# Fundamentals of ActionScript 3.0

DEVELOP AND DESIGN

Doug Winnie

# Fundamentals of ActionScript 3.0

## DEVELOP AND DESIGN

Doug Winnie

Peachpit Press

**Fundamentals of ActionScript 3.0: Develop and Design**
Doug Winnie

**Peachpit Press**
1249 Eighth Street
Berkeley, CA 94710
510/524-2178
510/524-2221 (fax)

Find us on the Web at: www.peachpit.com
To report errors, please send a note to errata@peachpit.com
Peachpit Press is a division of Pearson Education
Copyright © 2012 by R. Douglas Winnie

**Editor:** Nancy Peterson
**Production editor:** Myrna Vladic
**Development editor:** Robyn G. Thomas
**Copyeditor:** Liz Merfeld
**Technical Editor:** Christopher Coudron
**Cover design:** Aren Straiger
**Cover production:** Mimi Heft
**Interior design:** Mimi Heft
**Compositor:** Danielle Foster
**Indexer:** Jack Lewis

ISBN 13: 978-0-321-77702-7
ISBN 10: 0-321-77702-6

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

*This book is dedicated to Hoover. Hoover was a big part of my life,*
*and was always by my side while doing "tech-no" things. I miss you Hoover!*

*This book is also dedicated to my husband, Mike.*
*While not always into my "tech-no" things, he is my inspiration for doing*
*great things—"tech-no" or not. Thanks, Groovy Dude!*

# ACKNOWLEDGEMENTS

With too many people to mention individually, I'm going to do a group acknowledgement of all of the members of the Adobe Flash Professional, Flash Player, AIR, and Flex teams that have inspired me to create great things, and to a platform that overcomes amazing obstacles to give creative and web professionals the ability to express themselves wherever they go.

I also want to acknowledge Chris Coudron, my tech reviewer and friend, for his effort to make me look good in code and for reminding me that there is not a NOR operator in ActionScript. D'oh!

# CONTENTS

# INTRODUCTION

Welcome to ActionScript. Over the next several chapters, you'll be introduced to one of the most versatile programming languages to create web applications for the browser, desktop applications, and mobile apps for multiple platforms. For years the Flash Platform has provided people with the most powerful set of technologies to creatively express themselves across multiple screens and platforms with its combination of the Flash Player and AIR runtimes, tools like Flash Professional CS5.5 and Flash Builder 4.5, and languages and frameworks like ActionScript 3.0 and Flex 4.5.

Over the last several years, I have taught people how to make their projects interactive and how to captivate and engage users. During that time at San Francisco State University, my series on Adobe TV, and conference appearances, I have appreciated the difficulty of learning scripting and coding. Learning programming is a steep task, and there are many ways to teach it. What I have found is that combining programming basics, simple examples, problem solving, and real-world projects has been very effective, and it is what you have in your hands (or on your screen) now.

## WHO THIS BOOK IS FOR

This book is crafted for people who are familiar with Flash Professional, the animation and interactive design tool from Adobe Systems that is part of Creative Suite. The lessons and projects here assume that you have a basic understanding of the Flash Professional product. This book is designed for people who are new to coding or are struggling with the migration from ActionScript 2.0 to 3.0. Here are some examples of what you should know and be able to do before attempting to start with this book:

- Import graphical assets from Creative Suite design tools
- Create timeline animations using tweens using keyframes
- Create symbols using the Library panel
- Organize and rename timelines in the Timeline panel and symbols in the Library panel
- Publish and build animations for the web browser

With these basic skills, you can create very interesting web animations; however, without ActionScript, the animations lacked any interaction with the user, and there is no way to bring them to other platforms including mobile devices. That is exactly what this book will teach you—how to make these projects interactive and take them further.

The latest edition, Flash Professional CS5.5, has added a significant number of new features to support mobile app creation that are covered at the end of the book.

## WHO THIS BOOK IS NOT FOR

If you are already an intermediate or advanced coder, this book may be too basic for your needs. There are a significant number of books that focus on advanced ActionScript 3.0 concepts, including the adoption of best practices and code design patterns that will make you a better and more proficient coder.

In addition, if you have never worked with Flash Professional, I recommend you learn how to use the basic product before tackling the contents here. There are excellent books available to help you learn how to get started with Flash Professional to create animations and how to master design workflows when working with Creative Suite design applications like Photoshop, Illustrator, and Fireworks.

## HOW YOU WILL LEARN

This book has a specific methodology for how the concepts are introduced. First are the fundamentals of how to interact and work with objects that are on the Stage. The examples that are in the book are simple—and this is intentional, to help you understand how ActionScript works without getting into the weeds of your project's design or assets. You can adapt and expand these simple examples for your own projects.

After you gather a sizable amount of new ActionScript know-how, it is time to put it to work. There are three major projects in the book that pose real-world situations for you to solve using the skills you have learned. The projects present you with a programming challenge and ask you to solve it. You can compare your finished projects with the examples in the book to discover how your approach matches or differs.

