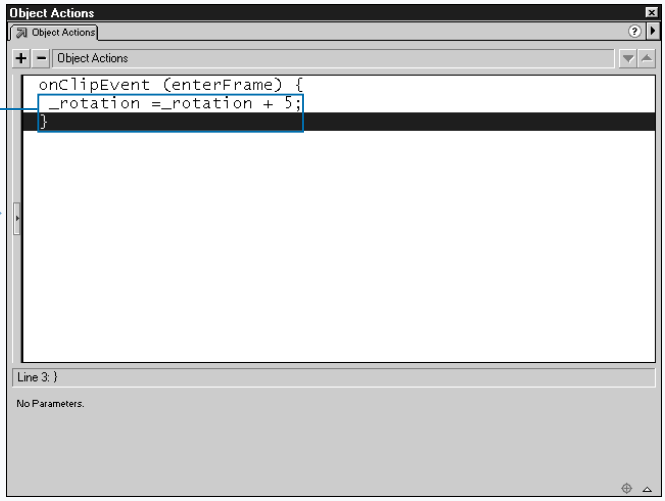
Example:

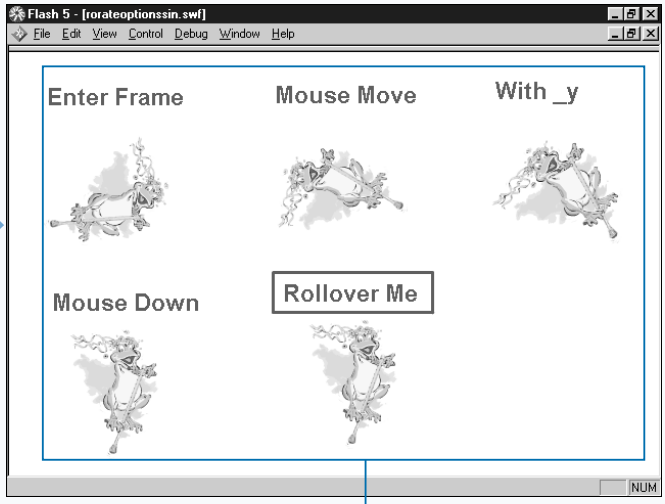
```
on (rollOver) {
   press = true;
}
on (rollOut) {
   press = false;
}
```

Then convert the button to a movie clip and attach the script shown here to it. The variable text is the name of the text box. In this example, `frogMC` is the movie clip name.

Example:

```
onClipEvent (enterFrame) {
   if (press == true) {
      _root.frogMC._rotation =_root.frogMC._rotation + 10;
   }
}
```





4 Set the rotation.

5 Repeat steps 3 and 4 for additional handlers.

6 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

7 Test your movie by clicking the mouse, moving the mouse, and rolling the cursor over the button to test the mouse handlers.

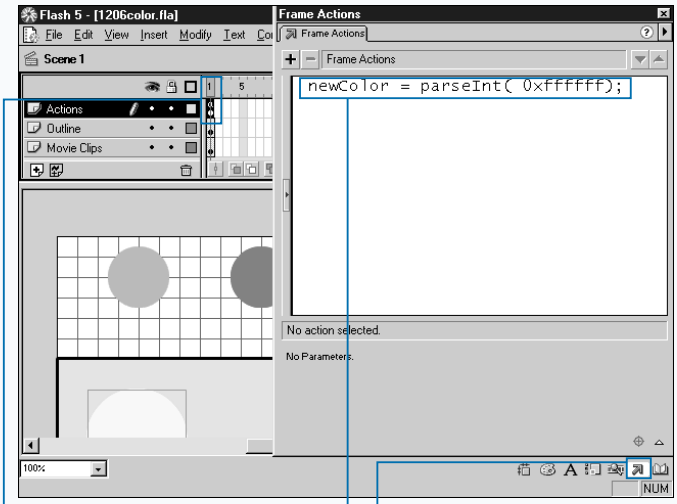
CREATE A COLORING BOOK

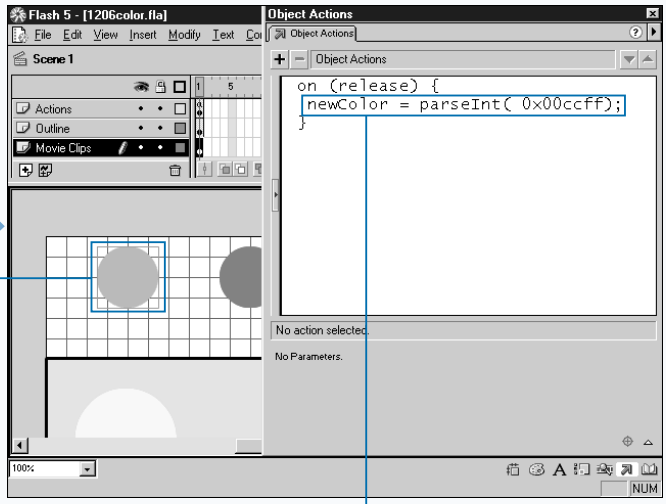
Using the `Color` object, you can create an electronic coloring book. There are many different techniques that you can use. You can place color selections on the Stage; when the user clicks a color, ActionScript assigns the selection to a variable. When the user clicks an object, ActionScript applies the color. Alternatively, you can create a color slider and have the user create the color and then apply the color to objects. You can also code color changes into the object and have the object change color each time the user clicks the mouse.

If you use the color selection method, start creating your coloring book by creating buttons to represent each possible color selection. Make the button the same color as the color that it represents. Store the hexadecimal color code to a variable when the user releases the mouse after clicking the button. Use the `parseInt` function to tell ActionScript that the number you are assigning the variable is a hexadecimal number. For example, in `newColor = parseInt( 0x00ccff);` any value that starts with `0x` is read by the `parseInt` function as a hexadecimal number.

Assigning a color when the user clicks an object is tricky. The `new Color` method requires you to enter a target. In order for you to be able to assign a color when an object is clicked, the object must be a button. You cannot target buttons. You must create the button, add the script that makes the color change, convert the instance to a movie clip, and assign the movie clip the name that you specified when you created the button. For more information on converting a button to a movie clip, see the section "Convert Symbols."

CREATE A COLORING BOOK





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file color.fla, which you can find on the CD-ROM that accompanies this book.

2 Click to open the Actions panel.

3 Assign a color to a color variable.

Note: This step initializes the color. ActionScript applies this color if the user does not select a color.

4 Select the frame, button, or movie clip to which you want to add ActionScript.

5 Assign the color of the button to a variable.

Apply it

You can have each click of the mouse change the color of an object. Start by creating a color object and assigning it a color. Place the following script in frame 1.

Frame 1

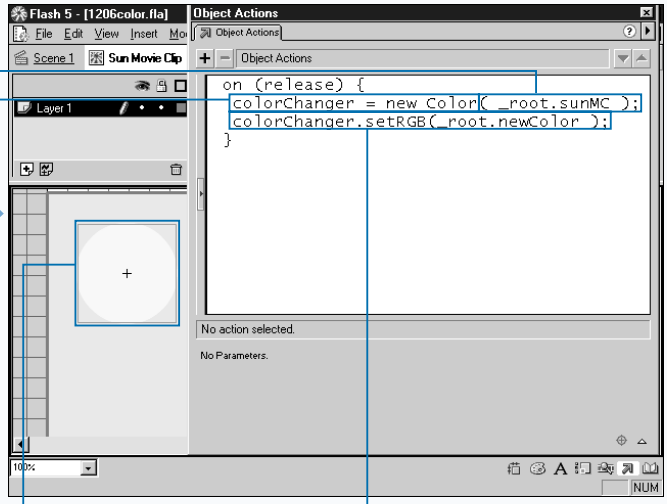
```
colorChanger = new Color( _root.circleMC );
colorChanger.setRGB(0xff0000);
```

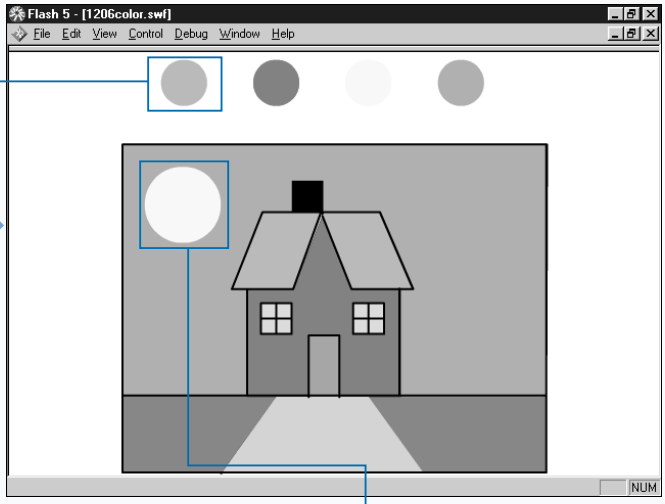
Then create a button and assign the following script.

Button

```
on (release) {
   if (colorChanger.getRGB() == parseInt(0xff0000)) {
      colorChanger.setRGB(0x0000FF);
   } else if (colorChanger.getRGB() == parseInt(0x0000ff)) {
      colorChanger.setRGB(0xFFFF00);
   } else if (colorChanger.getRGB() == parseInt(0xffff00)) {
      colorChanger.setRGB(0x009900);
   } else if (colorChanger.getRGB() == parseInt(0x009900)) {
      colorChanger.setRGB(0xff0000);
   }
}
```

Convert the button to a movie clip. Give the movie clip the name that you targeted in frame 1. In this example, the name is `circleMC`.





6 Select the frame, button, or movie clip to which you want to add ActionScript.

7 Create a `Color` object.

8 Target a movie clip.

9 Use the variable that you created to assign a color to the `Color` object.

10 Convert the button to a movie clip.

11 Name the movie clip, using the name that you specified in step 8.

12 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

13 Test your movie by first clicking the color palette.

14 Click the object.

■ The script enables the user to click a color palette to set a color and then make an object that color.

EMULATE PANELS

Many programs use panels to provide users with options and tools that enable them to perform tasks. Panels appear on-screen at user request and are draggable, and the user can hide them from view when they are not needed.

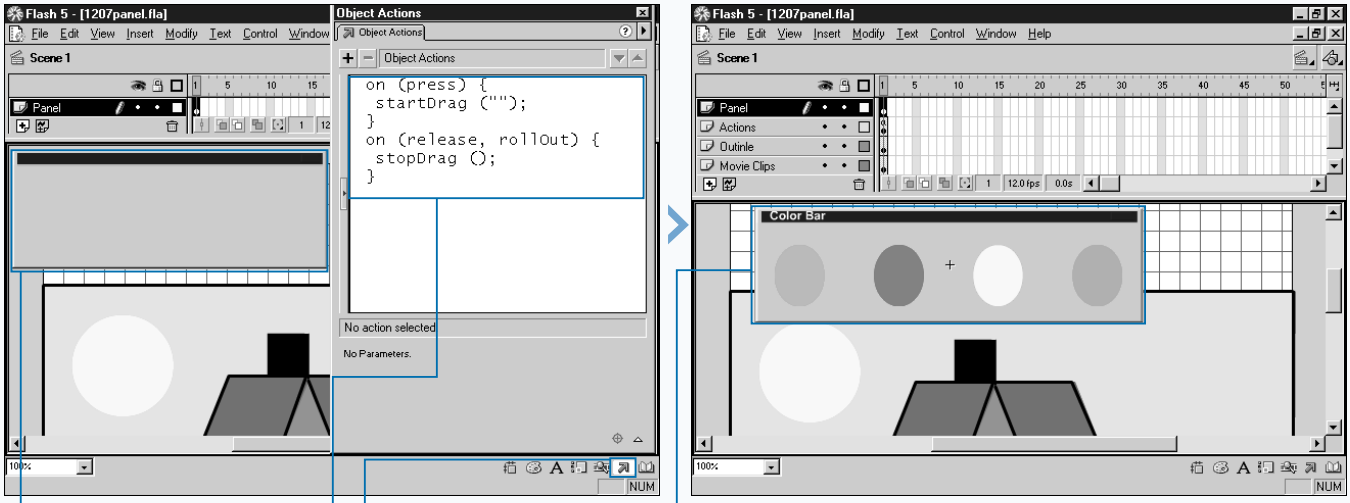
Emulating a panel using ActionScript is easy. You create an object that is draggable, visible when needed, invisible when not needed, and contains a set of options.

The `startDrag` and `stopDrag` actions enable you to create objects that the user can drag. Both movie clips and buttons can be draggable. However, you want an object that is draggable when you click the mouse while the mouse is located over the object and for which the drag action stops when you release the mouse. Movie clips can be draggable when clicked, and you can stop the drag action when the mouse is released, but the action is not limited to when the cursor is over the object.

With buttons, you can start the drag action when the user clicks the object using the `On Press` handler. You can stop the drag action with the user releases the mouse or the pointer rolls out of the area of the object by using the `Release` and `Roll Out` handlers. Therefore, you start creating your panel by creating a draggable button that will act as a panel.

Panels provide the user with options or tools that you develop. The options can be anything that you want. They can be buttons or movie clips that enable the user to perform an action. Create the options or tools and position them on the panel.

EMULATE PANELS



1 Create the button that will act as the panel.

Note: This example uses file panel.fla, which you can find on the CD-ROM that accompanies this book.

2 Click to open the Actions panel.

3 Make the panel draggable.

Note: See page 56 for more information on making an object draggable.

4 Create options and place them on the panel.

Note: The options can be anything that you want.

■ This example places a color bar on the panel.

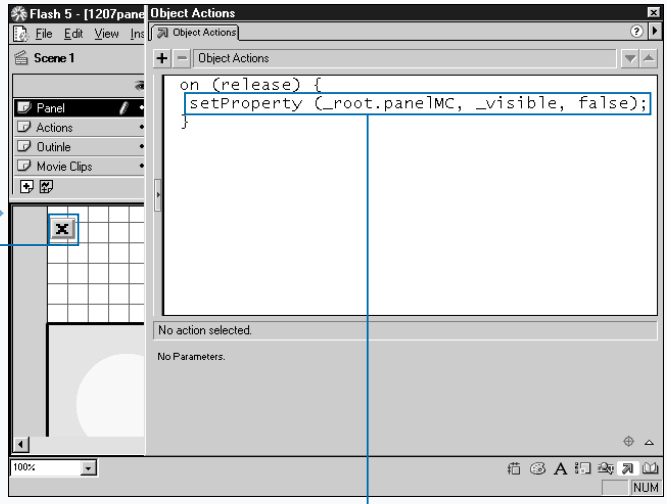
Extra

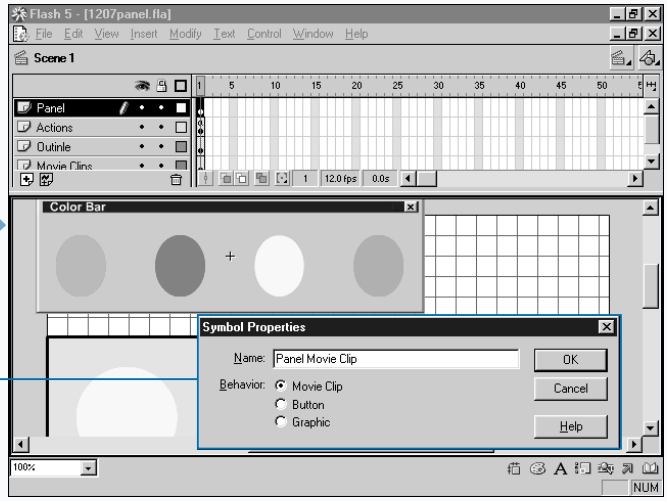
Panels are objects that you may want to use in many of your movies. The common library enables you to easily access objects that you use repeatedly, so you may want to place your panel in the common library. You may also want to turn your panel into a smart clip. Turning your panel into a smart clip enables you to use the same panel in many movies and enables you to adjust the features each time you use the panel.

Flash 5 ships with smart clips that enable you to create menus, radio buttons, and check boxes. You may find these useful when adding features to your panel. They are located in the common library.

When placing tools and options on a panel, make sure that your path names are correct. If you do not specify the correct paths, the panel will not function properly.

There are a number of standard features that you may want to add to your panel. For example, you may want your panel to be resizable or reappear on the screen at a specified location.





5 Create a Close button.

Note: See page 12 for information on creating a Close button.

6 Select the button.

7 Set the movie clip `_visible` property to `false` on release.

8 Convert the panel button, Close button, and options to a movie clip.

Note: See page 252 for more information on converting a button to a movie clip.

CONTINUED

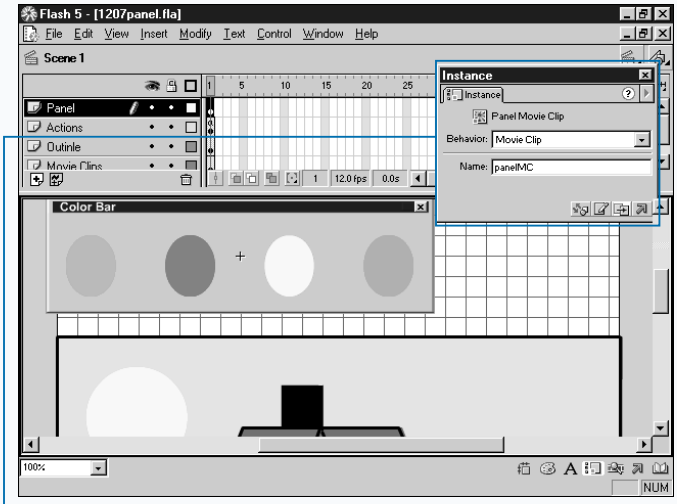
EMULATE PANELS (CONTINUED)

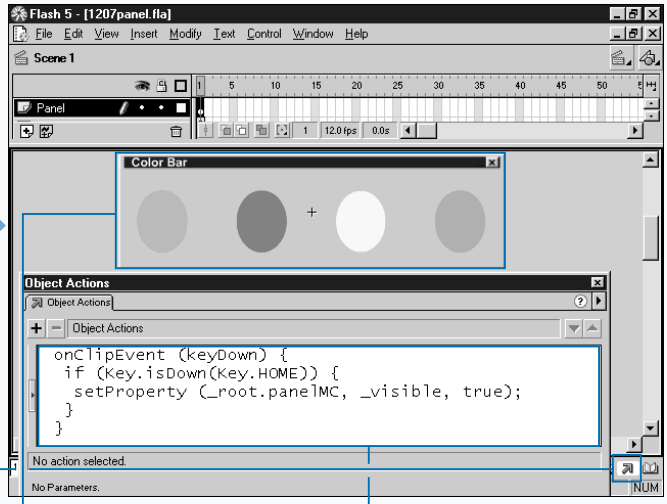
The user needs to be able to hide the panel from view when the user does not need the panel. You can use either the `_visible` property or the `_alpha` property to make an object invisible. However, when you use the `_alpha` property to make an object invisible, the object remains active. If the user clicks in the area in which the object is located, the object responds. If you use the `_visible` property, the object is inactive when it is not visible and does not respond when the user clicks in the area in which the object is located. When creating a panel, use the `_visible` property to hide the panel from view when the user does not need the panel.

The object that you create to use as a panel is a draggable button. You need to change the `_visible` property of the object. Unfortunately, you cannot change the `_visible` property of a button. You can only change the properties of movies and movie clips. Therefore, you must convert the button and all the options to a movie clip. But first, determine the instance name that you want to give the movie clip. Create a button that sets the `_visible` property of a movie clip to `false` when the user clicks the button. Use the movie clip name that you intend to give to the panel. Position the button on the panel. Group all the objects on the panel and convert everything to a movie clip. Then name the instance.

You now have a draggable panel that the user can hide. Once hidden, the user needs to be able to return the object to the Stage when needed. You can create a button that returns the panel to the screen and place the button on the Stage. Or instead, if you use a `Key` method, you can add an action to the panel that returns the panel to the Stage when the user presses a key. Using a key instead of a button makes your panel self-contained and reusable. You can of course use both a key and a button.

EMULATE PANELS (CONTINUED)





9 Name the movie clip instance, using the name that you specified in step 7.

10 Select the movie clip.

11 Click to open the Actions panel.

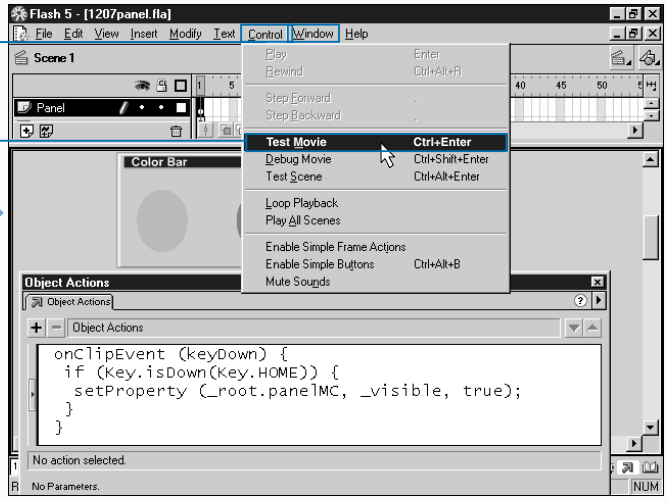
12 Set the `_visible` property to `true` when the user presses the Home key.