## WHAT YOU WILL LEARN

This book is divided into five major parts.

### PART 1: GETTING THE FUNDAMENTALS

You'll learn general ActionScript concepts that you can use to make ActionScript interact with objects on the Stage and in the Library of your project. You'll build on this, understanding how to flow your code through reusable modules called functions, and then how to respond to user interaction with event handlers.

### PART 2: EXPLORING THE BASICS OF CLASSES

You'll jump into the basics of what is called object-oriented programming (OOP), which is what separates the coders from the scripters. Through OOP you can unlock a lot of flexibility in how you create projects, learning how to make reusable objects and containers that can extend the sophistication of your projects.

### PART 3: RESPONDING TO CONDITIONALS AND WORKING WITH LOGIC

Adapting your project based on certain conditions then is the focus of the next section, where through the use of conditionals, your project can adapt to different interactions from the user or even to random events to begin introducing gaming concepts to your project.

### PART 4: GETTING CREATIVE WITH ACTIONSCRIPT

Although ActionScript is a programming language, it has its creative side. This is covered in the fourth section, where you will learn how to draw, animate, and work with external assets in your projects.

### PART 5: CREATING MULTI-SCREEN PROJECTS

After you have mastered all the previous topics, it is time to take your projects out of the browser and take advantage of the Flash Platform to create desktop applications for Windows and Mac OS X operating systems and mobile apps for the popular Android and iOS platforms.

You'll cover a lot, but at the end, you'll have a solid foundation on how ActionScript works and the power that you have at your fingertips to express yourself across screens and platforms.

**So let's get started!**

# WELCOME TO ACTIONSCRIPT 3.0

ActionScript 3.0 is the programming language of the Adobe Flash Platform, a multi-screen and mutli-device development platform for creating interactive and expressive content. With the latest generation of the Adobe runtimes, Flash Player and AIR, you can take your ideas and creative vision to the browser, desktop, mobile phones, tablets, and Internet-enabled televisions. Let's review some of the tools that you'll be working with.

## THE TOOLS AND RUNTIMES

In the course of this book, there are three main tools and runtimes that you'll be working with:

### FLASH PROFESSIONAL CS5.5

The latest generation of the Flash authoring tool combines powerful animation capabilities, library management, and an integrated coding environment designed for ActionScript 3.0 coding. Part of Creative Suite 5.5, Flash Professional CS5.5 adds new support to work with the latest generation of Adobe AIR and Flash Player 10.2 to create content and applications for the popular Android and iOS mobile platforms.

### FLASH PLAYER 10.2

Flash Player is what brings the web to life. It is the Internet plug-in for your desktop or mobile phone that allows you to play interactive content, video, and games. The latest version includes enhanced support for hardware acceleration, better video playback, and memory and processor performance optimization.

### ADOBE AIR 2.6

The Adobe AIR runtime is what allows interactive designers and developers to take their applications outside the browser and bring them to the Windows and Mac OS X operating systems as desktop applications, or to the Android and iOS platforms as installable mobile applications.

## OTHER HELPFUL TOOLS

Although not part of this book, there are other tools that are helpful for working with the Flash Platform, including:

### ADOBE FLASH BUILDER 4.5

Flash Builder is the professional coding IDE for the Flash Platform. It includes advanced programming functionality to optimize projects, and it makes working with larger projects and coordinating projects with teams easier. Flash Builder also supports working with Flash Professional projects and using the Adobe Flex framework.

### ADOBE FLEX 4.5

The Flex framework is used specifically to create data-driven applications for the browser, desktop, and mobile devices. Incorporating skinnable components, declarative layout, ActionScript logic, and support for a growing set of platforms, it is the fastest way to create a robust application for multiple screens and devices.

### ADOBE FLASH CATALYST CS5.5

Flash Catalyst is designed to work in a team environment when a designer and a developer are building an Internet application using the Flex framework. Interaction designers can create skins for Flex components and craft the overall user interface of a Flex application as a wireframe, prototype, or a finished application. Flash Catalyst CS5.5 introduces round-trip functionality with Flash Builder 4.5 to allow designers and developers to work collaboratively.

PART 1

# GETTING THE FUNDAMENTALS

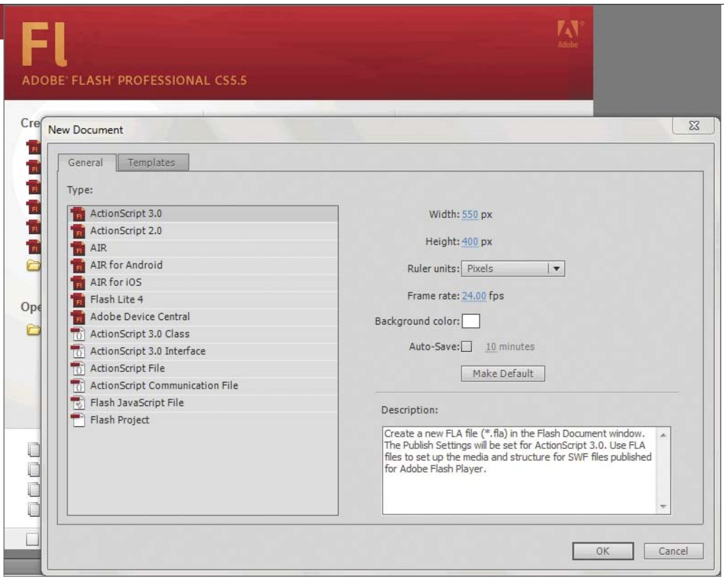
1

# **ACCESSING** AND **MANIPULATING** OBJECTS

ActionScript gives everyone the ability to add interactivity to graphics, video, and other items that are placed on the Stage. In order to add this functionality, ActionScript needs a way to access the various objects that are on the Stage. In this opening section, you'll learn how to access these objects so you can later add basic interactivity.

In this chapter, you'll learn how to change the visual properties of objects that you place on the Stage. To do this, you'll discover more about the importance of converting objects to symbols and giving objects names, called *instance names*. You'll create your first ActionScript code to change how things look on the screen. Along the way, you'll learn some powerful tools to help Flash communicate back to you through the Output panel with messages that can help you see how your code is working.

## **CREATING** A NEW PROJECT FOR **ACTIONSCRIPT 3.0**



Width: 550 px

Height: 400 px

Ruler units: Pixels

Frame rate: 24.00 fps

Background color:

Auto-Save: 10 minutes

Create a new FLA file (*.fla) in the Flash Document window. The Publish Settings will be set for ActionScript 3.0. Use FLA files to set up the media and structure for SWF files published for Adobe Flash Player.

**FIGURE 1.1** New Project dialog box in Flash Professional CS5.5

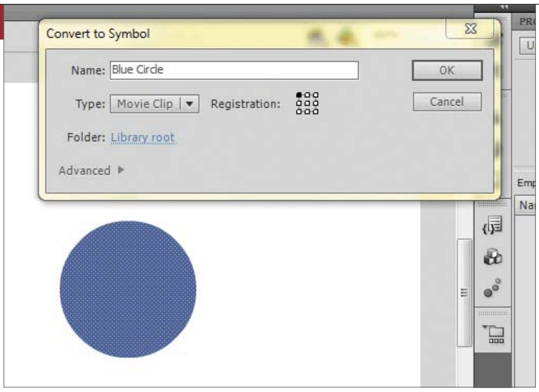
When you create a new file in Flash, you need to define which version of Action-Script you're going to work with. In addition, if you're creating an application for a mobile device, like a phone or tablet using the new AIR for Android or iOS feature, you need to specify that when you start your project. You can alter your settings later on, but knowing the ActionScript version and the deployment device ahead of time will help your workflow when working with Flash Professional CS5.5.

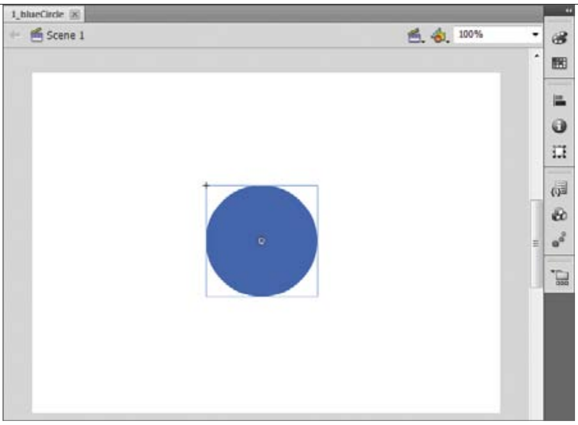
1. Launch Flash Professional CS5.5.
2. Select New from the File menu.

   You'll then be prompted with the New Project dialog box (**Figure 1.1**), where you can define the new project type.
3. Choose ActionScript 3.0 as the Type.

   The projects in this book use the ActionScript 3.0 language. You should also choose ActionScript 3.0 if you are building a basic project for a web browser.

## ACCESSING OBJECTS ON THE STAGE





When you create a new file in Flash, you create objects, place them on the Stage, and animate them using the timeline. You can access each of these items through ActionScript if you follow a couple of general rules.

**FIGURE 1.2** Convert to Symbol dialog box

**FIGURE 1.3** Blue Circle MovieClip

- You must convert the visual objects you want to access using ActionScript to a MovieClip. This is a special type of object in Flash that can be connected to ActionScript commands.
- You must give each instance a unique name.

Let's start with a simple example using a generic circle.

1. Draw a circle on the Stage using the Flash drawing tools.

   To work with the circle on the timeline, or with ActionScript, you need to convert it to a MovieClip.

2. Right-click the circle and select Convert to Symbol to convert it to a MovieClip.

   All MovieClips must have a name, which you can define in the Convert to Symbol dialog box (**Figure 1.2**).