Extra

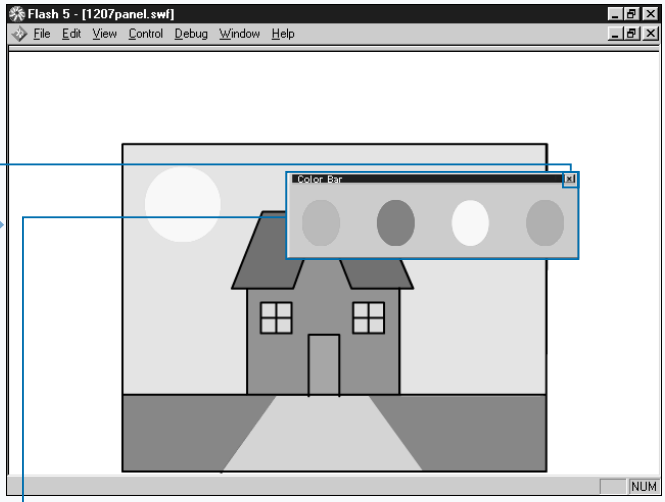
You can use code similar to the code shown here to change the size of a panel. When the user presses Page Up, the script increases the size of the panel by 25 percent. When the user presses Page Down, the script decreases the size of the panel by 25 percent. When the user presses End, the script returns the panel to standard size.

Example:

```
onClipEvent (keyDown) {
  if (Key.isDown(Key.PGUP)) {
    setProperty (_root.panelMC, _xscale, 125);
    setProperty (_root.panelMC, _yscale, 125);
  }
}
onClipEvent (keyDown) {
  if (Key.isDown(Key.PGDN)) {
    setProperty (_root.panelMC, _xscale, 75);
    setProperty (_root.panelMC, _yscale, 75);
  }
}
onClipEvent (keyDown) {
  if (Key.isDown(Key.END)) {
    setProperty (_root.panelMC, _xscale, 100);
    setProperty (_root.panelMC, _yscale, 100);
  }
}
```



13 Click Control ➪ Test Movie to move to the test environment.



14 Test your movie by first dragging the panel.

15 Click ☒ to close the panel.

16 Press the Home key to open the panel.

DEMYSTIFY TRIGONOMETRIC FUNCTIONS

You can use the `Math` object `sin` and `cos` methods to manipulate objects. These trigonometric functions find the sine or cosine of a value. You can use them with the `_x` and `_y` properties to move objects. Explaining trigonometry is beyond the scope of this book; however, the following is a formula that you can use to move objects located on the Stage: movieClip._x = l + Math.cos(s)*d movieClip._y = l + Math.sin(s)*a

The `l` represents location, `s` speed, and `a` area. Use `movieClip` to specify the movie clip that you want to move. Use `l` to adjust the coordinate at which the movie clip is located. Use `s` to adjust the speed at which the object moves. The higher the value of `s`, the faster the object will move. Use `a` to adjust the area in which the object moves. The higher the value of `a`, the larger the area in which the object will move.

In trigonometry, you use radians to measure the size of angles. You use the formula `Math.PI/180 * degrees` to convert degrees to radians. Although not essential, you may find the formula `Math.PI/180 * 2` a good base from which you can adjust the speed of your object. For the sake of the examples that follow, `Math.PI/180 * 2` is assigned to the variable `s`, as shown in the following:

```
s = Math.PI/180 * 2;
```

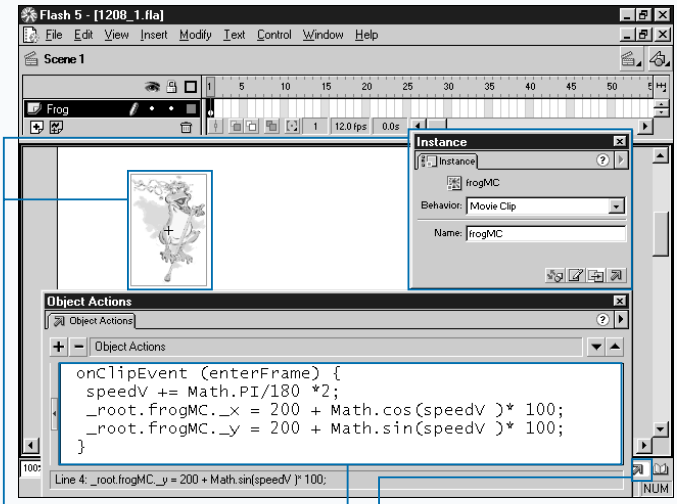
You can move an object in a clockwise circle by setting each of the variables in both formulas to the same value. The following is an example:

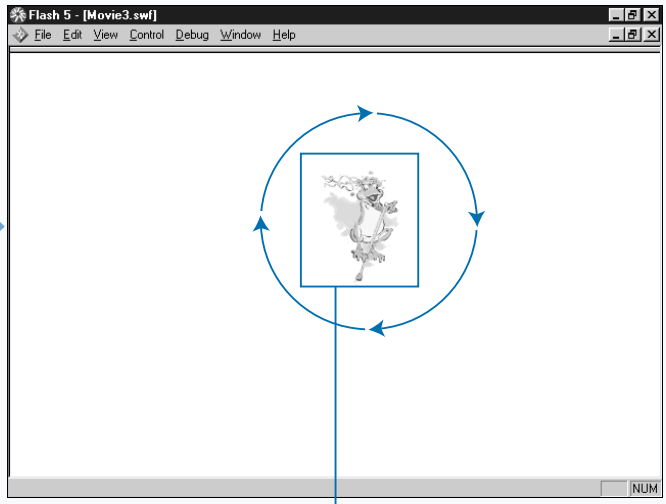
```
_sampleMC._x = 200 + Math.cos(s)*100;
_sampleMC._y = 200 + Math.sin(s)*100;
```

To cause the object to move in a counterclockwise direction, negate the speed value, as shown in the following:

```
_sampleMC._x = 200 + Math.cos(-s)*100;
_sampleMC._y = 200 + Math.sin(-s)*100;
```

MOVE AN OBJECT CLOCKWISE





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file trig_1.fla, which you can find on the CD-ROM that accompanies this book.

2 Name a movie clip instance.

3 Click to open the Actions panel.

4 Use the `enterFrame` handler.

5 Move the object clockwise.

6 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

7 Test your movie.

■ The movie clip moves in a clockwise direction.

Extra

You can add the scripts shown here to movie clips to create a simple but interesting design.

Example:

redMC

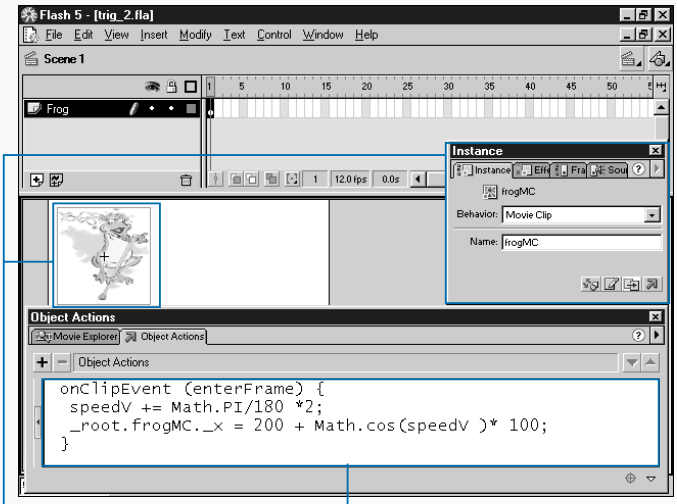
```
onClipEvent (enterFrame) {
   radianV += Math.PI/180*2;
   _root.redMC._y  = 200 + Math.sin(radianV* 5)*50;
   _root.redMC._x  = 200 + Math.cos(radianV* 1)*200;
}
```

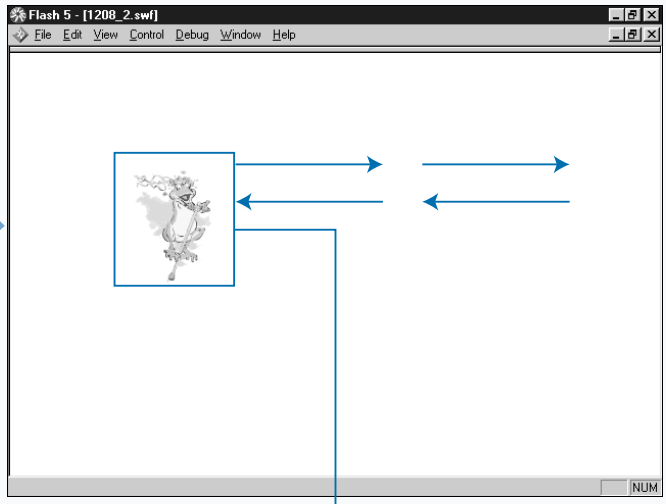
Example:

yellowMC

```
onClipEvent (enterFrame) {
   radianV += Math.PI/180*2;
   _root.yellowMC._y  = 200 + Math.sin(radianV* 5)* 50;
   _root.yellowMC._x  = 200 + Math.cos(radianV* 2)* 200;
}
```

MOVE AN OBJECT BACK AND FORTH





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file trig_2.fla, which you can find on the CD-ROM that accompanies this book.

2 Name a movie clip instance.

3 Click to open the Actions panel.

4 Use the `enterFrame` handler.

5 Move the object back and forth.

6 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

7 Test your movie.

■ The movie clip moves back and forth.

CONTINUED

DEMYSTIFY TRIGONOMETRIC FUNCTIONS (CONTINUED)

To move the object back and forth, use the formula for the _x coordinate but do not use the formula for the _y coordinate, as shown in the following:

```
_sampleMC._x = 200 + Math.cos(s)*100;
```

To move the object up and down, use the formula for the _y coordinate, but do not use the formula for the _x coordinate — for example:

```
_sampleMC._y = 200 + Math.sin(s)*100;
```

To have the object move diagonally, use the same trigonometric function for both the _x and the _y coordinates, as shown here:

```
_sampleMC._x = 200 + Math.cos(s)*100;
_sampleMC._y = 200 + Math.cos(s)*100;
```

To increase the speed, multiply s by the factor by which you want to increase the speed. To decrease the speed, divide s by the factor by which you want to decrease the speed:

```
_sampleMC._y = 200 + Math.sin(s *10)*100;
```

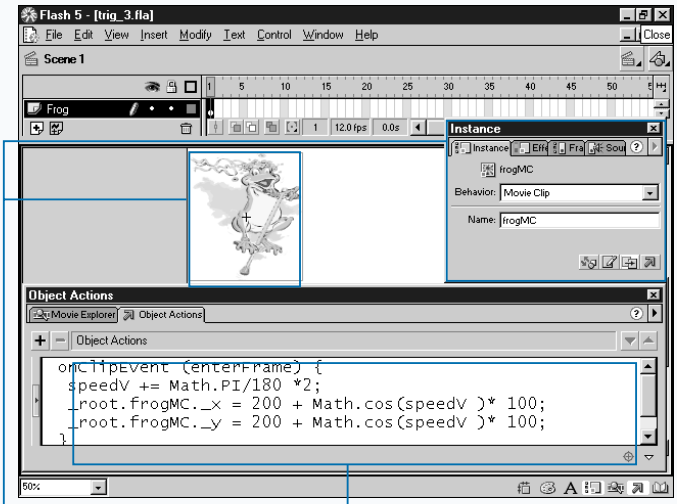
To cause the object to move across the stage in a wave-like motion, increase the area value of _x and the speed of _y:

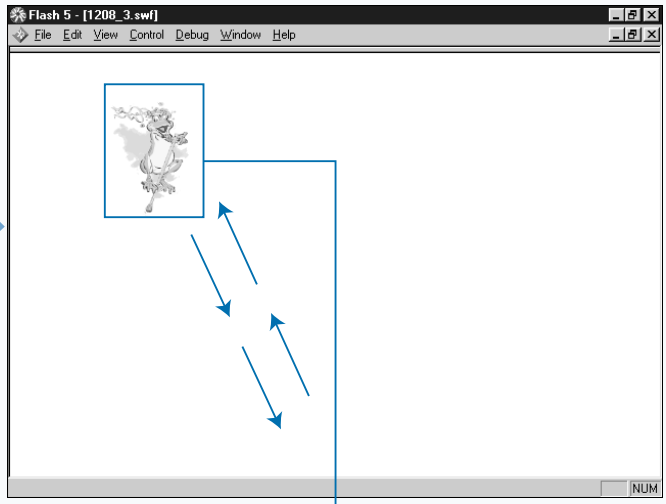
```
sampleMC._x = 200 + Math.cos(s)*300;
sampleMC._y = 200 + Math.sin(s*10)*100;
```

To cause the object to move up and down the Stage in a wave-like motion, increase the area value of _y and the speed of _x, as shown in the following:

```
sampleMC._x = 200 + Math.cos(s * 10)*100;
sampleMC._y = 200 + Math.sin(s)*300;
```

MOVE AN OBJECT DIAGONALLY





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file trig_3.fla, which you can find on the CD-ROM that accompanies this book.

2 Name a movie clip instance.

3 Click to open the Actions panel.

4 Use the `enterFrame` handler.

5 Move the object diagonally.

6 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

7 Test your movie.

■ The movie clip moves diagonally.

Extra

These scripts produce a very simple design.

Example:

redMC

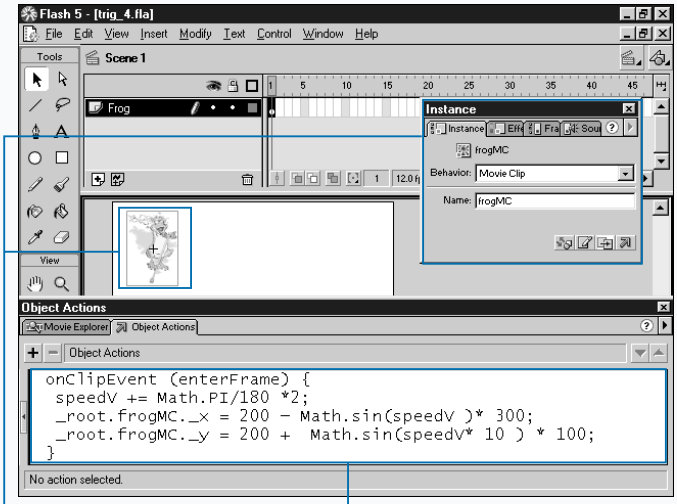
```
onClipEvent (enterFrame) {
   radianV += Math.PI/180*2;
   _root.redMC._x  = 200 + Math.cos(radianV)*100;
}
```

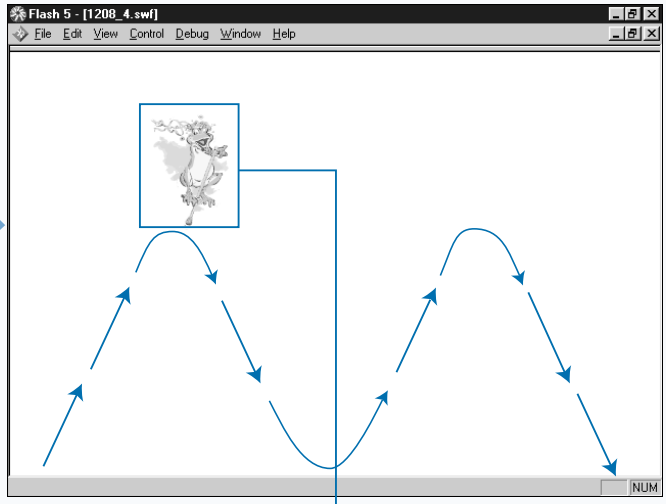
Example:

yellowMC

```
onClipEvent (enterFrame) {
   radianV += Math.PI/180*2;
   _root.yellowMC._y = 200 + Math.sin(radianV) * 100;
   _root.yellowMC._x = 200 + Math.cos(radianV) * 100;
}
```

MOVE AN OBJECT IN A WAVE-LIKE MOTION





1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file trig_4.fla, which you can find on the CD-ROM that accompanies this book.

2 Name a movie clip instance.

3 Click to open the Actions panel.

4 Use the `enterFrame` handler.

5 Move the object in a wave-like motion.

6 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

7 Test your movie.

■ The movie clip moves in a wave-like motion.

CREATE TRIGONOMETRIC SPECIAL EFFECTS

You can use the `Math` object `sin` or `cos` methods with the `_xscale`, `_yscale`, `_height`, `_width`, `_rotation`, and `_alpha` properties to create special effects. The `_height` and `_width` properties change the height and width of a movie clip using pixels. The `_xscale` and `_yscale` properties change the height and width of a movie clip using a percentage. Because these two methods change the size of a movie clip using two different methods, they produce slightly different results.

The `_xscale` property can produce the illusion of an object flipping from left to right. The `_yscale` property can produce the illusion of an object flipping up and down. The `_height` and `_width` properties shrink and grow an object. The `_rotation` property produces a spinning effect. The object spins in one direction and then spins in the opposite direction. The `_alpha` property fades color in and out.

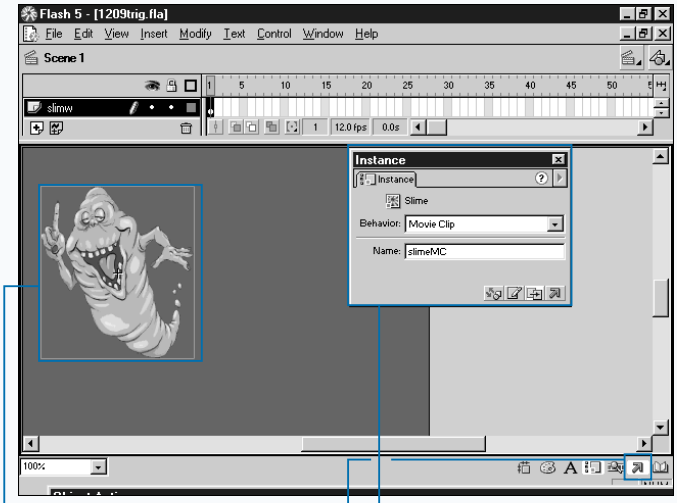
You can use the following formulas when using `sin` or `cos` with these properties:

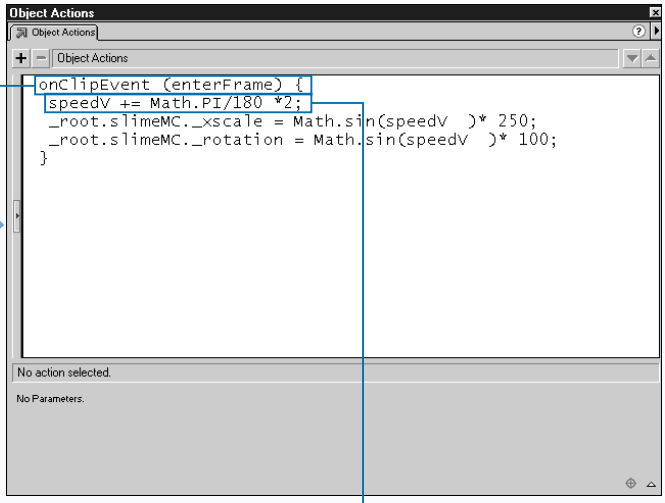
```
movieClip._property = Math.sin(s)*a
movieClip._property = Math.cos(s)*a
```

The `s` represents speed. The `a` represents area. Use `movieClip` to specify the name of the movie clip whose property you want to change. Use `s` to adjust the speed at which the property changes. The higher the value of `s`, the faster the property will change. Use `a` with the `_height`, `_width`, `_xscale`, and `_yscale` properties to determine the maximum size of the object. Remember, `_height` and `_width` use pixels, and `_xscale` and `_yscale` use a percentage. Use `a` with the rotation to adjust the number of rotations the object spins clockwise before spinning counterclockwise.

You may use the formula `Math.PI/180 * 2` as a base from which you adjust the speed of your object.

CREATE TRIGONOMETRIC SPECIAL EFFECTS





1 Select the frame, button, or movie clip to which you want to add ActionScript.

2 Name the instance.

Note: See page 10 for more information on naming an instance.

3 Click ⊠ to open the Actions panel.

4 Use the `enterFrame` handler.

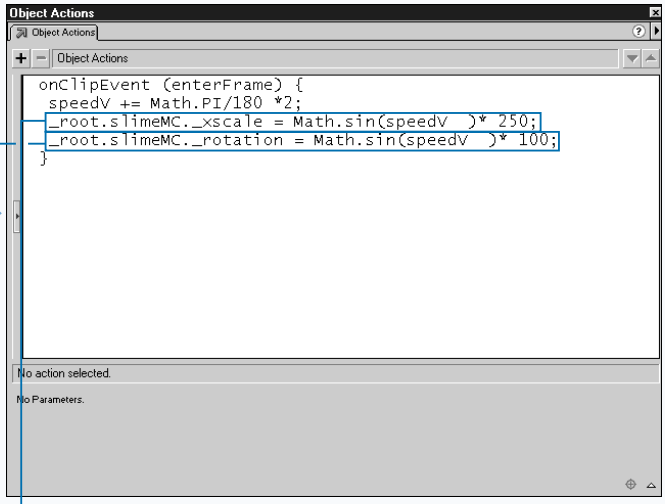
5 Set the base speed.

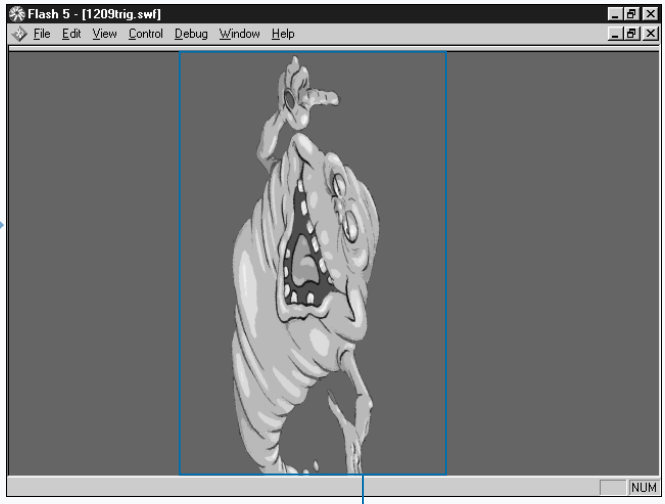
Extra

You attach the script that follows to three separate movie clips to create a design. It uses `sin` with the `_height`, `_width`, `_rotation`, and `_alpha` properties.

Example:

```
onClipEvent (enterFrame) {
   speedV += Math.PI/180 *2;
   _root.blockMC._height = Math.cos(speedV * 3 )* 100;
   _root.blockMC._width = Math.sin(speedV * 3 )* 100;
   _root.blockMC._rotation = Math.sin(speedV * 3 )* 100;
   _root.blockMC._alpha = Math.sin(speedV * 3 )* 100;
}
```





6 Use sin to set the `_xscale`.

7 Use sin to set the `_rotation`.

8 Move to the test environment.

Note: See page 38 for instructions on how to test your movie.

9 Test your movie.

■ Your movie clip will scale and rotate.

USING THE DEBUGGER

Correcting errors — often referred to as ***debugging*** — is a normal part of writing a program. Flash provides several tools that can help you debug ActionScript. Use the Debugger to display a list of loaded movies, movie clips, properties, and variables. The Debugger enables you to view and adjust the properties and variables as the movie plays. Using the Debugger, you can experiment with values until you find the one that you need. You can then return to your script and make the necessary changes.

The Debugger consists of a status bar, display list, Properties tab, Variables tab, and Watch list. The status bar displays the URL or path to a movie or movie clip.

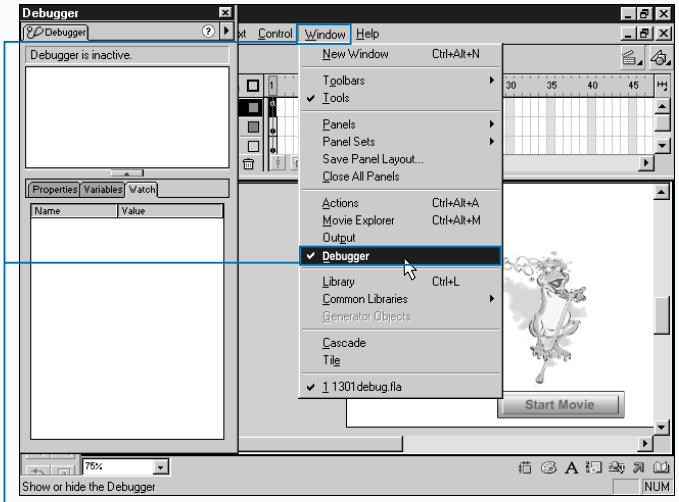
The display list displays all levels and instances that are currently loaded. As you add or remove movies and movie clips from the Stage, the display list automatically updates. A splitter bar enables you to open, close, or resize the display list.

The Properties tab displays all the properties associated with a movie clip. You can change a property value by double-clicking the property value and typing a new value. You cannot use an expression to change a value when using the Debugger. You must enter a string, Boolean, or number. When you change a property value, Flash reflects the change on the Stage immediately. You cannot change read-only values.

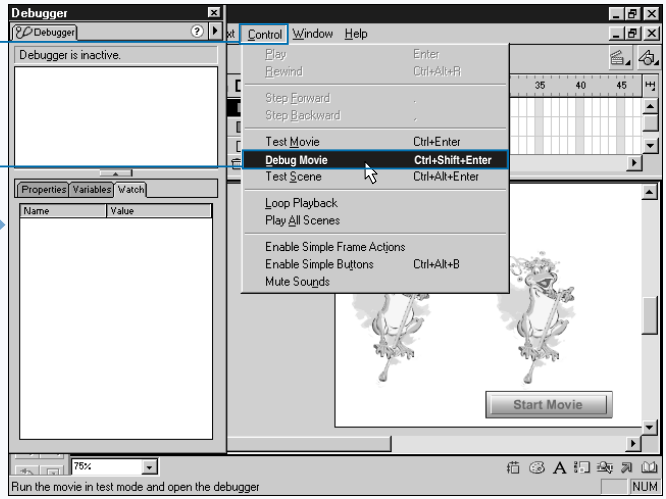
The Variables tab displays all the variables and the current values of the variables. You can change the value of a variable by double-clicking the variable and typing a new value. Object and array values are displayed, but you cannot change them.

If you have specific variables that you want to track, you can place them on the Watch list. The Watch list enables you to track a specified list of variables.

USING THE DEBUGGER



1 Click Window ➪ Debugger to open the Debugger.



2 Click Control ➪ Debug Movie to activate the Debugger.

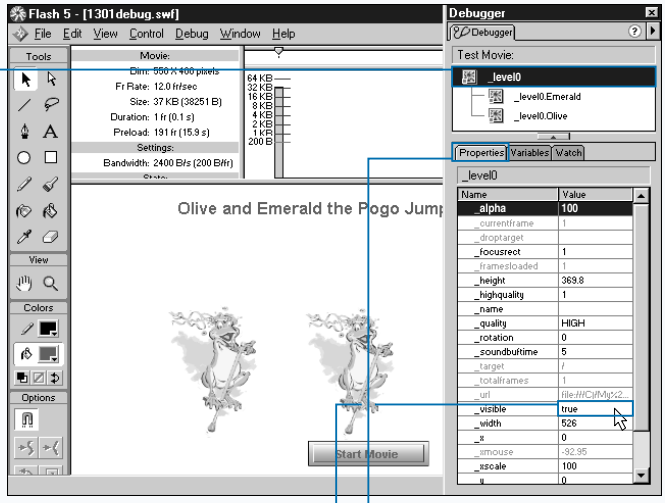
Extra

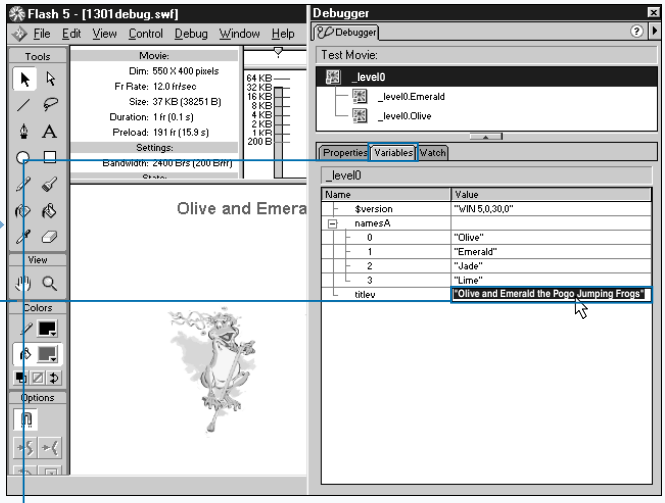
The Debugger uses a special version of Flash Player — the Flash Debug Player — that installs automatically when you install Flash 5. To activate the Flash Debug Player, click Control ➪ Debug Movie.

You can use the Flash Debug Player as a standalone player. If you choose Debugging Permitted under the Flash tab of the Publish Settings dialog box, when you publish your Flash movie, you can use the Debugger as you play your movie. Just click Control ➪ Debugger in the Flash Debug Player menu. When using Debugger with the Flash Debug Player, Flash must be open. You may want to password-protect your movie because anyone viewing your movie using the Flash Debug Player will also be able to view your property and variable settings. The Flash tab of the Publish Settings dialog box provides an option that enables you to password-protect the Debugger option.

You can download the latest version of the Flash Debug Player from the Macromedia Web site. A link to the site is included on the CD-ROM that accompanies this book.

To add a variable to the Watch list, select the variable, click ▶ in the upper-right corner of the Debugger panel to open the menu, and then click Add Watch. To remove a variable from the Watch list, select the variable, click ▶, and then click Remove Watch.





3 Click a level or instance to select it.

4 Click the Properties tab to view properties.

5 Double-click a value and type to change a property.

6 Click the Variables tab to view the variables.

7 Double-click a value and type to change a variable.

■ You can use Debugger to troubleshoot your movies.

USING THE OUTPUT WINDOW

The Output window is part of the test environment. The Output window displays information to help you troubleshoot your movie. If you have syntax errors in your movie, error messages automatically display in the Output window when you enter the test environment. Error messages help you locate problems with your script.

The Output window's Options menu provides you with options that enable you to copy the contents of the Output window to the clipboard, clear the Output window, save the contents of the Output window to a file, or print the contents of the Output window. Choosing the Save to File option on the menu enables you to save the contents of the Output window as a TXT or LOG file.

The Options menu also enables you to select how in-depth you want the error messages that you receive to be. Choose from None, Errors, Warnings, and Verbose. Choosing None turns off the display of error messages.

You can use the `trace` action to evaluate an expression. This is useful when you are debugging your movie and you need to know the value that an expression returns. The results of the `trace` action display in the Output window. The syntax for the `trace` action is

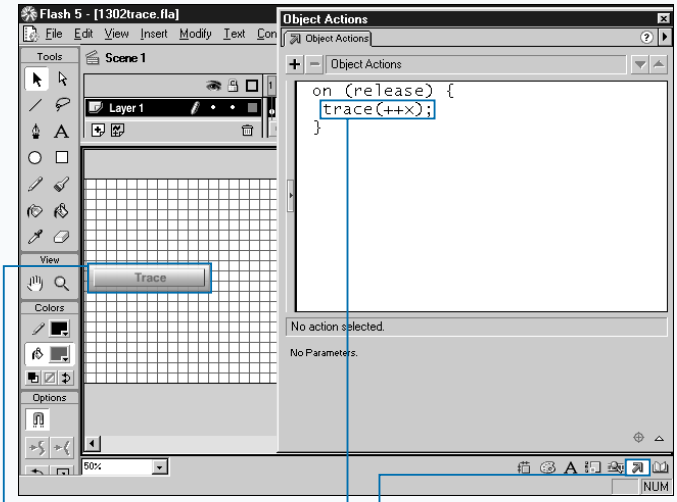
```
trace(expression);
```

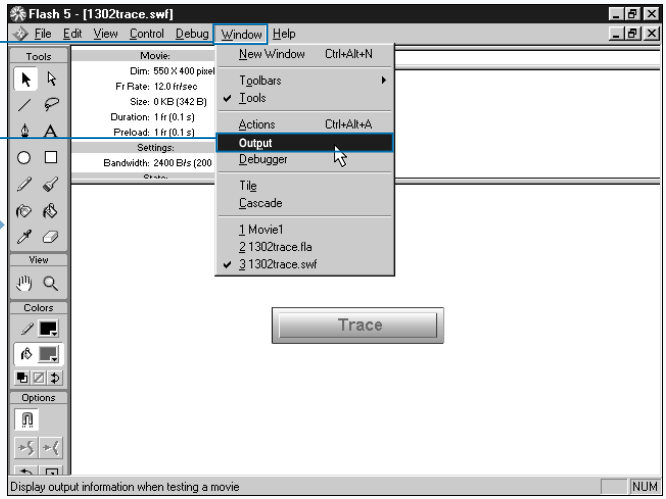
Use the `expression` argument to specify the expression that you want to evaluate. For example, the following script increments the value of `x` each time the user clicks a button and sends the results to the Output window:

```
on (release) {
trace(++x);
}
```

You can also use the `trace` action to display messages to anyone who views your movie using the test environment. For example, `trace("hello");` displays "hello" in the Output window.

USING THE OUTPUT WINDOW





USING THE TRACE ACTION

1 Select the frame, button, or movie clip to which you want to add ActionScript.

Note: This example uses file trace.fla, which you can find on the CD-ROM that accompanies this book.

2 Click to open the Actions panel.

3 Use the `trace` action to evaluate an expression.

4 Move to the test environment.

5 Click Window ➪ Output to open the Output window.

Extra

The script shown here uses the `trace` action to display changes made to an array.

Script:

```
on (release) {
    name = ["George", "John", "James"];
    trace (name);
    name[1] = "Chris";
    trace (name);
    name.push("Gene", "Charles", "Sue" );
    trace (name);
}
```

Results of the First Trace:

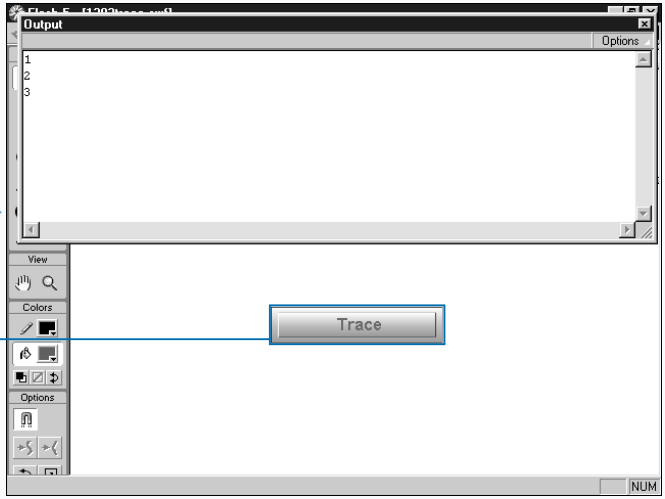
```
George,John,James
```

Results of the Second Trace:

```
George,Chris,James
```

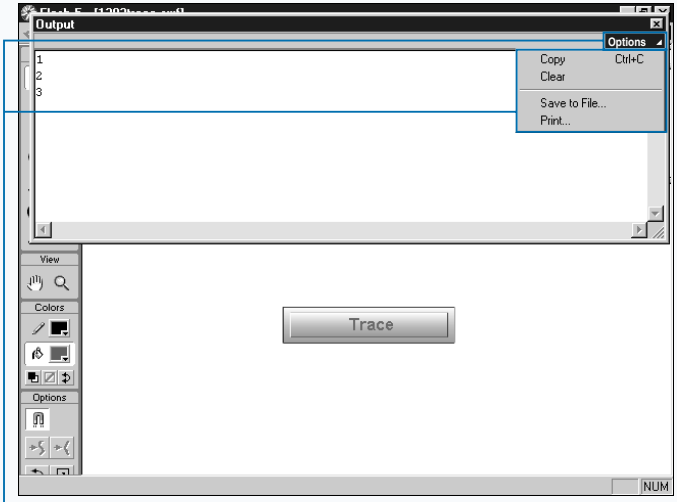
Results of the Third Trace:

```
George,Chris,James,Gene,Charles,Sue
```



6 Click the button to test your movie.

■ Each time you click the button, the ouput window will display the results of the trace.



THE OPTIONS MENU

1 Click Options to open the Options menu.

2 Select an option.

■ You can use the options menu to copy the contents of the Output window to the clipboard, clear the Output window, save the contents of the Output window to a file, or print the contents of the output window.

USING LIST OBJECTS

In the test environment, you can use the menu choice Debug ➪ List Objects to display levels, object types, and target paths for the current frame in the Output window. Object types include shape, movie clip, text, and button. The information provided by the Debug ➪ List Object menu choice is useful when you want to find a target path or what text is assigned to a text field.

When you use Debug ➪ List Objects, the Output window does not update automatically as the Debugger does. You must make the menu choice each time you want to retrieve information.

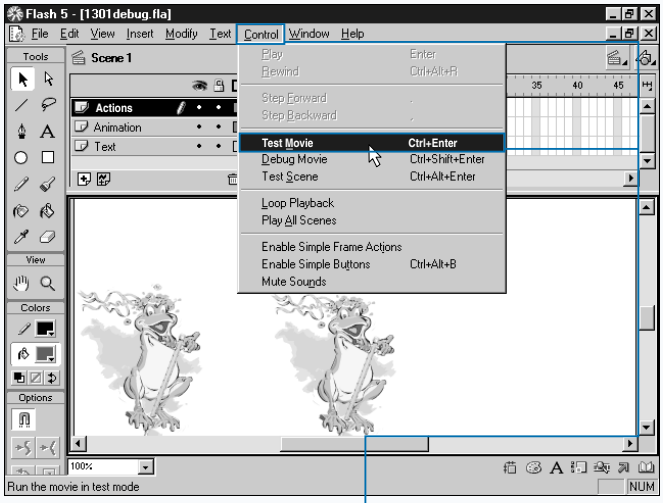
Extra

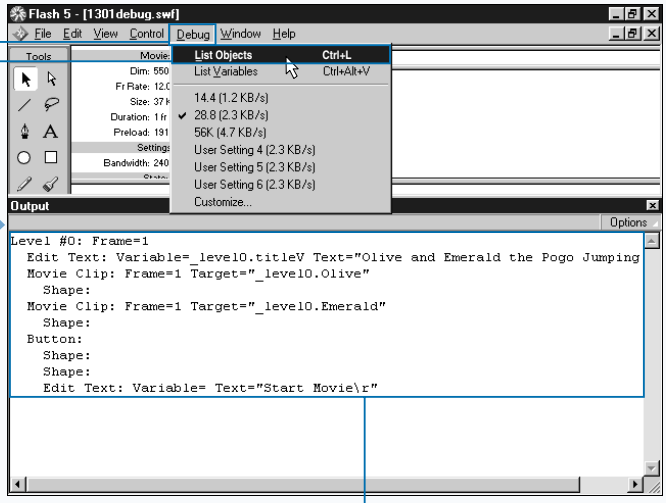
The following is an example of output from the Debug ➪ List Objects menu selection.

Example:

```
Level #0: Frame=1
Edit Text: Variable=_level0.titleV
Text=""
Movie Clip: Frame=1
Target="_level0.Olive"
  Shape:
Movie Clip: Frame=1
Target="_level0.Emerald"
  Shape:
Button:
  Shape:
  Shape:
  Edit Text: Variable= Text="Start
Movie\r"
```

USING LIST OBJECTS





1 Create your movie.

2 Click Control ➪ Test Movie to move to the test environment.

3 Click Debug ➪ List Objects.

■ The results display in the Output window.

USING LIST VARIABLES

In the test environment, you can use the menu choice Debug ➪ List Variables to display all the movie variables in the Output window. The information provided by the Debug ➪ List Variables menu choice is useful when you want to find a variable name or the value assigned to a variable. All the values assigned to an array are displayed in the Output window.

When you use Debug ➪ List Variables, the Output window does not update automatically as the Debugger does. You must make the menu choice each time that you want to retrieve information.

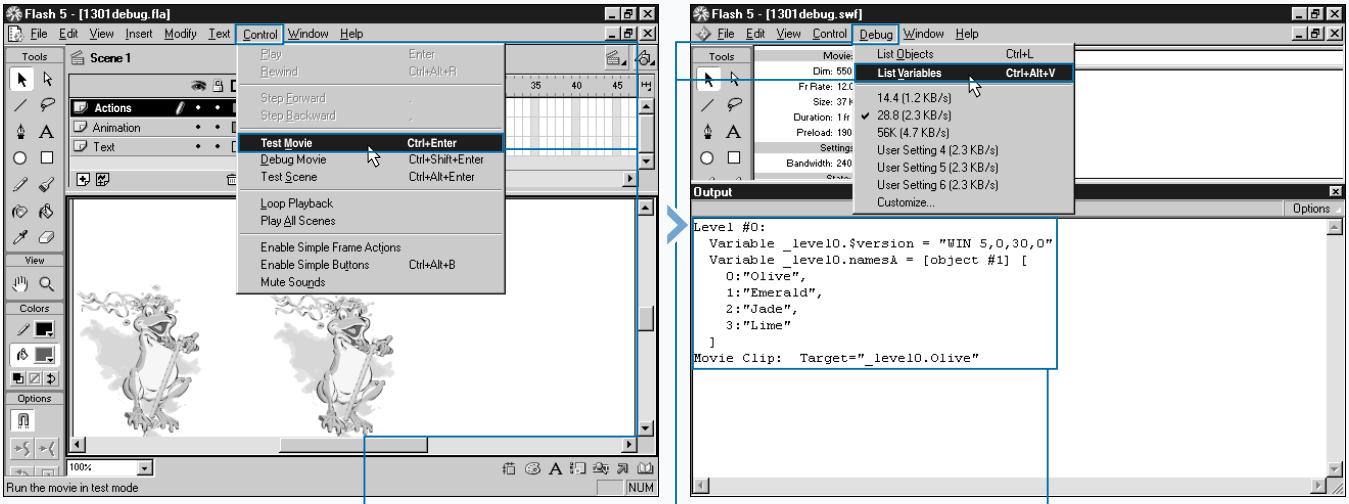
Extra

The following is an example of output from the Debug ➪ Lists Variables menu selection. `namesA` is an array.