3. Name the MovieClip **Blue Circle**. It will be listed as Blue Circle in the Library (**Figure 1.3**).

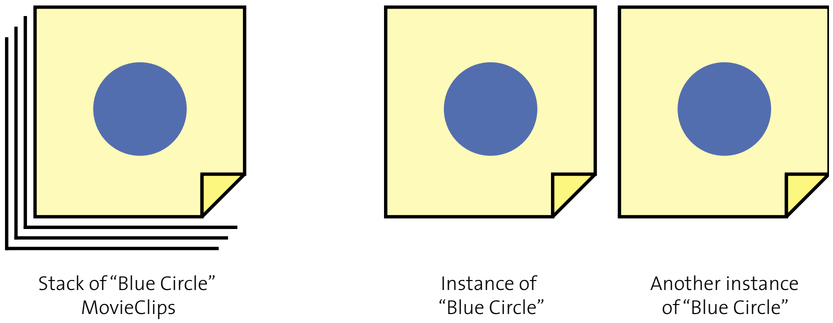

Stack of "Blue Circle" MovieClips
Instance of "Blue Circle"
Another instance of "Blue Circle"

**FIGURE 1.4** Sticky note "stacks" as Library objects versus individual notes as instances

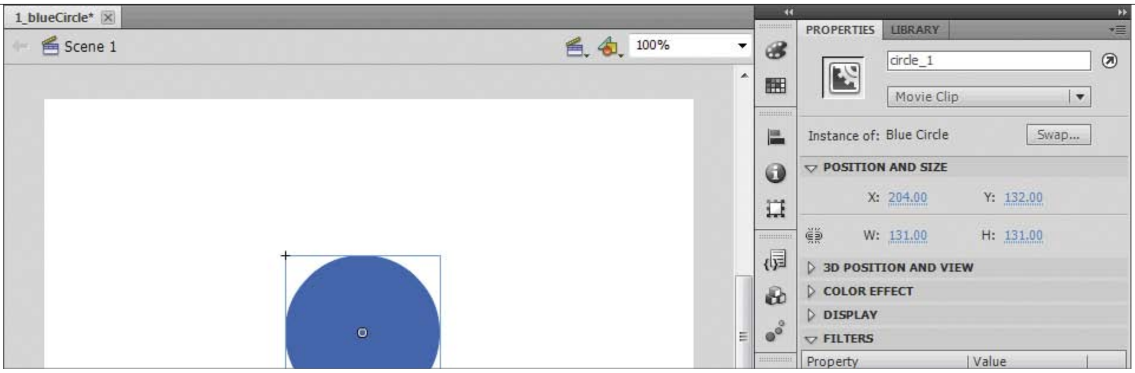
A good way to think of this is to imagine that the Blue Circle in the Library is an infinite stack of sticky notes, each looking exactly the same; in this case a moderately large blue circle. When you want to put one of these circles on the Stage, you peel off one of the sticky notes and place it on the Stage. This is called an *instance* of the symbol. No matter how many sticky notes you put on the Stage, they are all from the same sticky pad, looking exactly the same (**Figure 1.4**).

When you place the sticky note on the Stage, you need to have some way of referring to it. The name Blue Circle refers to the stack of sticky notes, not the individual instances. To refer to each instance discretely, you need to give it a unique instance name. The instance name is a special name that refers to only that specific instance.

4. Select the instance on the Stage, and open the Properties panel. The field at the top of the panel is where you enter the unique name for this instance of the circle. Name it **circle_1** (**Figure 1.5**).

   Instance names can contain letters, numbers, and underscores, but no other punctuation marks. They can never start with numbers. In addition, instance names generally are not capitalized; capitalized words in ActionScript refer to a different type of object that we'll cover a bit later.

   Now the instance of your circle (MovieClip) has a name, and you're ready to access it using ActionScript.



circle_1

Instance of: Blue Circle

X: 204.00 Y: 132.00

W: 131.00 H: 131.00

**FIGURE 1.5** Instance name defined as circle_1 in the Properties panel

## **RULES** FOR **NAMING INSTANCES** USING **CAMEL CASE**

There is a best practice for naming instances. It is called *camel case*, and it involves combining multiple words together in a format that is easy to read but is also a legal name that ActionScript will accept.

Camel case rules specify that the first letter of the instance name is lowercase, and then separate words are combined together without spaces, but each word is capitalized.

For instance, if you have “red box”, camel case rules would make that “redBox”. The first word, “red”, is not capitalized, and the second word, “box”, is capitalized, and is added after “red” without a space.

You can use this for multiple words, and it is an established best practice with developers that works in ActionScript and other languages.