Example:

```
Level #0:
  Variable _level0.$version = "WIN 5,0,30,0"
  Variable _level0.namesA = [object #1] [
    0:"Emerald",
    1:"Jade"
  ]
  Variable _level0.nameV = "Emerald the
   Pogo Jumping Frog"
```

USING LIST VARIABLES



1 Create your movie.

2 Click Control ➪ Test Movie to move to the test environment.

3 Click Debug ➪ List Variables.

■ The results display in the Output window.

DEBUG YOUR SCRIPT

There are techniques that you can use to effectively create and debug ActionScript. For starters, you should build your movie in a modular fashion, testing each section as you develop it. This enables you to identify problems as they arise.

As you create your scripts, you should save often, giving each save a different version number. If your script gets extremely complex and stops working, this enables you to go back to the last working version of the script.

If a portion of script does not work, try breaking the script down into component parts. This enables you to narrow things down so that you can determine which portion of the script is not working. You can also comment out portions of the script and test it. Again, this helps narrow down where the error is located.

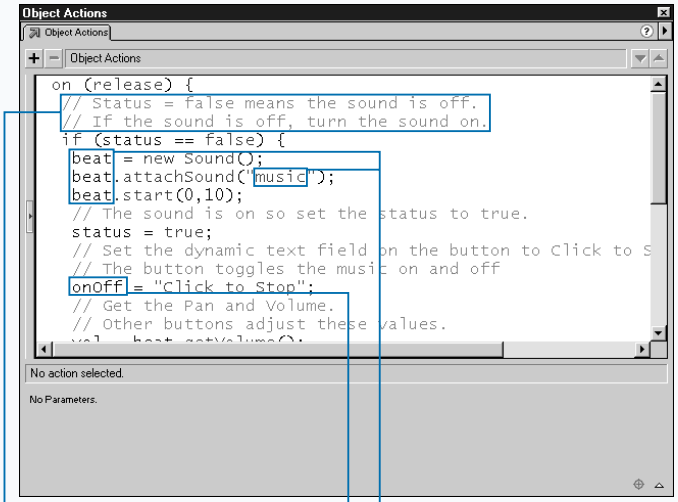
Use comments to document your code. This is very important if you are working with other developers. Even if you do not work with other developers, comments are important. You may need to revise your code weeks, months, or years later. It is very easy to forget the logic that you used to develop your code. Use comments to explain what you are doing and why you are doing it.

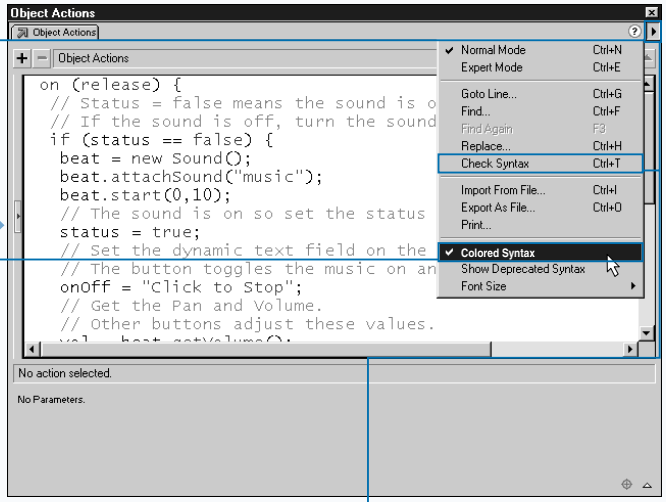
Give the variables, arrays, functions, and objects that you create meaningful names. This will make it easier for you and others who view your code to understand your code.

Develop a naming convention and stick with it. For example, you can type the first word in a variable name in small letters, and subsequent words can start with capital letters. The variable name `playerFirstName` is an example of this convention.

When trying to understand a movie written by someone else, use Movie Explorer to find and view ActionScript.

DEBUG YOUR SCRIPT





1 Add comments.

2 Add meaningful names.

3 Add a naming convention.

4 Click ▶ ➪ Colored Syntax to use colored syntax.

5 Click ▶ ➪ Check Syntax to perform a syntax check.

Extra

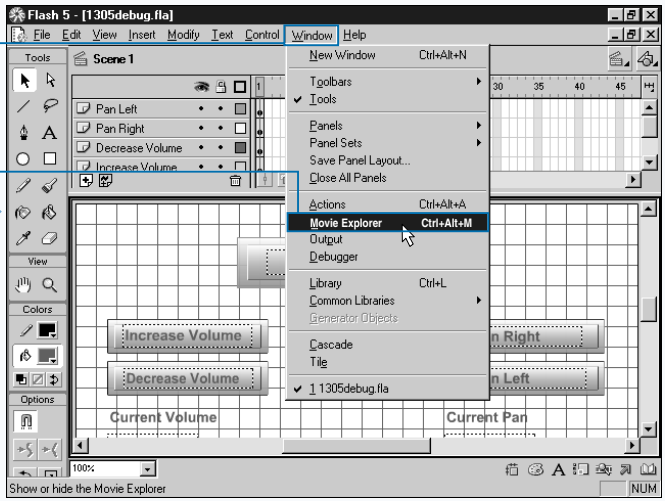
When using the Action panel, use Colored Syntax.

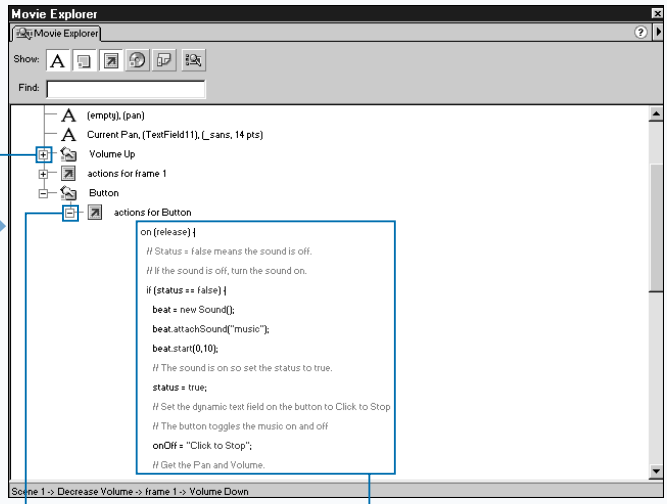
Perform syntax checks. This is particularly important if you are working in the Expert Mode.

Give every variable, function, object, and array a unique name.

COMMON ERRORS

ERROR	SOLUTION
Misspelled variable or array name	Make sure that you spell variable and array names the same each time that you use them.
Expression box not checked	If you are working in Normal Mode, make sure that you check the Expression box when you enter an expression.
Using = to perform a comparison	The operator = is the assignment operator. If you want to perform a comparison, such as `if name == "George"`, use ==, the equality operator.





6 Click Window ➪ Movie Explorer to open Movie Explorer.

7 Click − to contract the hierarchy.

8 Click + to expand the hierarchy.

9 View the ActionScript.

■ You can develop your scripts in a way that makes it easy for you to debug them.

KEY CODE VALUES

The tables that follow list all of the keys on a standard keyboard and the corresponding key code values.

LETTERS A TO Z AND NUMBERS 0 TO 9

LETTER OR NUMBER	KEY CODE	LETTER OR NUMBER	KEY CODE	LETTER OR NUMBER	KEY CODE	LETTER OR NUMBER	KEY CODE
A	65	J	74	S	83	1	49
B	66	K	75	T	84	2	50
C	67	L	76	U	85	3	51
D	68	M	77	V	86	4	52
E	69	N	78	W	87	5	53
F	70	O	79	X	88	6	54
G	71	P	80	Y	89	7	55
H	72	Q	81	Z	90	8	56
I	73	R	82	0	48	9	57

KEYS ON THE NUMERIC KEYPAD

KEY	KEYCODE	KEY	KEYCODE	KEY	KEYCODE
Number Pad 0	96	Number Pad 6	102	Add	107
Number Pad 1	97	Number Pad 7	103	Enter	108
Number Pad 2	98	Number Pad 8	104	Subtract	109
Number Pad 3	99	Number Pad 9	105	Decimal	110
Number Pad 4	100	Multiply	106	Divide	111
Number Pad 5	101				

FUNCTION KEYS

KEY	KEYCODE
F1	112
F2	113
F3	114
F4	115

KEY	KEYCODE
F5	116
F6	117
F7	118
F8	119

KEY	KEYCODE
F9	120
F10	121
F11	122
F12	123

OTHER KEYS

KEY	KEYCODE
Backspace	8
Tab	9
Clear	12
Enter	13
Shift	16
Control	17
Alt	18
Cap Lock	20

KEY	KEYCODE
Esc	27
Spacebar	32
Page Up	33
Page Down	34
End	35
Home	36
Left Arrow	37
Up Arrow	38

KEY	KEYCODE
Right Arrow	39
Down Arrow	40
Insert	45
Delete	46
Help	47
Num Lock	144
;:	186
=+	187

KEY	KEYCODE
-_	189
/?	191
`~	192
[{	219
\\\|	220
]}	221
‘”	222

URL ENCODING CHARACTERS

The `loadVariables` action reads data from an external file. You cannot read certain characters directly. You must use URL encoding for these characters. For example, if your text file includes `temperature=+15`, ActionScript will read it as `temperature=15`, dropping the plus. The URL code for the + is `%2b`. To have ActionScript read the entry correctly, enter this: `temperature=%2b15`. The following is a complete list of URL codes.

CHARACTER	URL CODE	CHARACTER	URL CODE	CHARACTER	URL CODE	CHARACTER	URL CODE
backspace	%08	3	%33	J	%4A	a	%61
tab	%09	4	%34	K	%4B	b	%62
linefeed	%0A	5	%35	L	%4C	c	%63
return	%0D	6	%36	M	%4D	d	%64
space	%20	7	%37	N	%4E	e	%65
!	%21	8	%38	O	%4F	f	%66
"	%22	9	%39	P	%50	g	%67
#	%23	:	%3A	Q	%51	h	%68
$	%24	;	%3B	R	%52	i	%69
%	%25	<	%3C	S	%53	j	%6A
&	%26	=	%3D	T	%54	k	%6B
‘	%27	>	%3E	U	%55	l	%6C
(	%28	?	%3F	V	%56	m	%6D
)	%29	@	%40	W	%57	n	%6E
*	%2A	A	%41	X	%58	o	%6F
+	%2B	B	%42	Y	%59	p	%70
,	%2C	C	%43	Z	%5A	q	%71
-	%2D	D	%44	[	%5B	r	%72
.	%2E	E	%45	\	%5C	s	%73
/	%2F	F	%46	]	%5D	t	%74
0	%30	G	%47	^	%5E	u	%75
1	%31	H	%48	_	%5F	v	%76
2	%32	I	%49	`	%60	w	%77

CHARACTER	URL CODE
x	%78
y	%79
z	%7A
{	%7B
|	%7C
}	%7D
~	%7E
¢	%A2
£	%A3
¥	%A5
|	%A6
§	%A7
«	%AB
¬	%AC
¯	%AD
º	%B0
±	%B1
ª	%B2
´	%B4
µ	%B5
»	%BB
1/4	%BC
1/2	%BD

CHARACTER	URL CODE
¿	%BF
À	%C0
Á	%C1
Â	%C2
Ã	%C3
Ä	%C4
Å	%C5
Æ	%C6
Ç	%C7
È	%C8
É	%C9
Ê	%CA
Ë	%CB
Ì	%CC
Í	%CD
Î	%CE
Ï	%CF
_	%D0
Ñ	%D1
Ò	%D2
Ó	%D3
Ô	%D4
Õ	%D5

CHARACTER	URL CODE
Ö	%D6
Ø	%D8
Ù	%D9
Ú	%DA
Û	%DB
Ü	%DC
_	%DD
_	%DE
ß	%DF
à	%E0
á	%E1
â	%E2
ã	%E3
ä	%E4
å	%E5
æ	%E6
ç	%E7
è	%E8
é	%E9
ê	%EA
ë	%EB
ì	%EC
í	%ED

CHARACTER	URL CODE
î	%EE
ï	%EF
_	%F0
ñ	%F1
ò	%F2
ó	%F3
ô	%F4
õ	%F5
ö	%F6
÷	%F7
ø	%F8
ù	%F9
ú	%FA
û	%FB
ü	%FC
_	%FD
_	%FE
ÿ	%FF

LIST OF KEYWORDS

Keywords are words that ActionScript reserves for specific use within the language. You cannot use keywords to name variables, functions, or labels. The following is a list of ActionScript keywords.

break	continue	delete	else
for	function	if	in
new	return	this	typeof
var	void	while	with

ACTIONS APPENDED WITH NUM

When you use `loadMovie`, `unLoadMovie`, `loadVariables`, `print`, or `printAsBitmap` in Normal Mode, ActionScript may substitute those actions with `loadMovieNum`, `unLoadMovieNum`, `loadVariablesNum`, `printNum`, or `printAsBitmapNum`.

In Normal Mode, when using the `loadMovie`, `unLoadMovie`, `loadVariables`, `print`, or `printAsBitmap` actions you use the location parameter to specify either a level number or target path. A level number is an integer. A target path is a string. Flash Player needs to know if the location is a string or a number. Therefore, ActionScript uses two versions of the actions: the action name and the action name appended with Num. ActionScript uses the action name with strings and the action name appended with Num with numbers. When in Normal Mode, if you select Target in the Location field, ActionScript uses the action name. If you select a Level in the Location field, ActionScript uses the action name appended with Num. ActionScript makes the selection automatically. In fact, Normal Mode does not list action names appended with Num in the Toolbox list. In the Expert mode, the programmer must decide whether to use the action name or the action name appended with Num. The Toolbox list lists them both. The programmer should use the action if specifying a target path and the action appended with Num if specifying a level using an integer.

Examples:

```
loadVariableNum ("http://www.
entercompanyhere.com/anymovie.swf", 3)
```

```
loadVariable ("http://www.entercompanyhere.
com /anymovie.swf", "_root.targetClip")
```

Note: If you specify the level using a string, use `loadVariable`.

```
loadVariable ("http://www.entercompanyhere.
com/anymovie.swf", "_level1")
```

HEXADECIMAL COLOR CODES

You can specify color using the red, green, and blue hexadecimal values shown in the following table. This table is not a complete list of hexadecimal values.

COLOR	RED	GREEN	BLUE
aliceblue	F0	F8	FF
antiquewhite	FA	EB	D7
aqua	00	FF	FF
aquamarine	7F	FF	D4
azure	F0	FF	FF
beige	F5	F5	DC
bisque	FF	E4	C4
black	00	00	00
blanchedalmond	FF	EB	CD
blue	00	00	FF
blueviolet	8A	2B	E2

COLOR	RED	GREEN	BLUE
brown	A5	2A	2A
burlywood	DE	B8	87
cadetblue	5F	9E	A0
chartreuse	7F	FF	00
chocolate	D2	69	1E
coral	FF	7F	50
cornflowerblue	64	95	ED
cornsilk	FF	F8	DC
crimson	DC	14	3C
cyan	00	FF	FF
darkblue	00	00	8B

COLOR	RED	GREEN	BLUE
darkcyan	00	8B	8B
darkgoldenrod	B8	86	0B
darkgray	A9	A9	A9
darkgreen	00	64	00
darkkhaki	BD	B7	6B
darkmagenta	8B	00	8B
darkolivegreen	55	6B	2F
darkorange	FF	8C	00
darkorchid	99	32	CC
darkred	8B	00	00
darksalmon	E9	96	7A
darkseagreen	8F	BC	8F
darkslateblue	48	3D	8B
darkslategray	2F	4F	4F
darkturquoise	00	CE	D1
darkviolet	94	00	D3
deeppink	FF	14	93
deepskyblue	00	BF	FF
dimgray	69	69	69
dodgerblue	1E	90	FF
firebrick	B2	22	22
floralwhite	FF	FA	F0
forestgreen	22	8B	22
fuchsia	FF	00	FF
gainsboro	DC	DC	DC
ghostwhite	F8	F8	FF
gold	FF	D7	00
goldenrod	DA	A5	20
gray	80	80	80
green	00	80	00
greenyellow	AD	FF	2F
honeydew	F0	FF	F0
hotpink	FF	69	B4
indianred	CD	5C	5C

COLOR	RED	GREEN	BLUE
indigo	4B	00	82
ivory	FF	FF	F0
khaki	F0	E6	8C
lavender	E6	E6	FA
lavenderblush	FF	F0	F5
lawngreen	7C	FC	00
lemonchiffon	FF	FA	CD
lightblue	AD	D8	E6
lightcoral	F0	80	80
lightcyan	E0	FF	FF
lightgoldenrodyellow	FA	FA	D2
lightgreen	90	EE	90
lightgray	D3	D3	D3
lightpink	FF	B6	C1
lightsalmon	FF	A0	7A
lightseagreen	20	B2	AA
lightskyblue	87	CE	FA
lightslategray	77	88	99
lightsteelblue	B0	C4	DE
lightyellow	FF	FF	E0
lime	00	FF	00
limegreen	32	CD	32
linen	FA	F0	E6
magenta	FF	00	FF
maroon	80	00	00
mediumaquamarine	66	CD	AA
mediumblue	00	00	CD
mediumorchid	BA	55	D3
mediumpurple	93	70	DB
mediumseagreen	3C	B3	71
mediumslateblue	7B	68	EE
mediumspringgreen	00	FA	9A
mediumturquoise	48	D1	CC

Continued

HEXADECIMAL COLOR CODES (CONTINUED)

COLOR	RED	GREEN	BLUE
mediumvioletred	C7	15	85
midnightblue	19	19	70
mintcream	F5	FF	FA
mistyrose	FF	E4	E1
moccasin	FF	E4	B5
navajowhite	FF	DE	AD
navy	00	00	80
oldlace	FD	F5	E6
olive	80	80	00
olivedrab	6B	8E	23
orange	FF	A5	00
orangered	FF	45	00
orchid	DA	70	D6
palegoldenrod	EE	E8	AA
palegreen	98	FB	98
paleturquoise	AF	EE	EE
palevioletred	DB	70	93
papayawhip	FF	EF	D5
peachpuff	FF	DA	B9
peru	CD	85	3F
pink	FF	C0	CB
plum	DD	A0	DD
powderblue	B0	E0	E6
purple	80	00	80
red	FF	00	00
rosybrown	BC	8F	8F

COLOR	RED	GREEN	BLUE
royalblue	41	69	E1
saddlebrown	8B	45	13
salmon	FA	80	72
sandybrown	F4	A4	60
seagreen	2E	8B	57
seashell	FF	F5	EE
sienna	A0	52	2D
silver	C0	C0	C0
skyblue	87	CE	EB
slateblue	6A	5A	CD
slategray	70	80	90
snow	FF	FA	FA
springgreen	00	FF	7F
steelblue	46	82	B4
tan	D2	B4	8C
teal	00	80	80
thistle	D8	BF	D8
tomato	FF	63	47
turquoise	40	E0	D0
violet	EE	82	EE
wheat	F5	DE	B3
white	FF	FF	FF
whitesmoke	F5	F5	F5
yellow	FF	FF	00
yellowgreen	9A	CD	32

BUTTON HANDLERS

You use button handlers to instruct Flash on how to respond when the user clicks, moves the mouse over, or interacts in some other way with a button. The table that follows lists button handlers and their purpose.

HANDLER	PURPOSE
`on(release)`	Performs specified actions when the pointer is over the button and the user releases the mouse. This is the default handler.
`on(press)`	Performs specified actions when the pointer is over the button and the user presses the mouse.
`on(releaseOutside)`	Performs specified actions when the user drags the pointer outside the button area and releases the mouse.
`on(rollOver)`	Performs specified actions when the user rolls the pointer over the button.
`on(rollOut)`	Performs specified actions when the user rolls the pointer over and then outside of the button area.
`on(dragOver)`	Performs specified actions when the user clicks the button, drags the pointer away from the button, and then drags the pointer back over the button.
`on(dragOut)`	Performs specified actions when the pointer is over the button and the user presses the mouse and then drags the pointer outside the button area.
`on(keyPress, "key")`	Performs specified actions when the user presses a specified key.

MOVIE CLIP HANDLERS

You use movie clip handlers to instruct Flash on how to respond when the user loads or unloads a movie clip, enters a frame, moves the mouse, presses a key, or interacts in some other way with a movie clip. The table that follows lists movie clip handlers and their function.

HANDLER	PURPOSE
`onClipEvent(load)`	Performs the specified actions when the movie clip appears on the Timeline. This is the default handler.
`onClipEvent(unload)`	Performs the specified actions in the first frame after you remove the movie clip from the Timeline.
`onClipEvent(enterFrame)`	Performs the specified actions as a frame plays.
`onClipEvent(mouseMove)`	Performs the specified actions every time the user moves the mouse.
`onClipEvent(mouseDown)`	Performs the specified actions when the user presses the mouse.
`onClipEvent(mouseUp)`	Performs the specified actions when the user releases the mouse.
`onClipEvent(keyDown)`	Performs the specified actions when the user presses any key.
`onClipEvent(keyUp)`	Performs the specified actions when the user releases any key.
`onClipEvent(data)`	Performs the specified actions when ActionScript receives data.

OPERATORS

You use operators to combine, compare, or modify values. The table that follows lists operators and their purpose.