## SENDING MESSAGES TO THE OUTPUT PANEL

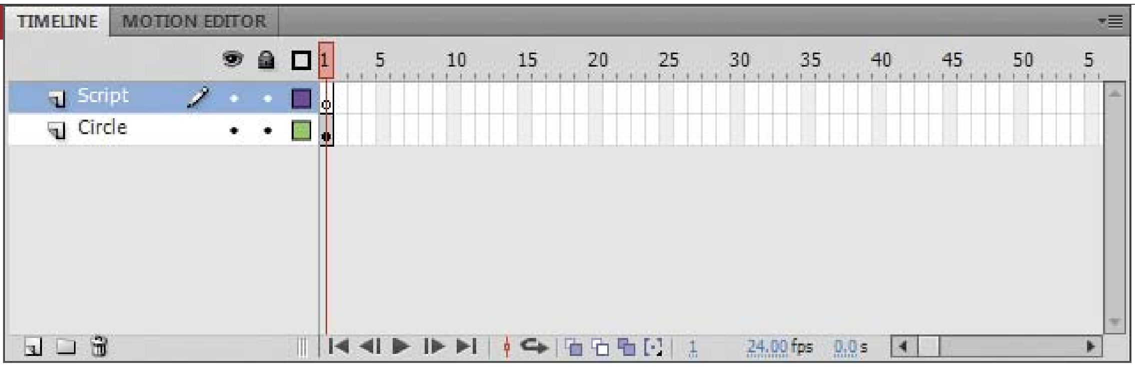


**FIGURE 1.6** Timeline layers created and named

Before you create any ActionScript, I want to point out that there is more than one way to create ActionScript in Flash Professional CS5.5. The type you are going to use initially is a *frame script*, where you create some code that you place on the timeline itself. The other method is a *class-based script*, which you'll learn about in a future chapter and will use for the rest of the book.

A frame script exists in the timeline on the keyframe of your choosing. Usually, frame scripts exist on Frame 1, with other scripts on other frames for when you want to stop or branch your animation or content. Frame scripts are usually on their own timeline *layer*. These layers refer to the various rows of timelines you can create in your project in the Timeline panel. Before you continue, name the current layer and then create a new one.

1. Double-click the current layer name and rename it **Circle**.
2. Create a new layer by clicking the new layer button at the bottom of the Timeline panel .
3. Label the new layer **Script**.

   The new layer will have a blank keyframe denoted by a hollow circle on the first keyframe; the layer containing the Blue Circle MovieClip has a filled circle (**Figure 1.6**).

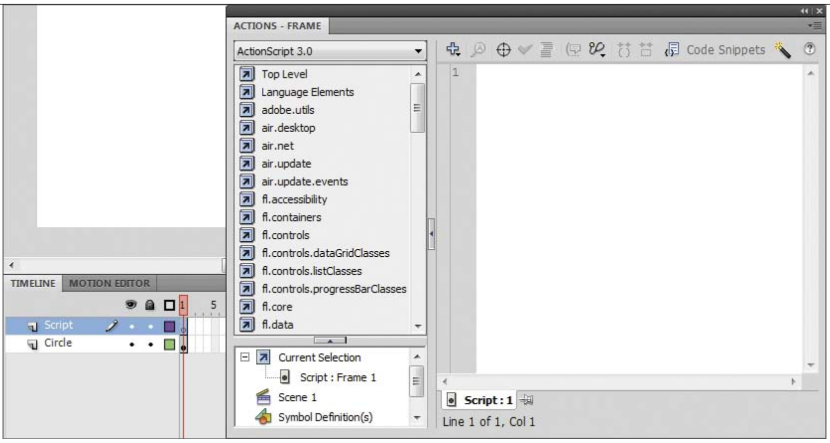


**FIGURE 1.7** Frame 1 selected and the Actions panel open

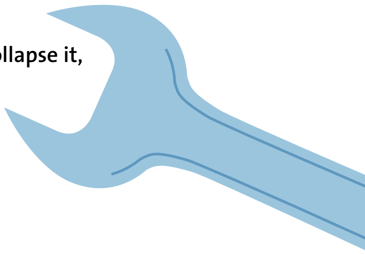
Now that you have created a layer for the scripts, you need to begin writing ActionScript using the Actions panel in Flash Professional CS5.5. The Actions panel allows you to add ActionScript to frames on the timeline. Before opening the Actions panel, you need to select where you want to place the script before writing your code.

4. Select Frame 1 in the Script layer on the timeline.

5. Open the Actions panel from the Window menu (**Figure 1.7**).

You're going to enter some code in the Actions panel that won't make much sense yet, but just follow along and everything will be explained fully in a bit.

**NOTE: You won't be using the left side of the Actions panel. To collapse it, click the disclosure icon to the right of the left column.**



6. Enter the following in the Actions panel (**Figure 1.8**):

   ```
   trace("Hello ActionScript!");
   ```

   You might see some tool tips pop up while you type. Don't worry, you can ignore them for now; just make sure that the line of code is typed and spelled correctly. In code, spelling and capitalization count, and frequently the most frustrating errors are caused by a typo, so make sure you pay attention to your spelling and capitalization.

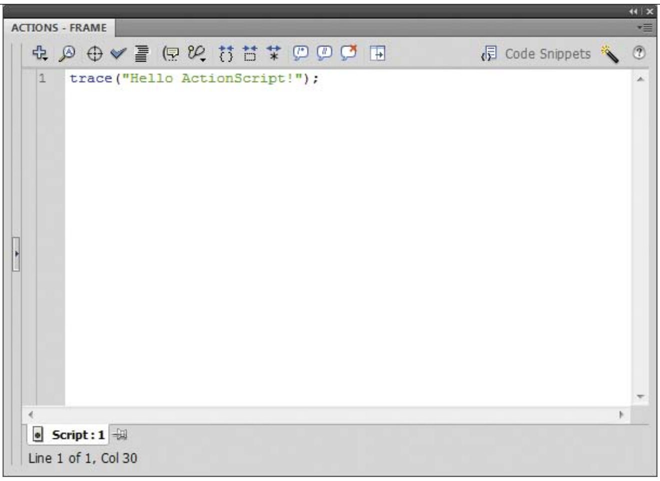
7. Open the Control menu and select Test Movie > Test, making sure that Flash Professional is selected (**Figure 1.9**). You can also use the keyboard shortcut Control-Enter (Windows) or Command-Enter (Mac).

8. Test the project. You will see a preview of the movie, with the circle, and the Output panel will open (**Figure 1.10**).

The Output panel displays the message "Hello ActionScript." The ActionScript `trace` statement takes what's in the parentheses and sends it to the Output panel. In this case, it is the contents inside the quotation marks. Note that the quotation marks aren't displayed. The quotes let ActionScript know when a piece of text begins and ends. Text is also known as a *string* in ActionScript.

The `trace` statement is helpful to send yourself messages that may help you debug issues or monitor what is going on in your application. Statements sent to the Output panel using `trace` do not appear when you publish to Flash Player or AIR; they are used only when you're working within Flash and are testing your project.

Congratulations, you have successfully written and executed your first ActionScript command! Now, if only everything were that easy, right?



```
trace("Hello ActionScript!");
```

**FIGURE 1.8** ActionScript code added to the Actions panel

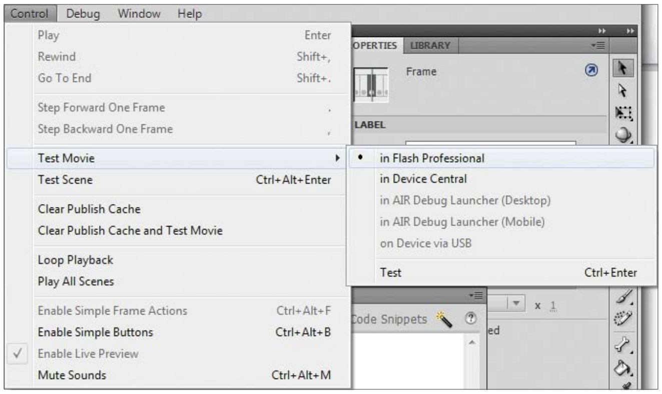
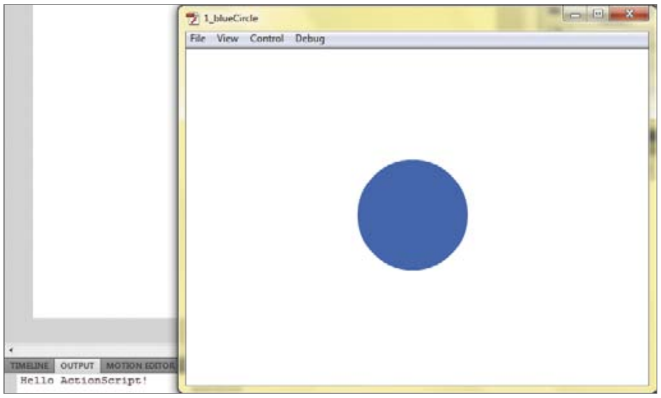


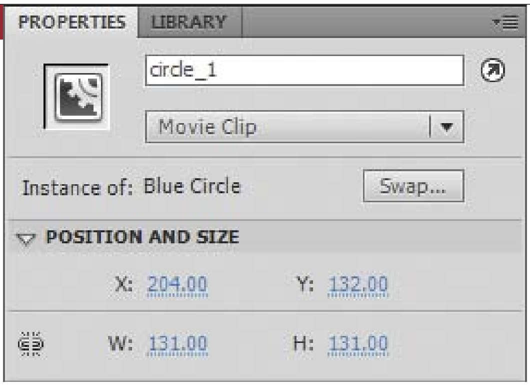
**FIGURE 1.9** Menu command to test movie