OPERATOR	NAME	PURPOSE
!	Logical Not	Converts an expression that returns true to false and an expression that returns false to true.
!=	Inequality	Used to compare to values. Returns true if two values are not equal. Returns false if two values are equal.
" "	Quotes	Used to enclose strings.
%	Modulo	Returns the remainder of one expression divided by another expression.
&&	Short-Circuit AND	Evaluates two expressions. Returns true if expression 1 and expression 2 return true.
( )	Parenthesis	Groups expressions to control precedence. Encloses arguments.
*	Multiplication	Multiplies expression1 by expression 2.
+	Addition	Adds expression 1 to expression 2.
++	Increment	Adds 1 to a value and returns the original value.
–	Minus	Negates a number or substracts expression 2 from expression 1.
--	Decrement	Substract 1 from a value and returns the original value.
/	Division	Divides expression 1 by expression 2.
<	Less than	Compares two expressions. Returns true if expression 1 is less than expression 2. Returns false if expression 1 is not less than expression 2.
<=	Less than or equal to	Compares two expressions. Returns true if expression 1 is less than or equal to expression 2. Returns false if expression 1 is not less than or equal to expression 2.
<>	Inequality	Used to compare two expressions. Returns true if the values are not equal and false if the values are equal.
==	Equality	Compares two expressions. Returns true if the expressions are equal; returns false if the expressions are not equal.
>	Greater than	Compares two expressions. Returns true if expression 1 is greater than expression 2. Returns false if expression 1 is not greater than or equal to expression 2.
>=	Greater than or equal to	Compares two expressions. Returns true if expression 1 is greater than or equal to expression 2. Returns false if expression 1 is not greater than or equal to expression 2.
and		Evaluates two expressions. Returns true if expression 1 and expression 2 return true. Deprecated in Flash 5.
not		Converts an expression that returns true to false and an expression that returns false to true. Deprecated in Flash 5.
or		Compares two expressions. Returns true if expression 1 returns true or expression 2 returns true. Deprecated in Flash 5.

OPERATOR	NAME	PURPOSE
typeof		Returns whether an expression is a movie clip, object, or function.
void		Returns a null value for an expression.
||	OR	Compares two expressions. Returns true if expression 1 returns true or expression 2 returns true.

FUNCTIONS

A function is a block of script that can be reused. The table that follows lists functions and their purpose.

FUNCTION	PURPOSE
Boolean	Converts a variable, number, or string to a Boolean.
escape	Converts an expression to URL-encoded format.
eval	Returns the value of a variable, property, value, or the reference to an object or movie clip.
false	The Boolean value false.
getProperty	Retrieves properties.
getTimer	Retrieves the number of milliseconds that have elapsed.
getVersion	Retrieves the version of Flash Player on the local computer.
int	Rounds a number to an integer.
isFinite	Returns true if a value is equal to infinity. Returns false if it is not.
isNaN	Returns true if an expression is not a number.
maxscroll	Finds the highest value allowed for the scroll property.

FUNCTION	PURPOSE
newline	Inserts a blank line in ActionScript code.
Number	Converts a string or Boolean to a number.
parseFloat	Converts a string to a floating point number.
parseInt	Converts a string to an integer.
random	Returns a random number.
scroll	Returns the line number of the top line in a text box.
String	Converts a number, Boolean, variable, or object to a string.
targetPath	Returns the target path of a MovieClip object.
true	The Booloean value true.
unescape	Converts an expression in URL-encoded format to ASCII format.
updateAfterEvent	Updates the display after the clip event.

ACTIONS

You use actions to send commands in ActionsScript. You can use actions to start or stop a movie, load or unload a movie, or perform myriad other tasks. The following table lists the actions available to you in ActionScript.

ACTION	PURPOSE
break	Used with the for, for...in, do while, and while actions. The break action instructs ActionSript to break out of a loop.
comment	Enables you to add text to your script. Comments have no effect on your script. Use comments to document your script.
continue	Used with while and do...while loops. With while loops, causes ActionScript to return to the top of the loop. With do...while loops, causes ActionScript to go to the bottom of the loop.
delete	Deletes an object or variable.
duplicateMovieClip	Copies of a movie clip instance.
else	Used with if statements. Executes statements if all other conditions are not met.
evaluate	Creates a blank line with a semi-colon.
for	Used to execute a statement or series of statements repeatedly.
for...in	Loops through object properties or elements in an array.
FSCommand	Sends commands to Flash Player.
function	Defines custom functions.
getUrl	Opens a Web page.
goto	Stops or plays a movie.
if	Evaluates a condition. If the condition is true, executes statements.
ifFrameLoaded	Checks the status of a movie download.
include	Includes script written with a text editor in your Flash movie.

ACTION	PURPOSE
loadMovie	Load and displays several movies at once or switches movies without closing Flash Player.
loadVariables	Loads variables from an external file.
onClipEvent	Enables you to select a movie clip handler.
play	Plays a movie.
print	Prints a movie.
removeMovieClip	Removes a movie clip from the Stage.
return	Specifies the value returned by a function.
set variable	Enables you to assign a value to a variable.
setProperty	Enables you to set the property of a movie clip.
startDrag	Makes an object draggable.
stop	Stops a movie from playing.
stopAllSounds	Stop all sounds that are currently playing.
stopDrag	Stops a drag action.
tellTarget	Sends statements to a Timeline.
toggleHighQuality	Toggles antialiasing on and off.
trace	Displays messages in the Output window.
unloadMovie	Unloads a movie.
var	Declares a local variable.
while	Runs a statement or series of statements repeatedly.
with	Enables you to execute a statement or series of statements on an object or movie clip.

PROPERTIES

This table lists movie clip properties. You can use the `setProperty` action to set many of these properties. You can use the `getProperty` function to retrieve the value of these properties. However, if a property is read-only, you can retrieve but not set the property. Read-only properties are marked with an asterisk.

PROPERTY	PURPOSE
`_alpha`	Returns a value between 0 and 100. A transparent movie clip has a value of 0. An opaque movie clip has a value of 100.
`_currentframe*`	Returns the frame in which the playhead is located.
`_droptarget*`	Returns the absolute path of the movie clip on which the user dropped a draggable instance.
`_focusrect`	Returns true if a yellow rectangle will appear around the button or text box that has focus when the user presses the tab key. Otherwise, it returns false.
`_framesloaded*`	Returns the number of frames loaded.
`_height`	Returns the height of a movie clip in pixels.
`_highquality`	Returns the movie clip quality.
`_name`	Returns the name of a movie clip.
`_quality`	Returns the movie clip quality.
`_rotation`	Returns the rotation value.
`_soundbuftime`	Returns the number of seconds of streaming sound buffered before the movie starts to stream.
`_target*`	Returns the target path of a movie clip instance.

PROPERTY	PURPOSE
`_totalframes*`	Returns the total number of frames in a movie clip.
`_url*`	Returns the URL from which the movie clip was downloaded.
`_visible`	Returns the visibility status. Returns `true` if the movie clip is visible. Returns `false` if the movie clip is not visible.
`_width`	Returns the width of the movie clip.
`_x`	Returns the x coordinate of a movie clip, using global coordinates.
`_xmouse*`	Returns the x coordinate of the mouse position.
`_xscale`	Returns the amount of horizontal scaling that the script or the user has applied to a movie clip.
`_y`	Returns the y coordinate of a movie clip, using global coordinates.
`_ymouse*`	Returns the y coordinate of the mouse position.
`_yscale`	Returns the amount of vertical scaling that the script or the user has applied to a movie clip.

OBJECTS

ARRAY OBJECT

You can use an array to group related data. The table that follows lists the methods you can use with the Array object.

METHOD	PURPOSE
`concat`	Concatenates and returns an array.
`join`	Joins elements in an array to form a string.
`pop`	Removes the last element from an array.
`push`	Adds elements to the end of an array.
`reverse`	Reverses an array.
`shift`	Removes the first element from an array.

METHOD	PURPOSE
`slice`	Extracts elements from an array.
`sort`	Sorts an array.
`splice`	Adds or removes elements from an array.
`toString`	Converts an array to a string.
`unshift`	Adds elements to the beginning of the array.
`length`	Returns the length of an array.

BOOLEAN OBJECT

You can use the `Boolean` object to find the string value of a Boolean or to return the primitive value of the `Boolean` object. The following table lists the methods of the `Boolean` object.

METHOD	PURPOSE
`toString`	Returns the string value of a Boolean object.
`valueOf`	Returns the primitive value of a Boolean object.

COLOR OBJECT

You can use the `Color` object methods to retrieve and set the color values. The table that follows lists the methods of the `Color` object and their purpose.

METHOD	PURPOSE
`getRGB`	Returns the value of the last setRGB call.
`getTransform`	Returns the value of the last setTansform call.
`setRGB`	Sets the hexadecimal value for a Color object.
`setTransform`	Sets the color transform for a Color object.

DATE OBJECT

You can use the `Date` object to retrieve and set the date. The table that follows lists the methods and purposes of the `Date` object.

METHOD	PURPOSE
`getDate`	Returns the day of the month in local time.
`getDay`	Returns the day of the week in local time.
`getFullYear`	Returns the four-digit year in local time.
`getHours`	Returns the hour in local time.
`getMilliseconds`	Returns the millisecond in local time.
`getMinutes`	Returns the minute in local time.
`getMonth`	Returns the month in local time.
`getSeconds`	Returns the second in local time.
`getTimer`	Returns the number of milliseconds since midnight January 1, 1970 universal time.
`getTimezoneOffset`	Returns the difference between local time and universal time.
`getUTCDate`	Returns the day of the month in universal time.
`getUTCDay`	Returns the day of the week in universal time.
`getUTCFullYear`	Returns the four-digit year in universal time.
`getUTCHours`	Returns the hour in universal time.
`getUTCMilliseconds`	Returns the millisecond in universal time.
`getUTCMinutes`	Returns the minute in universal time.
`getUTCMonth`	Returns the month in universal time.

METHOD	PURPOSE
`getUTCSeconds`	Returns the second in universal time.
`getYear`	Returns the year in local time.
`setDate`	Sets the day of the month in local time.
`setFullYear`	Sets the year in local time.
`setHours`	Sets the hour in local time.
`setMilliseconds`	Sets the millisecond in local time.
`setMinutes`	Sets the minute in local time.
`setMonth`	Sets the month in local time.
`setSeconds`	Sets the second in local time.
`setTime`	Sets the time in milliseconds.
`setUTCDate`	Sets the date in universal time.
`setUTCFullYear`	Sets the year in universal time.
`setUTCHours`	Sets the hour in universal time.
`setUTCMilliseconds`	Sets the milliseconds in universal time.
`setUTCMinutes`	Sets the minute in universal time.
`setUTCMonth`	Sets the month in universal time.
`setUTCSeconds`	Sets the second in universal time.
`setYear`	Sets the year in local time.
`toString`	Returns the date and time as a string.
`Date.UTC`	Returns the number of milliseconds between midnight January 1, 1970, universal time, and a specified time.

KEY OBJECT

You can use the methods of the `Key` object to enable users to manipulate objects with keys or to respond to key presses. The following table lists the methods of the `Key` object and their purpose.

METHOD	PURPOSE
`getAscii`	Returns the ASCII code for the last key pressed.
`getCode`	Returns the virtual key code for the last key pressed.
`isDown`	Returns true if a specified key is pressed.
`isToggled`	Returns true if Num Lock or Caps Lock are on.

MATH OBJECT

You can use the methods of the Math object to manipulate numbers and perform mathematical calculations. The following table lists the methods of the `Math` object and their purpose.

METHOD	PURPOSE
`abs`	Returns the absolute value of a value.
`acos`	Returns the arc cosine of a value.
`asin`	Returns the arc sine of a value.
`atan`	Returns the arc tangent of a value.
`atan2`	Computes the angle from the x-axis to the point.
`ceil`	Rounds a number up.
`cos`	Returns the cosine of a value.
`exp`	Computes an exponential value.
`floor`	Rounds a number down.
`log`	Returns the natural logarithm of a value.

METHOD	PURPOSE
`max`	Compares two numbers and returns the larger.
`min`	Compares two numbers and returns the smaller.
`pow`	Raises a number to a power.
`random`	Returns a random number.
`round`	Rounds a number.
`sin`	Returns the sine of a value.
`sqrt`	Returns the square root of a value.
`tan`	Returns the tangent of a value.

MOUSE OBJECT

You can use the methods of the `Mouse` object to show or hide the mouse. The table that follows lists the methods of the `Mouse` object and their purpose.

METHOD	PURPOSE
`hide`	Hides the cursor.
`show`	Shows the cursor.

MOVIECLIP OBJECT

You can use the methods of the MovieClip object to manipulate movie clips. The table that follows lists the methods of the Mouse object and their purpose.

METHOD	PURPOSE
`attachMovie`	Attaches a movie clip in the Library and places it on the Stage.
`duplicateMovieClip`	Duplicates a movie clip.
`getBounds`	Returns the boundaries of a movie clip.
`getBytesLoaded`	Returns the number of bytes loaded.
`getBytesTotal`	Returns the size of a movie clip.
`getURL`	Opens a Web page.
`globalToLocal`	Converts the Stage coordinates to local coordinates.
`gotoAndPlay`	Begins to play a movie clip in the frame specified.
`gotoAndStop`	Sends the playhead to a frame and stops the movie.
`hitTest`	Returns true if a movie clip touches or overlaps with another movie clip.
`loadMovie`	Loads a movie into a movie clip.

METHOD	PURPOSE
`loadVariables`	Loads variables.
`localToGlobal`	Converts the Stage coordinates to global coordinates.
`nextFrame`	Sends the playhead to the next frame in the movie clip.
`prevFrame`	Sends the playhead to the previous frame in the movie clip.
`removeMovieclip`	Removes movie clips created with duplicateMovieClip and attachMovie from the Timeline.
`startDrag`	Makes a movie clip draggable.
`stop`	Stops a movie clip.
`stopDrag`	Stops the drag actions.
`swapDepths`	Swaps the depth level of a movie clip.
`unloadMovie`	Removes a movie that was loaded using loadMovie.

NUMBER OBJECT

You can use the `Number` object to find the string value of a number or to return the primitive value of the Number object. The following table lists the methods of the `Number` object.

METHOD	PURPOSE
`toString`	Returns a number as a number.
`valueOf`	Returns the primative value of a number.

OBJECT OBJECT

You can use the `Object` object to find the string value of an object or to return the primitive value of the `Object` object. The following table lists the methods of the `Object` object.

METHOD	PURPOSE
`toString`	Returns the object as a string.
`valueOf`	Returns the primative value of an Object object.

SELECTION OBJECT

You can use the methods of the `Selection` object to control the text box that currently has focus. The following table lists the methods of the `Selection` object and their purpose.

METHOD	PURPOSE
`getBeginIndex`	Returns the index value of the beginning of a selection pan.
`getCaretIndex`	Returns the index value of the blinking cursor.
`getEndIndex`	Returns the index of the end of a selection pan.
`getFocus`	Returns the name of the variable for the text box that currently has focus.
`setFocus`	Sets the focus to specified text box.
`setSelection`	Sets the selection span.

SOUND OBJECT

You can use the methods of the `Sound` object to control sound. The table that follows lists the `Sound` methods and their purpose.

METHOD	PURPOSE
`attachSound`	Attaches a sound.
`getPan`	Returns the pan value.
`getTransform`	Returns the sound transform value.
`getVolume`	Returns the volume value.
`setPan`	Sets the pan value.
`setTransform`	Sets the transform value.
`setVolume`	Sets the volume.
`start`	Starts a sound.
`stop`	Stops a sound.

STRING OBJECT

You can use the `String` object to manipulate strings. The table that follows lists the methods of the `String` object and their purpose.

METHOD	PURPOSE
`charAt`	Returns the character at a specified index position.
`charCodeAt`	Returns the numeric value of a character at a specified index position.
`concat`	Concatenates strings.
`fromCharCode`	Returns the character assigned to a code.
`indexOf`	Returns the first occurrence of the index value of a substring.
`lastIndexOf`	Returns the last occurrence of the index value of a substring.
`slice`	Returns a substring of a string.
`substr`	Returns a substring of a string. You specify the starting position and length.
`substring`	Returns a substring of a string. You specify the starting and ending position.
`toLowerCase`	Converts a string to lowercase.
`toUpperCase`	Converts a string to uppercase.

APPENDIX

WHAT'S ON THE CD-ROM

The CD-ROM disc included in this book contains many useful files and programs that can be used when working with Flash. You will find files that contain all the sample code used in this book, as well as several popular programs you can install on your computer. Before installing any of the programs on the disc, make sure that a newer version of the program is not already installed on your computer. For information on installing different versions of the same program, contact the program's manufacturer.

SYSTEM REQUIREMENTS

While most programs on the CD-ROM disc have minimal system requirements, your computer should be equipped with the following hardware and software to make the best use of all the contents of the CD-ROM disc:

- A Pentium processor running Windows 95 or later or Windows NT 4 or later
- At least 32 MB of available RAM
- At least 40 MB of available disk space
- Microsoft Internet Explorer 4.0 or later, or Netscape Navigator 4.0 or later recommended
- 800 x 600 color display
- CD-ROM drive

AUTHOR'S SOURCE CODE

For Windows 2000. The CD provides files that contain all the sample code used throughout this book. You can browse these files directly from the CD-ROM, or you can copy them to your hard drive and use them as the basis for your own projects. To find the files on the CD-ROM, open the D:\RESOURCES\CODE folder. To copy the files to your hard drive, just run the installation program D:\RESOURCES\CODE.EXE. The files will be placed on your hard drive at C:\ProgramFiles\FlashActionScript. After installing, you can access the files from the Start menu. You will need Flash 5 installed on the machine to run the samples. Please see Using Creative Techniques in Chapter 12 for more information.

ACROBAT VERSION

The CD-ROM contains an e-version of this book that you can view and search using Adobe Acrobat Reader. You can also use the hyperlinks provided in the text to access all Web pages and Internet references in the book. You cannot print the pages or copy text from the Acrobat files. If you do not currently have Adobe Acrobat Reader 5 installed, the computer will prompt you to install the software Acrobat files. A freeware version of Adobe Acrobat Reader is also included on the disc.

INSTALLING AND USING THE SOFTWARE

This CD-ROM disc contains several useful programs. Before installing a program from the CD, you should exit all other programs. In order to use most of the programs, you must accept the license agreement provided with the program. Make sure you read any ReadMe files provided with each program.

Program Versions

Shareware programs are fully functional, free trial versions of copyrighted programs. If you like a particular program, you can register with its author for a nominal fee and receive licenses, enhanced versions, and technical support.

Freeware programs are free, copyrighted games, applications, and utilities. You can copy them to as many computers as you like, but they have no technical support.

GNU software is governed by its own license, which is included inside the folder of the GNU software. There are no restrictions on distribution of this software. See the GNU license for more details.

Trial, demo, or evaluation versions are usually limited either by time of functionality. For example, you may not be able to save projects using these versions.

For your convenience, the software titles on the CD are listed in alphabetic order.

Acrobat Reader

For Mac and Windows. Freeware. Acrobat Reader lets you view the online version of this book. For more information

on using Adobe Acrobat Reader, see page 296. From Adobe Systems, Inc., www.adobe.com.

Adobe Photoshop
For Mac and Windows. Trial Version. Enables you to create, edit, or retouch images. From Adobe Systems, Inc., www.adobe.com.

Dreamweaver 4
For Mac and Windows. Trial version. Enables you to quickly and easily develop HTML-based Web pages. From Macromedia, Inc., www.macromedia.com.

Dreamweaver UltraDev 4
For Mac and Windows. Trial Version. Enables you to develop Web pages that include ASP, JSP, or ColdFusion. From Macromedia, Inc., www.macromedia.com.

Flash 5
For Mac and Windows. Trial Version. Enables you to create interactive multimedia Web pages. From Macromedia, Inc., www.macromedia.com.

Flash Player
For Mac and Windows. Commercial version. Enables you to view Macromedia Flash content. From Macromedia, Inc., www.macromedia.com.

FreeHand 10
For Mac and Windows. Trial Version. Enables you to create vector-based illustrations for Flash and for print. From Macromedia, Inc., www.macromedia.com.

Paint Shop Pro
For Windows. Evaluation Version. Enables you to create, edit, or retouch images. From JASC Software, Inc., www.jasc.com/.

Director 8.5 Shockwave Studio
For Mac and Windows. Commercial version. Enables you to create multimedia content, including advanced 3D games. From Macromedia, Inc., www.macromedia.com.

Shockwave
For Mac and Windows. Trial Version. Enables you to create multimedia content, including advanced 3D games. From Macromedia, Inc., www.macromedia.com.

Stuffit Expander
For Mac. Commercial version. Enables users to access all downloads and attachments. From Aladdin Systems, www.aladdinsys.com.

Stuffit Lite
For Mac. Shareware. Enables you to compress files and open compressed files. From Aladdin Systems, www.aladdinsys.com/.

WinZip
For Windows. Shareware. Enables you to compress files and open compressed files. From Nico Mak Computing, Inc., www.winzip.com/.

TROUBLESHOOTING

We tried our best to compile programs that work on most computers with the minimum system requirements. Your computer, however, may differ and some programs may not work properly for some reason.

The two most likely problems are that you don't have enough memory (RAM) for the programs you want to use, or you have other programs running that are affecting installation or running of a program. If you get error messages like `Not enough memory` or `Setup cannot continue`, try one or more of these methods and then try using the software again:

- Close all running programs.
- Restart your computer.
- Turn off any anti-virus software.
- Close the CD-ROM interface and run demos or installations directly from Windows Explorer.
- Add more RAM to your computer.

If you still have trouble installing the items from the CD-ROM, please call the Hungry Minds Customer Service phone number: 800-762-2974 (outside the U.S.: 317-572-3994), or e-mail techsupdum@hungryminds.com.

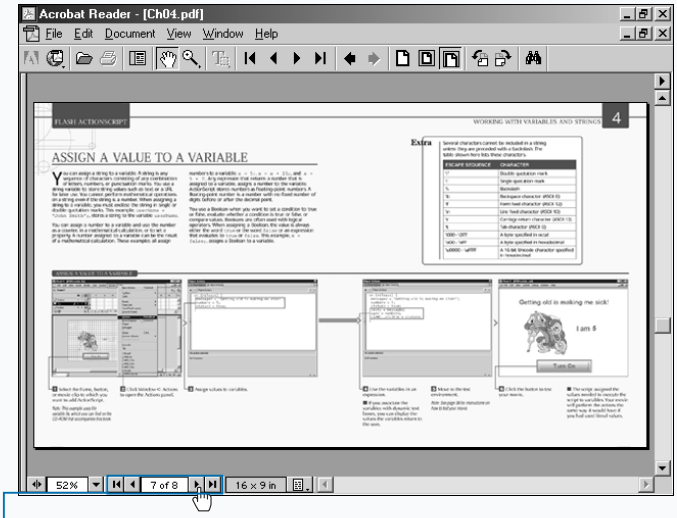
USING THE E-VERSION OF THE BOOK

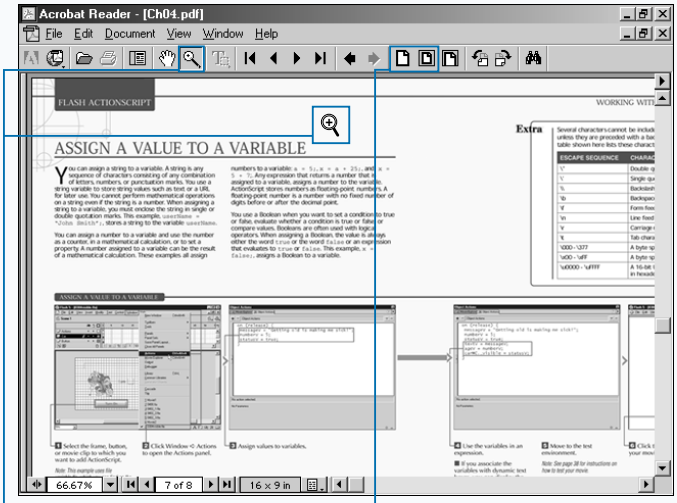
You can view *Flash ActionScript: Your visual blueprint for creating Flash-enhanced Web sites* on your screen using the CD-ROM included at the back of this book. The CD-ROM allows you to search the contents of each chapter of the book for a specific word or phrase. The CD-ROM also provides a convenient way of keeping the book handy while traveling.

You must install Adobe Acrobat Reader on your computer before you can view the book on the CD-ROM. This program is provided on the disc. Acrobat Reader allows you to view Portable Document Format (PDF) files, which can display books and magazines on your screen exactly as they appear in printed form.

To view the contents of the book using Acrobat Reader, insert the CD-ROM into your drive. The autorun interface will appear. Navigate to the eBook, and open the book.pdf file. You may be required to install Acrobat Reader 5.0 on your computer, which you can do by following the simple installation instructions. If you choose to disable the autorun interface, you can open the CD root menu and open the Resources folder, then open the eBook folder. In the window that appears, double-click the eBook.pdf icon.

USING THE E-VERSION OF THE BOOK





FLIP THROUGH PAGES

1 Click one of these options to flip through the pages of a section.

- First page
- Previous page
- Next page
- Last page

ZOOM IN

1 Click to magnify an area of the page.

2 Click the area of the page you want to magnify.

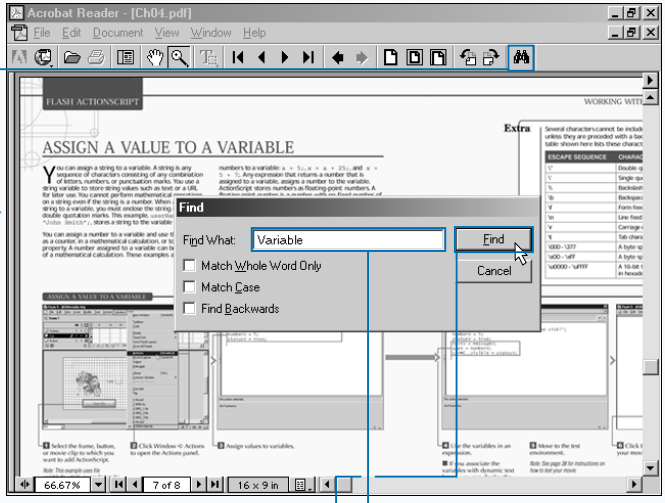
- Click one of these options to display the page at 100% magnification ( ) or to fit the entire page inside the window ( ).

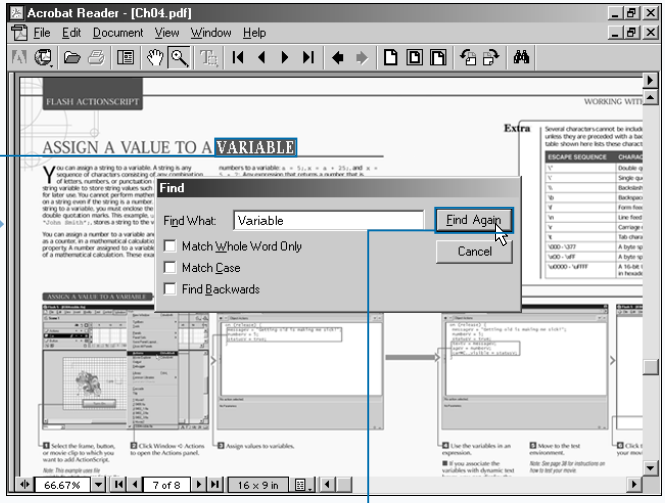
Extra

To install Acrobat Reader, insert the CD-ROM disc into a drive. In the screen that appears, click Software. Click Acrobat Reader and then click Install at the bottom of the screen. Then follow the instructions on your screen to install the program.

You can make searching the book more convenient by copying the .pdf files to your own computer. Display the contents of the CD-ROM disc and then copy the PDFs folder from the CD to your hard drive. This allows you to easily access the contents of the book at any time.

Acrobat Reader is a popular and useful program. There are many files available on the Web that are designed to be viewed using Acrobat Reader. Look for files with the .pdf extension. For more information about Acrobat Reader, visit the Web site at www.adobe.com/products/acrobat/readermain.html.





FIND TEXT

1 Click to search for text in the section.

■ The Find dialog box appears.

2 Type the text you want to find.

3 Click Find to start the search.

■ The first instance of the text is highlighted.

4 Click Find Again to find the next instance of the text.

HUNGRY MINDS, INC. END-USER LICENSE AGREEMENT

READ THIS. You should carefully read these terms and conditions before opening the software packet(s) included with this book ("Book"). This is a license agreement ("Agreement") between you and Hungry Minds, Inc. ("HMI"). By opening the accompanying software packet(s), you acknowledge that you have read and accept the following terms and conditions. If you do not agree and do not want to be bound by such terms and conditions, promptly return the Book and the unopened software packet(s) to the place you obtained them for a full refund.

1. License Grant. HMI grants to you (either an individual or entity) a nonexclusive license to use one copy of the enclosed software program(s) (collectively, the "Software") solely for your own personal or business purposes on a single computer (whether a standard computer or a workstation component of a multi-user network). The Software is in use on a computer when it is loaded into temporary memory (RAM) or installed into permanent memory (hard disk, CD-ROM, or other storage device). HMI reserves all rights not expressly granted herein.

2. Ownership. HMI is the owner of all right, title, and interest, including copyright, in and to the compilation of the Software recorded on the disk(s) or CD-ROM ("Software Media"). Copyright to the individual programs recorded on the Software Media is owned by the author or other authorized copyright owner of each program. Ownership of the Software and all proprietary rights relating thereto remain with HMI and its licensers.

3. Restrictions On Use and Transfer.

(a) You may only (i) make one copy of the Software for backup or archival purposes, or (ii) transfer the Software to a single hard disk, provided that you keep the original for backup or archival purposes. You may not (i) rent or lease the Software, (ii) copy or reproduce the Software through a LAN or other network system or through any computer subscriber system or bulletin-board system, or (iii) modify, adapt, or create derivative works based on the Software.

(b) You may not reverse engineer, decompile, or disassemble the Software. You may transfer the Software and user documentation on a permanent basis, provided that the transferee agrees to accept the terms and conditions of this Agreement and you retain no copies. If the Software is an update or has been updated, any transfer must include the most recent update and all prior versions.

4. Restrictions on Use of Individual Programs. You must follow the individual requirements and restrictions detailed for each individual program in the What's on the CD-ROM appendix of this Book. These limitations are also contained in the individual license agreements recorded on the Software Media. These limitations may include a requirement that after using the program for a specified period of time, the user must pay a registration fee or discontinue use. By opening the Software packet(s), you will be agreeing to abide by the licenses and restrictions for these individual programs that are detailed in the What's on the CD-ROM appendix and on the Software Media. None of the material on this Software Media or listed in this Book may ever be redistributed, in original or modified form, for commercial purposes.

5. Limited Warranty.

(a) HMI warrants that the Software and Software Media are free from defects in materials and workmanship under normal use for a period of sixty (60) days from the date of purchase of this Book. If HMI receives notification within the warranty period of defects in materials or workmanship, HMI will replace the defective Software Media.

(b) HMI AND THE AUTHOR OF THE BOOK DISCLAIM ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE, THE PROGRAMS, THE SOURCE CODE CONTAINED THEREIN, AND/OR THE TECHNIQUES DESCRIBED IN THIS BOOK. HMI DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE ERROR FREE.

(c) This limited warranty gives you specific legal rights, and you may have other rights that vary from jurisdiction to jurisdiction.

6. Remedies.

(a) HMI's entire liability and your exclusive remedy for defects in materials and workmanship shall be limited to replacement of the Software Media, which may be returned to HMI with a copy of your receipt at the following address: Software Media Fulfillment Department, Attn.: ***Flash ActionScript: Your visual blueprint for creating Flash-enhanced Web sites***, Hungry Minds, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, or call 1-800-762-2974. Please allow four to six weeks for delivery. This Limited Warranty is void if failure of the Software Media has resulted from accident, abuse, or misapplication. Any replacement Software Media will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

(b) In no event shall HMI or the author be liable for any damages whatsoever (including without limitation damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising from the use of or inability to use the Book or the Software, even if HMI has been advised of the possibility of such damages.

(c) Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation or exclusion may not apply to you.

7. U.S. Government Restricted Rights. Use, duplication, or disclosure of the Software for or on behalf of the United States of America, its agencies and/or instrumentalities (the "U.S. Government") is subject to restrictions as stated in paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013, or subparagraphs (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, and in similar clauses in the NASA FAR supplement, as applicable.

8. General. This Agreement constitutes the entire understanding of the parties and revokes and supersedes all prior agreements, oral or written, between them and may not be modified or amended except in a writing signed by both parties hereto that specifically refers to this Agreement. This Agreement shall take precedence over any other documents that may be in conflict herewith. If any one or more provisions contained in this Agreement are held by any court or tribunal to be invalid, illegal, or otherwise unenforceable, each and every other provision shall remain in full force and effect.

INDEX

A

absolute target path, 224–225
access [ ] operator, arrays, 162–163
Actions panel
  Colored Syntax colors, 27
  comments, adding, 42–43
  Expert mode, 24–27
  fonts, sizing, 27
  #include statement, 25
  line, moving to, 27
  modes, switching between, 26
  movies, quality settings, 50–51
  Normal mode, 24–27
  open, 270
  playhead, positioning, 48–49
  scripts, printing, 27
  Show Deprecated Syntax, 27
  statements, 24–25, 27, 37
  string characters, find/replace, 27
  strings, search, 27
  syntax errors, checking, 27
  Toolbox list, 24–27
actions reference, 286
Actions statement, 24
actions
  appended with NUM, 280
  argument association, 36
  buttons, ActionScript definitions, 12
  test, 38
ActionScript
  Actions panel, 24–27
  argument, 36–37
  AS file extension, 25
  buttons
    action definitions, 12
    assignment, 28
    association, 6
  code blocks, curly braces {and} characters, 36–37
  comments, 42–43
  expressions, 37
  frames, assignment, 30–31
  handlers, 28–29
  #include statement, 25
  keywords, case sensitive, 36
  movie clips
    association, 6
    assignment, 29
    referencing by name, 19
  movies, adding, 244
  predefined objects, 144
  smart clips, 18, 236–239
  statements, 24–27, 36–37
  static images, non-support, 8
  view, 275
active layer, 4–5
Add a Statement button, 24, 26
Add Scene icon, 3
add
  comments, 42–43
  numeric values, 112–113
  scenes, 3
  sounds, 16–17
addition (+) operator, 88, 100, 110
adjust, transparency, 68–69
alignments, 20–21
alpha, instances, 10–11
ampersand (&) character, bitwise AND, 100
AND (&&) operator, 132–133
animated buttons, 12, 19
animations
  buttons, 8
  frame-by-frame, 14
  movie clips
    conversion, 19
    reusable, 8
  preloader, 62
  QuickTime, 32
  start/end keyframes, 14
  tween adjustments, 15
  tweened, 14
antialiasing
  movie clip quality, 86–87
  movie quality, 50–51
  smoothing text, 21
arguments
  comma (,) character as separator, 36
  defined, 36
  parentheses (and) characters, 36
  statements, 24
  value assignments, 36

arrays
- access operator [ ], 162–163
- comma (,) character as separator, 92
- conversions, string, 174–175
- create, 162–165
- data types, 93
- described, 162
- display
  - changes, 13
  - values, 273
- elements
  - adding, 168–169
  - assigning, 164–165
  - number (index), 162–163
  - removing, 170–171
  - value changes, 164–165
- empty, 164–165
- extract, 172–173
- length, finding, 166–167
- reverse, 172–173

arrow line, tween complete indicator, 15
Arrow tool, 6–7, 9, 11
AS file extension, 25
ASCII characters, URL-encoded conversion, 95
ASCII values, 198–200
assign, target paths, 224–225
assignments
- ActionScript, 28–31
- variable values, 90–93

associations
- buttons, ActionScript, 6
- dynamic text box/variables, 23
- input text box/variables, 22
- movie clips, ActionScript, 6
- operator precedence, 111

asterisk (*) character, 22, 88, 116–117
asterisk (*) slash (/) characters, comments, 42
attach, movie clips, 148–151

B

background images, symbol advantages, 9
backgrounds, 22–23
backslash (\) character, escape sequences, 91
Bandwidth Profiler, movie information display, 40–41
base 16, hexadecimal numbers, 177
base, numbers, 215
Basic Actions statement, 24
behaviors
- instances, 11
- movie clips, selecting, 18
- symbols, 9

BEST quality, movie clips, 86–87
binary operators, operands, 110
bitmap images, movies, printing as, 60
Boolean
- conversions, 108–109
- data type, 88
- primitive data type, 92
- values, false, 56

borders, 22–23
boundary
- movie clip, 152–153
- movie display, 33

brackets [and] characters, array access operator, 162–163
brightness, instances, 10–11
broken line, tween start indicator, 15
Brush tool, size/shape, 6
buttons
- ActionScript, 28
- animated, create, 19
- antialiasing on/off, 50–51
- associations, ActionScript, 6
- check mark beside, 13
- convert to movie clip, 252
- enable/disable, 13
- execute continuous action, 252
- four-frame Timeline, 12
- handlers, 252, 283
- images, importing, 13
- instance behavior type, 11
- interactive graphics, 8–9
- keyframe, inserting, 13
- movie clips, converting to panels, 260
- movie startup point, 46
- objects, dragging, 259
- rotate continuously, 255
- states, 12–13
- stop action assignment, 44
- tests, 13, 38
- Timeline, 8–9

C

Caps Lock key, Status bar display, 41
case sensitive, search limitation, 249

case, strings, changing, 106–107
CGI scripts, load/unload movies, 228–229
Change the Statement Order button, 24–25
change, colors, 176–177
characters
  string, extracting, 96–97
  URL encoding, 278–279
check, frame load, 62–63
check mark, 13, , 27
child movie clips, parent placement, 234
circles, Oval tool, 6
Clip Parameters dialog box, 243
Clip Parameters panel, smart clips, 238–241
clocks, 184–185
code
  blocks
    comments, 42–43
    curly braces {and} characters, 36–37
  document with comments, 274
  hexadecimal color, 280–282
  panel size changes, 261
collision, detection, 158–159
color boxes, tools, stroke/fill, 6–7
Color object, movie clips, color change, 176–177
Colored Syntax, 27, 274
coloring book, create, 256–257
colors
  add to objects, 256–257
  Colored Syntax, 27
  comments, 27
  deprecated syntax, 27
  fill, 7
  fonts, 20–21
  hexadecimal codes, 280–282
  hexadecimal values, 176–177
  instances, 10–11
  keywords, 27
  magenta, comments, 42
  mouse, change, 257
  movie clips, changing, 176–177
  predefined identifiers, 27
  properties, 27
  quoted strings, 27
  statements, 27
  stroke, 7
  transform values, 178–179
columns, 5
comma (,) character
  argument separator, 36
  array separator, 92
commands
  Control
    Debug Movie, 268
    Loop Playback, 38
    Mute Sounds, 38–39
    Play, 38
    Play All Scenes, 38
    Rewind, 38
    Step Backward, 38
    Step Forward, 38
    Test Movie, 38–40, 43, 261, 272–273
    Test Scene, 38–39
  Debug
    List Objects, 272
    List Variables, 273
  Edit
    Copy Frames, 18
    Deselect All, 19
    Edit Movie, 9
    Edit Symbols, 9, 253
    Paste Frames, 19
    Select All, 19
  File
    Import, 13, 16
    Print, 61
    Publish Preview, Default, 39
    Publish Settings, 32, 50
    Save, 38, 149
  Insert
    Convert to Symbol, 9, 12, 219
    Create Motion Tween, 14
    Frame, 15
    Keyframe, 13, 30
    Layer, 4, 30
    New Symbol, 8, 12–13, 18, 240
  Modify
    Frame, 15
    Instance, 11
    Layer, 5
    Transform, Edit Center, 73
    Transform, Rotate, 73
    Transform, Scale, 79
  Options, Define Clip Parameters, 237

Text
Character, 53
Options, 20–21
Paragraph, 21
View
Antialias, 21
GoTo, 3
High Quality, 50
Magnification, 7
Quality, 41
Scene, 3
Show Streaming, 41
Zoom In, 7, 41
Zoom Out, 7, 41
Window
Actions, 28–29, 31, 36, 41, 244, 246
Common Libraries, 237
Common Libraries, Learning Interactions, 241, 243
Controller (Mac), 39
Debug, 268
Library, 11, 237, 241–242
Movie Explorer, 222, 275
Output, 270
Panels,
Clip Parameters, 239, 243
Effect, 11, 69
Fill, 7
Frame, 43
Instance, 11, 19, 66, 242
Mixer, 7
Scene, 3
Sound, 16
Stroke, 7
Text Options, 88
Toolbar, 41
Toolbar, Controller (PC), 39
Tools, 7
comments
colors, 27
document code, 274
magenta color, 42
multiline, 42–43
single-line, 42–43
common library, smart clips, 259
communicate, Flash Player, 54–55
compound assignment operators, 130–131
compression, 16, 34
concatenate, strings, 100–101
conditional (?) operator, ternary type, 110
conditional checking, date validation, 246–247
conditional loops, 140–141
conditions, test, 134–135
constructors
new Color, 176
new Date, 182–183, 186
new Sound, 202
newArray, 162, 166
continuous actions, start/stop, 252
Control menu, movies, testing, 38–39
Controller Toolbar, access methods, 39
convert
array to string, 174–175
Booleans, 108–109
numbers, 108–109
string to number, 214–215
strings, 108–109
symbols, 252
coordinates, 160–161, 180
copy, selected objects, 6
counters, loops, 122
create
animated buttons, 19
animations, 14–15
array, 162–165
buttons, trigger actions, 252–253
clocks, 184–185
coloring book, 256–257
colors, 7
custom functions, 220–221
draggable objects, 56–59
dynamic text box, 23
forms, 22
frame label, 31
gradients, 7
graphics, 6
input text boxes, 22
instance, 10–11
keyframes, 30
layers, 4–5, 30
menus, getURL action, 53
motion guide layers, 4
motion tween, 4
movie clips, 18–19
movies, one-frame, 252

objects, draggable, 258
rotation effects, 254–255
scenes, 2–3
scrollable text box, 218–219
smart clips, described, 236
sounds, 202–203
static text boxes, 20–21
symbol, 8–9
trigonometric special effects, 266–267
crop, movies, 33
curly braces {and} characters, code blocks, 36–37
curves, Pen tool, 6
custom functions, create, 220–221
custom interface, smart clips, 239–243