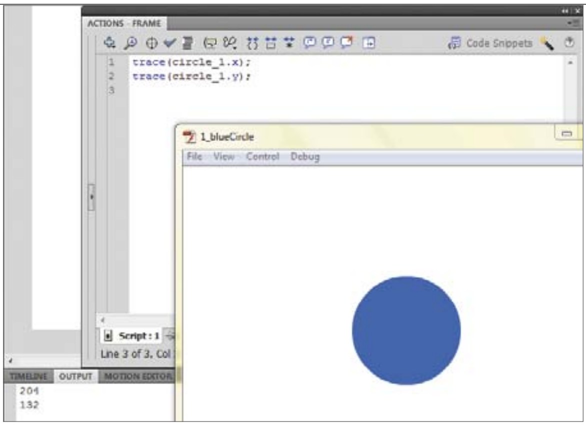
```
Hello ActionScript!
```

**FIGURE 1.10** Publish preview with Output panel opened

## WORKING WITH OBJECT PARAMETERS



circle_1
Movie Clip
Instance of: Blue Circle
X: 204.00 Y: 132.00
W: 131.00 H: 131.00



```
trace(circle_1.x);
trace(circle_1.y);
```

```
204
132
```

**FIGURE 1.11** Properties of the circle_1 instance

**FIGURE 1.12** Output panel with x and y coordinate locations

Now that you have the basics down, you'll start writing some ActionScript that will manipulate the blue circle that you have on the Stage.

Every object, symbol, graphic, and animation has properties, or attributes, that define certain parts or values. For example, the circle on the Stage has a few obvious properties right off the bat.

Based on the Properties panel, you know that the circle has a value for its width, height, and its position on the x and y axes (**Figure 1.11**).

You can access these properties using ActionScript to read their current value or to send new values that overwrite the existing ones.

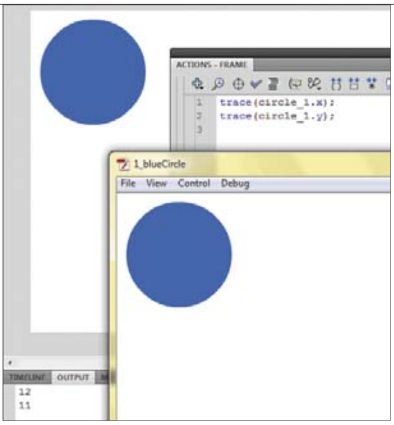
To access these, you need to identify the object you are working with, which is why you need to give all the instances unique instance names.

1. With Frame 1 selected in the Script layer, delete the `trace` statement that you placed earlier and enter in the following:

   ```
   trace(circle_1.x);
   trace(circle_1.y);
   ```

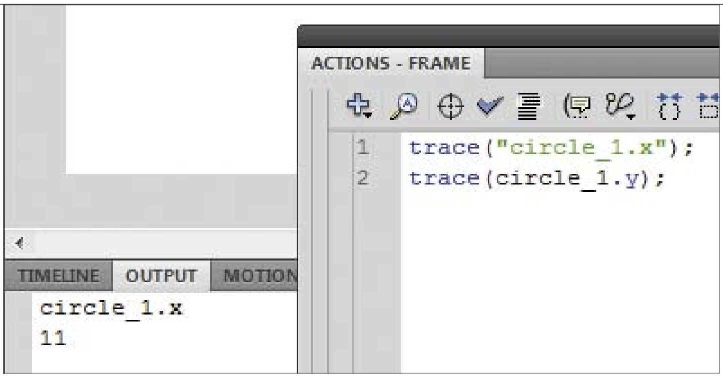
2. Run the movie from the Control menu again. You'll see the same movie, but you'll see two numbers appear in the Output panel (**Figure 1.12**).

   These two lines are accessing the circle_1 instance on the Stage, accessing the location of the object on the x axis, and sending that to the `trace` statement, which sends the value to the Output panel. It repeats the process again but for the location of the instance on the y axis.



```
trace(circle_1.x);
trace(circle_1.y);
```

```
12
11
```



```
trace("circle_1.x");
trace(circle_1.y);
```

```
circle_1.x
11
```

So you have successfully accessed the x and y properties of the instance of the circle on the Stage.

3. Move the location of the circle on the Stage and run the movie again. You'll see that the values sent to the Output panel change to reflect the x and y properties of the new location (**Figure 1.13**).

   Notice that unlike the first time you used the `trace` statement, there are no quotation marks before or after the contents in the parentheses. If you remember, the quotation marks indicate the begin and end points of a sequence of text. Instead of a discrete set of text, or string, you are accessing a stored value for the x and y coordinates. These are also called *variables.*

   Variables are containers that store data. They can be accessed or modified through ActionScript. When you refer to variables, or variable properties of instances, you don't use quotation marks. Change the first line to illustrate this.