D

data types
array, 93
Boolean, 88
movie clip, 88
number, 88
object, 88
primitive, 92
reference, 92
string, 88
data, validate, 244–245
dates
current, retrieving, 183
set, 186–187
update, 189
validate, 246–247
values, 188–189
retrieving, 184–185
days, validation, 246–247
debug, script, 274–275
Debugger
display list, 223, 268
error correction, 268–269
movies, activating, 32
password protect movies, 269
Properties tab, changing property values, 268–269
status bar, 268
Variables tab, 268–269
Watch list, add/remove variables, 268–269
decimal places, round numbers, 194
decimal point (.), floating point numbers, 119
decrement (—) operator
mathematical data type, 88
unary type, 110
set values, 124–125
Define Clip Parameters dialog box, 236–239, 243
delete
layers, 4–5
selected objects, 6
statements, 24–25
Delete a Statement button, 25
Delete Scene icon, 3
deprecated syntax, display color, 27
depths, swap, 154–155
detect, collision, 158–159
device fonts
movies, 34
static text boxes, 20–21
display list
Debugger, levels/instances, 268
Flash Player, 222–223
divide, numeric values, 118–119
division (/) operator
mathematical data type, 88
numeric values, dividing, 118–119
dot syntax, target path, 224–225
double hyphen (—) character, decrement operator, 88, 110, 124–125
double plus sign (++), increment operator, 88
double slashes (//) characters, single-line comments, 42
Down state, buttons, 12–13
downloads, frames, checking, 62–63
draggable objects
create, 56–59
handlers, 58–59
scroll bars, 57
sound volume/pan, 57
Dropper tool, fill/stroke attributes, 6–7
duplicate
movie clips, 234–235
scenes, 3
Duplicate Scene icon, 3
dynamic text
invalid data entry, 244
variables, 88
dynamic text boxes, 23, 208–209

E

editable text boxes, 20–21
Effect panels, instances
  adjustments, 11
  _alpha property changes, 69
effects
  sounds, 16
  spotlight, mask layers, 4
  tweens, fading, 69
electronic coloring book, create, 256–257
elements, array
  adding, 168–169
  assigning, 164–165
    removing, 170–171
ellipses, Oval tool, 6
embedded fonts, static text boxes, 20–21
emulate panels, 258–261
Enable Simple Buttons, 38
Enable Simple Frame Actions, 38
equal sign (=) character, 65, 88
equality (==) operator, 128–129
equations, mathematical, 37
Eraser tool, 6
error messages
  date validation, 247
  Options menu, Output Window, 270
  to user, 245
errors, mathematical evaluation, 216–217
escape sequence characters, strings, 91
event sounds, 16
Event synchronization, sounds, 16
Events, ActionScript, handlers, 28–29
Expert mode
  Actions panel, 24–27
  syntax checks, 275
exponentials, mathematical functions, 192
expression argument, 270
expressions
  argument, 36
  convert, 108–109
  defined, 37, 228
  jumps, 48
  mathematical equations, 37
  operators
    AND (&&), 132–133
    compound assignment, 130–131
    equality (==), 128–129
    greater than (>), 126
    greater than or equal to (>=), 127
    inequality (!=), 128–129
    less than (<), 126
    less than or equal to (<=), 127
    logical, 132–133
    NOT (!), 132–133
    OR (||), 132–133
    post-decrement (—), 124–125
    post-increment (++), 122–123
    pre-decrement (—), 124–125
    pre-increment (++), 122–123
  property, 37
  _rotation = _rotation + x, 254
  _rotation = _xmouse, 254
  variable, 37
  variable assignments, 88
external file, loadVariables action, 94
extract
  arrays, 172–173
  string
    characters, 96–97
    substring, 97–98

F

fades, tween effect, 69
faucet, Eraser tool, 6
fields, determining user use, 245
file extensions, 25, 39
file formats, movie supported types, 32
files
  CD-ROM
    1key.fla, 198
    2Key.fla, 200
    add.fla, 112
    alpha.fla, 68
    attachMC.fla, 148
    birth.fla, 246
    case.fla, 106
    chick.fla, 252
    clock.fla, 184
    color.fla, 176, 256
    colorsc.fla, 240
    colortransform.fla, 178
    compound.fla, 130
    concat.fla, 100
    createarray.fla, 162

createarray1.fla, 164
date.fla, 182, 186
decrement.fla, 124
divide.fla, 118
duplicate.fla, 234
elseif.fla, 138
equality.fla, 128
eval.fla, 213
extract.fla, 96
for.fla, 142
fsc.fla, 54
function.fla, 220
getbounds.fla, 152
getproperties.fla, 210
height2.fla, 76
hittest.fla, 158
if.fla, 134
ifelse.fla, 136
increment.fla, 122
joinarray.fla, 174
jump.fla, 49
length.fla, 102, 166
LessThan.fla, 126
LessThanE.fla, 127
load.fla, 226, 228
localtoglobal.fla, 160
logical.fla, 132
maxmin.fla, 196
modulo.fla, 121
mouse.fla, 180
multiply.fla, 116
object.fla, 92, 144
panel.fla, 258
path.fla, 224
poparray.fla, 170
pusharray.fla, 168
quality.fla, 50, 86
random.fla, 195
rotate.fla, 72, 254
round.fla, 194
sales.txt, 94
scroll.fla, 218
search.fla, 104, 248
selections.fla, 208
setdate.fla, 188
settime.fla, 190
slicearray.fla, 172
smartclip.fla, 236
smartclip2.fla, 242
sound.fla, 202
sqr.fla, 193
substring.fla, 97
subtract.fla, 114
swap.fla, 154
telltarget.fla, 230
tonumber.fla, 214, 216
trig.fla, 192
trig_1.fla, 262
trig_2.fla, 263
trig_3.fla, 264
trig_4.fla, 265
url.fla, 52
valstring.fla, 244
variable.fla, 90
vertimer.fla, 212
visible.fla, 70
volppan.fla, 206
while.fla, 140
width.fla, 74
with.fla, 232
x.fla, 82
xscale.fla, 78
y.fla, 84
yscale.fla, 80
FlashPla.exe (PC), 54
FlashPlayer (Mac), 54
text, URL encoding, 94
Fill panel, 7
fills, bitmap, 7
find
array length, 166–167
modulo, 120–121
numeric values, 196–197
square root, 193
fixed width text boxes
create, 20–21
square handle indicator, 20
FLA extension, movies, 32, 39
Flash, HTML tag support, 23
Flash 5, smart clips, 259
Flash Debug Player, password use, 269
Flash Player
display list, 222–223
fscommand action arguments, 54

movies, printing, 61
system clock, date/time determination, 182–183
version return, 212
floating-point numbers, 119
fonts
antialias, 21
colors selections, 20–21
device, 20, 34
embedded, 20
scripts, sizing, 27
styles, 20
for loops, 142–143
forms
input text boxes, 22
user input, capturing, 89
variables, 88
four-frame Timeline, buttons, 12
Frame Actions panel, frame labels, 31
Frame panel
comments, adding, 43
tween adjustments, 15
frame-by-frame animations, 14
frames
ActionScript, assignment, 30–31
comments, adding, 42–43
downloads, checking, 62–63
Frame Actions panel, 31
interface element, 2
jumps, 48
keyframes, 14
labels, 31, 48
layer placement, 31
move backward, 38–39
move forward, 38–39
movie clips, selecting, 18
movie startup point, 46
playhead, positioning, 48–49
preloader animations, 62
rewind, 38–39
select, 19
stop action assignment, 44
tweens, start/end, 14
free-form lines, Pencil tool, 6
functions
argument association, 36
custom, 220–221
trigonometric, 192, 262–265
functions reference, 285
Functions statement, 24

G

generate, random numbers, 195
generic objects, create, 145
global variables, 88
gradients, fill, 7
graphics, 6, 11–12, 32, 60
graphs, Bandwidth Profiler display, 40–41
greater than (>) operator, 126
greater than or equal to (>=) operator, 127
group, selected objects, 6
guide layer, create, 4–5
guided layer, create, 4–5

H

Hand tool, 7
handlers
button, 252
draggable objects, 58–59
enterFrame, 262, 264, 266
event, selecting, 254
mouse, testing, 255
on (press), 28, 58–59
on (rollout), 58
on (dragOut), 28
on (dragOver), 28
on (keyPress, "key"), 28
on (release), 28
on (releaseOutside), 28
on (rollOver), 28
On Press, 258
on rollover, 53
onClipEvent (data), 29
onClipEvent (enterFrame), 29
onClipEvent (keyDown), 29
onClipEvent (keyUp), 29
onClipEvent (load), 29
onClipEvent (mouseDown), 29
onClipEvent (mouseMove), 29
onClipEvent (mouseUp), 29
onClipEvent (unload), 29
onClipEvent, 29
handles, objects, 73, 79
heights, movie clips, 75–76
hexadecimal color codes, 280–282

hexadecimal numbers, color selection, 256
hexadecimal values, colors, 176–177
hide
    layers, 4–5
    mouse, 180
    status bar,
    toolbox, 7
hierarchy, movies, 222–223
HIGH quality, movie clips, 86–87
HTML documents, templates, 34
HTML tags, 33
HTML Template Info dialog box, 34
hues, instances, 10–11
hyphen (-) character
    numeric values, subtracting, 114–115
    subtraction operator, 88

I

identifiers
    colors, 27
    movie clip attachment, 148
images
    antialiasing, 50–51
    background, 9
    bitmap, printing movie as, 60
    frame-by-frame animations, 14
    static, 8
    tweened frame animations, 14
import
    files, sounds, 16–17
    images, buttons, 13
increment (++) operator
    mathematical data type, 88
    unary type, 110
    value setting, 122–123
indents, static text boxes, 21
index, array element number, 162–163
index position
    character string extraction, 96
    substring extraction, 98
inequality (!=) operator, 128–129
infinity, mathematical error evaluation, 216–217
Ink Bottle tool, stroke line color/style, 6
input text boxes
    asterisk character password display, 22
    backgrounds, 22
    borders, 22
    create, 22
    forms, 22
    HTML tag format, 22
    multiline, 22
    password, 22
    single line, 22
    text selections, 208–209
    user input, 22
    variable associations, 22
    Word Wrap box, 22
insert, layers, 5
Insert Target Path dialog box, 230
Instance panel, movie clips, naming/renaming, 19, 66–67
instances
    behaviors, 11
    movie clips
        naming, 19
        targeting, 67
    names, 66–67
    symbols, 9–11
        registration point, 73
instantiated objects, 144
integer numbers, 119
integers, numbers, rounding, 194
interactive graphics, ActionScript support, 8–9
interface, 2–3, 239–243
Internet Explorer, object tag, 33
Item Preview window, symbol preview, 9

J

JavaScript, messages, passing with fscommand action, 55
JPEG graphics, quality adjustments, 32
jumps, 48

K

key code vales, 276–277
keyboards, Key object, 198–201
keyframes, 13–14, 30
keys, movie startup point, 46
keywords
    case sensitive, 36
    colors, 27
    list, 279
    this, 56, 225

L

labels, frames, 31, 48
Lasso tool, object selection, 6–7
Layer Properties dialog box, 5
layers, 2, 4–5, 19, 30–31
Leap Year
  date validation, 246–247
  modulo (%) operator determination, 120
  occurrences, 247
Learning Interactions, Knowledge Tracks, 243
lengths, array, 166–167
less than (<) operator, 126
less than or equal to (<=) operator, 127
lettercase, input value, 244
levels
  defined, 226
  depth, 154–155
  movie clips, quality, 86–87
  movies, specification, 228
Library
  bitmap fill, 7
  instances, create, 10–11
  movie clips, sharing, 8
  sounds, imported, 16
  store panels, 259
  symbols, automatic storage, 8–9
  versus common library, 237
Line tool, 6
line weight, stroke, 7
lines, 6–7
linkages, movie clips, attachment, 148–149
links, Web pages, opening, 53
List Objects, test environment, 272
List Variables, test environment, 273
literal values, quote (") character, 36–37
load
  movie clips, 156–157, 226–229
  movies, 226–229
  variables, 94–95
local variables, 88
lock, layers, 4–5
Lock/Unlock All Layers column, 5
Lock/Unlock icon, 5
logarithms, mathematical functions, 192
logical operators, 132–133
loops
  compound assignment operators, 130–131
  conditional, 140–141
  counters, 122
  for action, 142–143
  Go To action, 49
  movie playback, 34
  sounds, 16–17
LOW quality, movie clips, 86–87

M

Macintosh
  Controller Toolbar access, 39
  FlashPlayer file, 54
margins, static text boxes, 21
mask layer, 4–5
masked layer, create, 4–5
mathematical calculations
  convert a string to a number, 214–215
  error evaluation, 216–217
  numeric values, 196–197
  raise a power, 193
  round numbers, 194
  square root, 193
mathematical equations, expressions, 37
mathematical functions, 192
mathematical operators, data types, 88
MEDIUM quality, movie clips, 86–87
menus, 53
messages, 55, 62
methods
  acos, 192
  asin, 192
  atan, 192
  attachMovie, 148–151, 234
  attachSound, 202
  concat, 168
  cos, 192, 262
  dot syntax, 224–225
  duplicateMovie, 234
  getAscii, 198, 200
  getBeginIndex, 208
  getbounds, 152–153
  getBytesLoaded, 156–157
  getBytesTotal, 156–157
  getCode, 198
  getDay, 184

getEndIndex, 208
getFullYear, 182–183
getHours, 184
getMilliseconds, 184
getMinutes, 184
getMonth, 182–183
getPan, 206
getRGB, 176–177
getSeconds, 184
getTimezoneOffset, 185
getUTCDate, 182–183
getUTCDay, 184
getUTCFullYear, 182–183
getUTCHours, 184
getUTCMilliseconds, 184
getUTCminutes, 184
getUTCMonth, 182–183
getUTCSeconds, 184
getYear, 183
globalToLocal, 160–161
hitTest, 158–159
join, 174
Key, 260
Key.isDown, 200
Key.isToggled, 200
localToGlobal, 160–161
log, 192
Math.abs, 196
Math.cell, 194
Math.floor, 194
Math.max, 196–197
Math.min*mf, 196–197
Math.pow, 193
Math.random, 195
Math.round, 194
Math.sqrt, 193
Mouse.hide, 180
Mouse.show, 180
MovieClip object, 146–147
new Color, 256
play, 46
pop, 170
push, 168
reverse, 172
Selection.getBeginIndex, 208
Selection.getCaretIndex, 208
Selection.getEndIndex, 208
Selection.getFocus, 208, 246
Selection.setSelcection, 246, 250
setDate, 188–189
setFullYear, 188
setHours, 190
setMilliseconds, 190
setMinutes, 190
setMonth, 188
setRGB, 176
setSeconds, 190
setTransform, 178
setUTCDate, 188
setUTCFullYear,, 188
setUTCHours, 190
setUTCMilliseconds, 190
setUTCMinutes, 190
setUTCMonth, 188
setUTCSeconds, 190
setVolume, 206
setYear, 189
shift, 170–171
sin, 192, 262
slash syntax, 224–225
slice, 99, 172–173
splice, 170–171
start, 204
string.charAt, 96
string.concat, 100
string.fromCharCode, 97
string.indexOf, 104, 248, 250
string.lastIndexOf, 105, 251
String length, 244
string.substring, 97
string.toLowerCase, 106, 249–250
string.toUpperCase, 106, 250
substring, 244
swapDepths, 154–155
tangent, 192
toString, 174–175
unshift, 168
minus (-) operator, numeric values, subtracting, 114–115
mixer, colors, creating, 7
modems, speed selection, 40
modes
Actions panel, 24–27
symbol-editing, 9, 19

modifiers
interface element, 2
tools, 6–7
modulo (%) operator
Leap Year, 246–247
mathematical data type, 88
remainder, finding, 120–121
months, current, retrieving, 183
motion guide layers, motion tween, 4
motion tween, create, 4, 14
mouse
coordinates, retrieve, 161
hide/display, 180
objects
change color, 257
roll over actions, 254–255
pointers, 180–181
x and y coordinates, 180
mouse handlers, testing, 255
move
movie clips
across the Stage, 82–83
up/down, 84–85
objects
diagonally, 264
trigonometric functions, 262
script line, 27
selected objects, 6
Stage, 7
statements, 24–25
movie clips
ActionScript, 29
animation conversion, 19
antialiasing, 86–87
associations, ActionScript, 6
attach, 148–151
attachment identifiers, 148
boundary, 152–153
buttons, 12
change height/width, 266
child/parent relationship, 234
colors, changing, 176–177
convert
Boolean, 108–109
buttons, 252–253
coordinates, retrieving, 160–161
create, 18–19
data type, 88
Debugger, 268
depth level, 154–155
duplicate, 234–235
exchange, 240
frames
pasting, 19
selecting, 18
handlers, 252, 283
heights
adjusting, 76–77
scaling, 80–81
instance behavior type, 11
instances
naming, 19
naming/renaming, 66–67
targeting, 67
intermediary, 240
load, 226–229
load check, 156–157
mouse, custom cursor, 180–181
move, across the Stage, 82–83
names, attached, 150–151
playback, symbol advantages, 8
preloader, 156
properties, 64–65
properties, retrieving, 210–211
quality levels, 86–87
reference data types, 92
reusable animations, 8
rotate, 72–73
scripts, create designs, 263, 265, 267
shared library, 8
smart clip, 18
statements, sending, 230–231
stop action, 44
stop script, 45
targeting, 46
transparency, adjustments, 68–69
unload, 226–229
variables, assignment, 92–93
visible/invisible, 70–71
widths
adjusting, 74–75
scaling, 78–79
Movie Explorer, display list, 222–223
movie projectors, Flash Player, 54–55

MovieClip object, methods, 146–147
movies
antialiasing, 50–51
boundary, 33
comments, adding, 42–43
compression, 34
crop, 33
Debugger
activation, 32
troubleshooting, 269
device fonts, 34
display list, 222–223
display quality, 41
display variables, 273
elapsed time, 212
FLA extension, 32, 39
Flash Player, 54–55
fonts, embedded, 20
height adjustments, 65
hierarchy, 222–223
HTML document templates, 34
ignore trace actions, 32
JPEG graphics quality adjustments, 32
load, 226–229
load order, 32
location, adjustments, 65
loop playback, 34
loops, continuous playback, 38
modems, speed selection, 40
names, 32
one-frame, 252
Output Window, syntax errors, 270–271
pause, 34
play action, 46–47
playback, 34
playhead, positioning, 48–49
preloader animations, 62
preloaders, 156
print, 60–61
publish, 32–35
rewind, 39
scenes, testing current, 38–39
scripts, debug, 274–275
shortcut menu, 34
size report, 32
smart clips, custom interface, 240–243
sounds, 33–34, 38–39
stop action, 44
stop playback, 39
streaming, 41
supported file formats, 32
SWF extension, 32, 39
targeting, 46
test environment, 39–41
testing, 38–39, 245, 271
transparency, adjustments, 68–69
unload, 226–229
width adjustment, 33, 65
zoom in/out, 41
multiline comments, 42–43
multiline dynamic text box, 23
multiline input text box, 22
multiplication (*) operator, 88, 116–117
multiply, numeric values, 116–117

N

names
instance, 66–67
layers, 4–5
movie, 32
movie clips
attached, 150–151
instance, 19
scenes, 3
symbols, 9
tweens, 14
variables, conventions, 88–89
Netscape Navigator, embed tag, 33
non-numeric (NaN) values, 196
normal layer, create, 4–5
Normal mode, 24–27, 48–49
NOT (!) operator, 132–133
Num Lock key, Status bar display, 41
NUM, appended actions, 280
numbers
array element, 162–163
base ten, 215
base two, 215
conversions, 108–109
data type, 88
floating-point, 119
integer, 119
random, generating, 195

round, 194
strings, convert, 214–215
numeric values
add, 112–113
divide, 118–119
find, 196–197
multiply, 116–117
NaN (non-numeric), 196
subtract, 114–115

O

Object Actions panel, handlers, 28–29
objects
activate layer, 4
collision detection, 158–159
Color, 176–177, 256–257
convert, 108–109
create, 56–59
data type, 88
Date, 182–183, 186–187
described, 144
draggable
creating, 56–59
handlers, 58–59
scroll bars, 57
sound volume/pan, 57
generic, 145
group, 6
handles
rotation, 73
scaling, 79
instantiated, 144
Key, 198–201
layers, separate, 5
Math, 192, 262
Mouse, 180–161, 257
move
back and forth, 263–264
clockwise, 262
counterclockwise, 262
diagonally, 264
MovieClip, 46
predefined, 144–147
properties, 145
reference data types, 92
registration point, 73, 254
rotation, speed, 254
select, 6–7, 9
Selection, 208–209
set rotation value, 254
Sound, 202–205
symbol conversion, 9
tweens
naming, 14
scaling, 79
variables, assignment, 92–93
objects reference, 288–293
Objects statement, 24
obtain, string length, 102–103
open, Web page, 52–53
operands, 110
operators
addition (+), 88, 100
AND (&&), 132–133
array access [ ], 162–163
assignment (=), 88
binary, 110
compound assignment, 130–131
conditional (?), 110
decrement (—), 88, 124–125
division (/), 88, 118–119
equal sign (=) character, 88
equality (==), 128–129
greater than (>), 126
greater than or equal to (>=), 127
if statement, 135
increment (++), 88, 122–123
inequality (!=), 128–129
less than (<), 126
less than or equal to (<=), 127
logical, 132–133
mathematical, 88
minus (-), subtracting numeric values, 114–115
modulo (%), 88, 246–247
multiplication (*), 88, 116–117
new, 144
NOT (!), 132–133
OR (||), 132–133
plus (+), adding numeric values, 112–113
post-decrement (—), 124–125
post-increment (++), 122–123
precedence order, 110–111, 117
pre-decrement (—), 124–125
pre-increment (++), 122–123

subtraction (-), 88
ternary, 110
unary, 110
operators reference, 284–285
Operators statement, 24
Options menu, Output Window, 270–271
OR (||) operator, 132–133
order
layers, changing, 4–5
scenes, changing, 3
outline view, layers, 5
outlines, graphic stroke lines, 6–7
Outlines icon, 5
Output Window, 270–273
Oval tool, 6
Over state, buttons, 12–13

P

padlock icon, locked layer indicator, 4–5
panels
Actions, 24–27
change size, 261
create, 258–261
described, 258
emulate, 258–261
Frame Actions, 31
hiding, 260
keys versus button use, 260
Parameters, 24–25
Scene, opening, 3
Text Options, 20–23
pans, sound volume, 206–207
parameter values, add/remove from smart clips, 236–239
Parameters panel, statements, arguments, 24–25
parameters, 51, 178, 228
parent movie clips, child placement, 234
parentheses (and) characters, arguments, 36
passwords, 22
paste, frames into movie clip, 19
path names, importance of accuracy, 259
pause, movies, 34
Pen tool, 6
pencil icon, active layer indicator, 4
Pencil tool, 6
percent sign (%) character, modulo operator, 88, 120–121
period (.) character, floating point number decimal point, 119
play, movie, 46–47
playback
movies, 34
sounds, 16, 204–205
playhead, 2, 48–49
plus (+) operator, numeric values, adding, 112–113
plus sign (+) character
addition operator, 88, 100
numeric values, adding, 112–113
PNG graphics, movie support, 32
pointers, mouse, 180–181
post-decrement (—) operator, 124–125
post-increment (++) operator, 122–123
power, raise, 193
precedence, operators, 110–111, 117
predefined identifiers, colors, 27
pre-increment (++) operator, 122–123
preloader animations, 62
preloader movies, 156
primitive data types, 92
print, 27, 60–61
projectors, Flash Player, 54–55
properties
color transform, 179
colors, 27
continuous update statement, 65
Debugger, changing, 268–269
equal sign (=) value assignment, 65
Key object, 201
movie clip, 64–65
retrieving, 210–211
special effects, 254–255
objects, 145
retrieve, 210–211
symbols, 8–11
variables, 37
properties reference, 287
Properties statement, 24
Properties tab, Debugger, movie clip, 268
Publish Settings dialog box, 16, 32–35, 50
publish, movies, 32–35

Q

quality, movie clips, levels, 86–87
queries, system clock, 183
question mark (?) character, conditional operator, 110
QuickTime animations, movie support, 32

quote (“) characters, literal values, 36–37
quoted strings, colors, 27

R

random numbers, generate, 195
read-only values, 268
Rectangle tool, 6
reference data types, 92
registration point
  rotation angle change, 254
  symbols, 73
relative target path, 224–225
remainder, modulo operator (%), 120–121
remove, scenes, 3
rename, scenes, 3
reports, movie size, 32
retrieve
  coordinates, 160–161
  mouse coordinates, 161
  movies, elapsed time, 212
  properties, 210–211
reverse, arrays, 172–173
rollover menu, create with getURL action, 53
rotation
  effects, create, 254–255
  movie clips, 72–73
  registration point, 73
round handle, 20
round, numbers, 194
rounded corners, Rectangle tool, 6
rules, variables, naming conventions, 89

S

Scale, movie clips, 78–81
Scene panel, 3
scenes, 3, 38–39, 48
scope, variables, 88
scripts
  age, calculation base on date entered, 187
  array access [ ] operator, 163
  arrays, removing elements, 171
  ASCII code retrieval, 97
  buttons, rotate continuously, 255
  CGI, 228–229
  colored syntax use, 274
  concatenate strings, 101
  continuous action, 253
  continuous scroll button, 219
  debug, 274–275
  develop in parts, 274
  document with code, 274
  draggable objects, 59
  error message, 217
  fonts, sizing, 27
  mouse
    custom pointer, 181
    retrieving coordinates, 161
  movie clips
    create design, 263, 265
    depth swap, 155
    draggable duplicate, 235
    duplicating, 235
    height adjustment, 77
    instance scaling, 81
    move up/down with key press, 85
    moving across the Stage, 83
    placing attached movie clip on Stage, 153
    placing multiple instances on Stage, 151
    property adjustments, 232
    replacing, 227
    restarting, 47
    stopping, 45
    toggle size, 211
    width adjustment, 75
  movies, length checking, 103
  music, toggle on/off, 203
  naming conventions, 274
  numbers, raise a power, 193
  numeric values
    comparing two values, 197
    discount subtraction, 115
    subtotal calculation, 113
  objects
    change color, 257
    collision detection, 159
  post-decrement (—) operator, 125
  post-increment operator, 123
  preloader, 63
  printing, 27
  smiley face, moving, 199
  sounds
    increasing volume, 205
    volume panning, 207
  string search, 27

strings
last substring occurrence search, 105
lowercase to uppercase conversion, 106
syntax check, 274
syntax errors, checking, 27
text, selections, 209
use meaningful names for parts, 274
visible/invisible movie clips, 71
scroll bars, draggable objects, 57
scrollable text box, create, 218–219
search
strings, 27, 104–105
text box, 248–251
select
enabled button, 13
frames, 19
instances, 11
keyframes, 14
layers, 19
Stage objects, 6–7, 9
statements, 24
semi-colon (;) character, statements, 36–37
set
date, 186–187
date values, 188–189
movie clip quality, 86–87
movie quality, 50–51
volume and panning, 206–207
shape, tweened animation type, 14
Shift key
draw, graphics, 6
frame block selection, 19
shortcut menu, movies, 34
Show Deprecated Syntax, 27
Show Layers as Outlines column, 5
Show Library icon, 9, 13
Show/Hide All Layers column, 5
Show/Hide icon, 5
single line dynamic text box, 23
single line input text box, 22
single line text boxes, 20–21
single-line comments, 42–43
size report, movies, 32
slash (/) asterisk (*) characters, comments, 42
slash (/) character
division operator, 88
numeric values, dividing, 118–119
slash syntax, target path, 224–225
smart clips
create, 236–239
custom interface, 239–243
described, 18
Learning Interactions Library, 243
panels, 259
parameters, defined, 243
Smart Clips icon, 237
smile face, moving, 199
smooth curves, Pen tool, 6
smooth lines, Pencil tool, 6
Sound Panel, effects, 16
Sound Properties dialog box, 16
Sound Setting dialog box, 33–34
sounds
add, 16–17
compression, 16, 34
create, 202–203
draggable objects, 57
event, 16
Library, imported sound storage, 16
loops, 16–17
movies, settings, 33–34
mute, 38–39
panning, 206–207
playback, 16, 204–205
Sound Panel, effects, 16
stream, 16
synchronization types, 16
volume, setting, 206–207
spaces, static text boxes, 21
special effects, trigonometric functions to create, 266–267
splitter bar, Toolbox list, sizing, 26
spotlight effects, mask layers, 4
square handle, 20
square root, 193
squares, Rectangle tool, 6
Stage
coloring book, color selection, 256
Debugger, show changes, 268
Hand tool, moving the Stage, 7
instance of a symbol, 11
interface element, 2
layers, making active, 4

movie clips
boundary, 152–153
moving across, 82–83
placing, 150–151
movies
playing, 46
stopping, 44
objects
moving, 262–263
rotation value, 254
selecting, 6–7
symbol conversion, 9
Start synchronization, sounds, 16
statements
ActionScript, 24–27
arguments, 24
color selections, 27
conditions, testing, 134–135
continuous property update, 65
delete, 24–25
if, 250
instanceName.property, 65
modulo (%) operator execution, 120
movie clips, sending, 230–231
nested, 138–139
semi-colon (;) character, 36–37
states, buttons, 12–13
static images, ActionScript non-support, 8
static text boxes, create, 20–21
Status bar, hide/display, 41
Stop synchronization, sounds, 16
stop, movie, 44–45
straight lines, 6
stream sounds, 16
Stream synchronization, sounds, 16
streaming bar, test environment, 41–42
string functions, validate data, 244–245
strings
arrays, converting, 174–175
case, changing, 106–107
characters, extracting, 96–97
concatenate, 100–101
conversions, 108–109, 174–175, 214–215
convert to number, 214–215
data type, 88
escape sequence characters, 91
length, obtaining, 102–103
modulo (%) operator, 120
quoted, color, 27
search, 104–105
search specifications, 244
substring extraction, 97–99
validate, 244–245
stroke, adjustments, 7
stroke lines, graphic outlines, 6–7
Subselect tool, line adjustments, 6
substring
check value, 244
extracting, 97–98
subtract, numeric values, 114–115
subtraction (-) operator, 88, 110
swap, depths, 154–155
SWF extension, movies, 32, 39
SWF file, custom interface movie, 242
Symbol Linkage Properties dialog box, 148–149, 202
Symbol Properties dialog box, 8–9, 12–13, 240–241, 252
symbol-editing mode, movie clips, pasting frames, 19
symbols, 8–11, 73, 252–253
syntax
ActionScript, 36–37
dot, 224–225
methods, 246
slash, 224–225
variables, 88–89
syntax check, scripts, 274
syntax errors, 27, , 270–271
system clock, date/time determination, 182–183

T

target path, 224–225
templates, HTML documents, 34
ternary operators, operands, 110
test environment
access methods, 39
Bandwidth Profiler, 40–41
color changes, 257
List Objects, 272
List Variables, 273
modems, speed selection, 40
mouse handlers, 255
movies, 272
script to validate user entries, 245
testing, 40–41

scripts, 274–275
Status bar, hide/display, 41
streaming bar, 41–42
validate, dates, 246–247
text
assigning to variable, 248
dynamic, variables, 88
layout, 21
make selectable, 21
selections, 208–209
text boxes
associate variable, 248
dates, displaying, 191
dynamic, 23
input, 22
scrollable, 218–219
search, 248–251
search from bottom, 251
static, 20–21
text selections, 208–209
text editors, ActionScript, writing, 25
text files, URL encode, 94
text formats, Text Option panel, 20
Text Options panel, 20, 22–23
Text tool, 88–89
time zones, offsets, 185
Timeline
buttons, 8–9
comments, adding, 42–43
four-frame buttons, 12
frames, preloader animations, 62
functions, calling, 220
interface element, 2
movie clips, selecting, 18
play action, 46
stop action, 44
target path, 224–225
target requirements, 231
tweens, 15
times
movies, elapsed time, 212
time zones, offsets, 185
values, setting, 190–191
tint, instances, 10–11
Toolbox, 2, 7
Toolbox list, 24, 26–27
tools, 6–7
trace actions, ignore, 32
trace action, evaluate expression, 270
transform, color values, 178–179
transparency, movies/movie clips, 68–69
trigger actions, 252
trigonometric functions, 192, 262–265
trigonometric special effects, create, 266–267
troubleshoot
Debugger, 268–269
Output Window, 270–271
tweened animations, 14
tweened frames, 14
tweens
fading effect, 69
motion, 4
objects, scaling, 79
start/end frames, 14

U

unary operators, operands, 110
unhide
layers, 4–5
toolbox, 7
Universal Coordinated Time, 182–191
unload, movie clips, 226–229
unlock, layers, 4–5
Up state, buttons, 12–13
URL encode, text files, 94
URL encoding characters, 278–279
use, Actions panel, 24–27
user input, 22, 89
users, 56–59, 244

V

validate
dates, 246–247
strings, 244–245
values
arrays, comma (,) character separator, 92
arrays, elements, 164–165
ASCII, 198–200
Boolean variables, 90
Boolean, false, 56
colors, hexadecimal, 176–177
colors, transform, 178–179

date, 184–185, 188–189
dates, updating, 189
Debugger, changing, 268
decrement (—) operator, 124–125
equal sign (=) character, 65
expression, 36
false, 90
increment (++) operator, 122–123
indexOf, 248
literal, 36
non-numeric (NaN), 196
numeric
add, 112–113
divide, 118–119
multiply, 116–117
subtract, 114–115
times, setting, 190–191
true, 90
variable assignment, 88, 90–91
Variables tab, Debugger, 268–269
variables
$QU, 50
add/remove from Watch list, 269
arguments, sending application to specified URL, 52
assigning text, 248
associate text box, 248
associate with dynamic text box, 245
color, 256
dynamic text box associations, 23
dynamic text display, 88–89
equal sign (+=) operator, 88
expressions, 37
forms, 88
global scope, 88
input text box associations, 22
inputpowV, 193
inputV, 193
load, 94–95
local scope, 88
movie clips, assignment, 92–93
names, conventions, 88–89
objects, assignment, 92–93
onOff, 203
result, 193
scope, 88
startSearch, 104
StartV, 248
syntax, 88–89
user input forms, 89
values, assigning, 88–91
vector graphics, movies, printing, 60
versions, Flash Player, return, 212
vertical splitter bar, Toolbox list size, 26
view, 3, 5, 222–223
virtual key codes, 198
visibility, movie clips, 70–71

W

Web browsers, movies, testing, 39
Web pages, 52–53
Web sites, Macromedia, 269
widths, movie clips, 74–75
Windows PC, 39, 54
Word Wrap box, 22–23

X

x-coordinates, 160–161, 180

Y

y-coordinates, 160–161, 180
year, current, retrieving, 183

Z

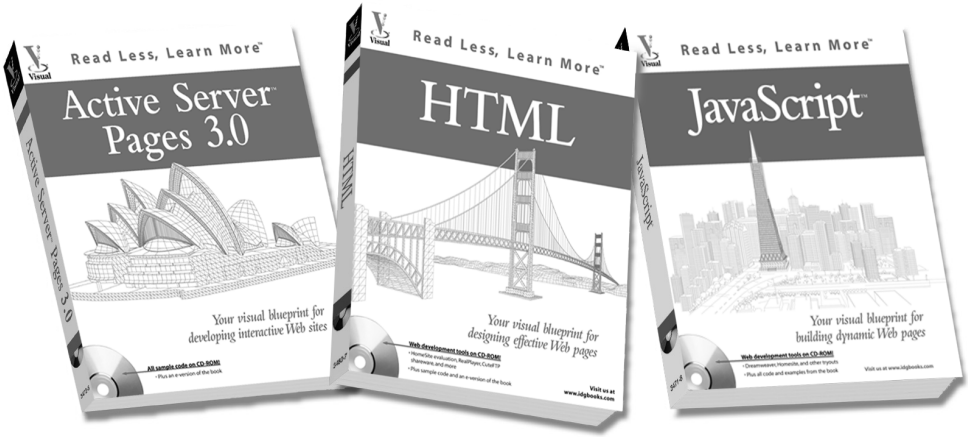
zero x-coordinate, 160
zero y-coordinate, 160
Zoom tool, 7
zoom, Stage, 7

Read Less – Learn More™

New Series!

The visual alternative to learning complex computer topics

For experienced computer users, developers, network professionals who learn best visually.



Extra

Apply It

"Apply It" and "Extra" provide ready-to-run code and useful tips.

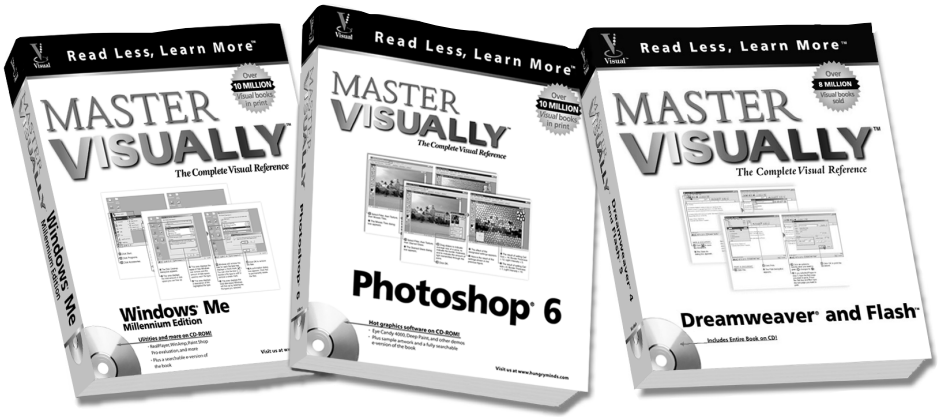
Title	ISBN	Price
Active Server™ Pages 3.0: Your visual blueprint for developing interactive Web sites	0-7645-3472-6	$26.99
HTML: Your visual blueprint for designing effective Web pages	0-7645-3471-8	$26.99
Java™: Your visual blueprint for building portable Java programs	0-7645-3543-9	$26.99
JavaScript™: Your visual blueprint for building dynamic Web pages	0-7645-4730-5	$26.99
JavaServer™ Pages: Your visual blueprint for designing dynamic content with JSP	0-7645-3542-0	$26.99
Linux®: Your visual blueprint to the Linux platform	0-7645-3481-5	$26.99
Perl: Your visual blueprint for building Perl scripts	0-7645-3478-5	$26.99
PHP: Your visual blueprint for creating open source, server-side content	0-7645-3561-7	$26.99
Unix®: Your visual blueprint to the universe of Unix	0-7645-3480-7	$26.99
XML: Your visual blueprint for building expert Web pages	0-7645-3477-7	$26.99

Over 10 million *Visual* books in print!

with these two-color Visual™ guides

The Complete Visual Reference

MASTER VISUALLY™



For visual learners who want an all-in-one reference/tutorial that delivers more in-depth information about a technology topic.

MASTER IT

"Master It" tips provide additional topic coverage.

Title	ISBN	Price
Master Active Directory™ VISUALLY™	0-7645-3425-4	$39.99
Master Microsoft® Access 2000 VISUALLY™	0-7645-6048-4	$39.99
Master Microsoft® Office 2000 VISUALLY™	0-7645-6050-6	$39.99
Master Microsoft® Word 2000 VISUALLY™	0-7645-6046-8	$39.99
Master Office 97 VISUALLY™	0-7645-6036-0	$39.99
Master Photoshop® 5.5 VISUALLY™	0-7645-6045-X	$39.99
Master Red Hat® Linux® VISUALLY™	0-7645-3436-X	$39.99
Master VISUALLY™ Dreamweaver® 4 and Flash™ 5	0-7645-0855-5	$39.99
Master VISUALLY™ FrontPage® 2002	0-7645-3580-3	$39.99
Master VISUALLY™ HTML 4 & XHTML™ 1	0-7645-3454-8	$39.99
Master VISUALLY™ Microsoft® Windows® Me Millennium Edition	0-7645-3496-3	$39.99
Master VISUALLY™ Office XP	0-7645-3599-4	$39.99
Master VISUALLY™ Photoshop® 6	0-7645-3541-2	$39.99
Master VISUALLY™ Windows® 2000 Server	0-7645-3426-2	$39.99
Master Windows® 95 VISUALLY™	0-7645-6024-7	$39.99
Master Windows® 98 VISUALLY™	0-7645-6034-4	$39.99
Master Windows® 2000 Professional VISUALLY™	0-7645-3421-1	$39.99

The Visual™ series is available wherever books are sold, or call 1-800-762-2974.

Outside the US, call 317-572-3993

ORDER FORM

TRADE & INDIVIDUAL ORDERS

Phone: **(800) 762-2974**
or **(317) 572-3993**
(8 a.m.–6 p.m., CST, weekdays)
FAX : **(800) 550-2747**
or **(317) 572-4002**

EDUCATIONAL ORDERS & DISCOUNTS

Phone: **(800) 434-2086**
(8:30 a.m.–5:00 p.m., CST, weekdays)
FAX : **(317) 572-4005**

CORPORATE ORDERS FOR VISUAL™ SERIES

Phone: **(800) 469-6616**
(8 a.m.–5 p.m., EST, weekdays)
FAX : **(905) 890-9434**

Qty	ISBN	Title	Price	Total

Shipping & Handling Charges

	Description	First book	Each add'l. book	Total
Domestic	Normal	$4.50	$1.50	$
	Two Day Air	$8.50	$2.50	$
	Overnight	$18.00	$3.00	$
International	Surface	$8.00	$8.00	$
	Airmail	$16.00	$16.00	$
	DHL Air	$17.00	$17.00	$

Subtotal _____

CA residents add applicable sales tax _____

IN, MA and MD residents add 5% sales tax _____

IL residents add 6.25% sales tax _____

RI residents add 7% sales tax _____

TX residents add 8.25% sales tax _____

Shipping _____

Total _____

Ship to:

Name _____

Address _____

Company _____

City/State/Zip _____

Daytime Phone _____

Payment: ☐ Check to Hungry Minds (US Funds Only)
☐ Visa ☐ MasterCard ☐ American Express

Card # _____ Exp. _____ Signature _____

Hungry Minds™

maranGraphics®