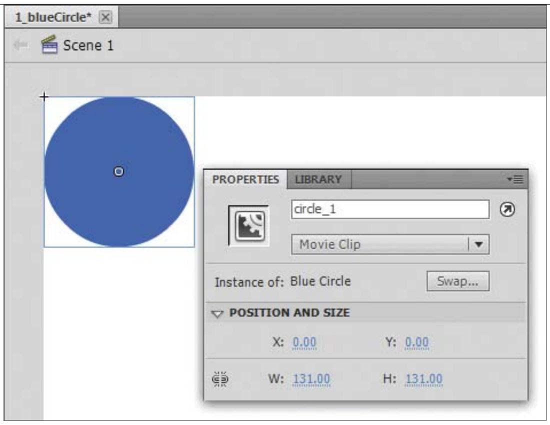
4. Insert quotation marks before and after the contents of the parentheses so your code looks like this:

   ```
   trace("circle_1.x");
   trace(circle_1.y);
   ```

5. Run the movie. Notice that the first line of the Output panel contains the text "circle_1" and then the next line contains a number (**Figure 1.14**). That is because you told the `trace` statement to use the string identified with the

**FIGURE 1.13** Output panel with x- and y-coordinate locations after moving on the Stage

**FIGURE 1.14** Output panel with string and y coordinate



circle_1

Instance of: Blue Circle

X: 0.00 Y: 0.00

W: 131.00 H: 131.00

**FIGURE 1.15** Circle repositioned at the 0,0 coordinate

quotation marks instead of accessing the x coordinate variable property of the circle_1 instance.

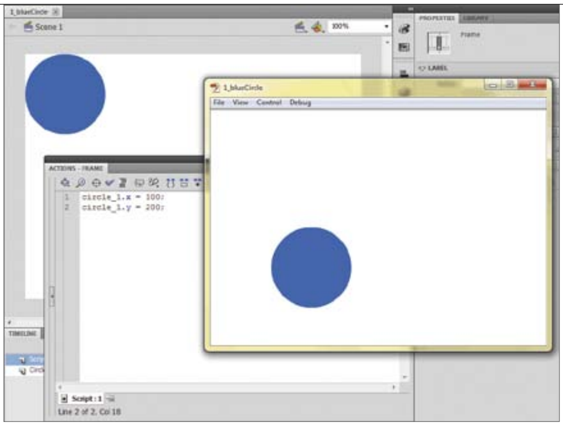
So now that you are able to access the instance's properties, you can change them using ActionScript. Before continuing though, move the object to the upper-left corner of the Stage.

6. Using the Properties panel, enter **0** for the x coordinate and **0** for the y coordinate (**Figure 1.15**).

Next, you'll overwrite the values that are stored in the Properties panel with values you'll define in ActionScript. When you run the movie, the ActionScript code will run and immediately replace the x and y values of the circle_1 instance.

When accessing values of objects, you use their names as placeholders, and ActionScript finds the values that they hold and then replaces the names with the actual values. All MovieClips have a set of properties. In this example, you have been working with the x and y properties of the object. These map to the x and y coordinates on the Stage.

When you want to change a value in a variable, you need to assign a value to it by using the assignment operator, which is an equals sign (=).



```
circle_1.x = 100;
circle_1.y = 200;
```

**FIGURE 1.16** Circle repositioned at coordinate 100,200 using ActionScript

7. In the Actions panel, replace the contents with the following code:

   ```
   circle_1.x = 100;
   circle_1.y = 200;
   ```

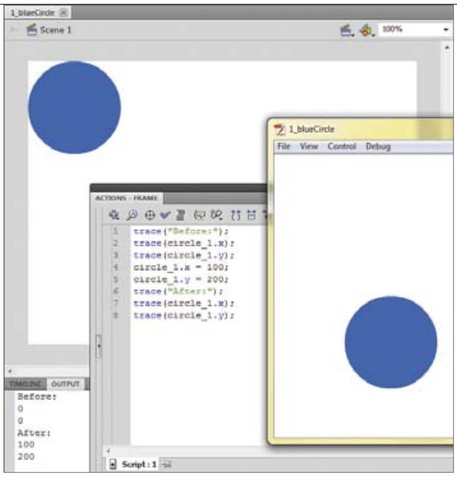
   This code accesses the x and y properties of the circle_1 instance and assigns the values on the right side of the assignment operator to them. So for the x coordinate, the x property value is 100, for the y coordinate, the y property value is 200.

   The period between `circle_1` and x and y indicates that you are accessing the x and y properties of the instance named before the period, in this case `circle_1`. This is called *dot notation*, where you can select properties of objects by chaining them together using periods.

   I should also point out the semicolons at the end of each line. The semicolons at the end of the line tell ActionScript when you have finished with a specific action. Consider them the "periods" of your ActionScript "sentences." They are required, so be sure you include them!

8. Run the movie. Notice that even though the circle was positioned at the upper-left corner of the Stage (at coordinate 0,0), the ActionScript overwrote those settings and placed the object at a new location (**Figure 1.16**).

   To prove that ActionScript is overwriting the manual settings for the instance, add some more code to track the values of the coordinates.



```
trace("Before:");
trace(circle_1.x);
trace(circle_1.y);
circle_1.x = 100;
circle_1.y = 200;
trace("After:");
trace(circle_1.x);
trace(circle_1.y);
```

```
Before:
0
0
After:
100
200
```

**FIGURE 1.17** Displaying the before and after positions of the object

9. Update the Actions panel with the highlighted lines in the following code:

```
trace("Before:");
trace(circle_1.x);
trace(circle_2.y);
circle_1.x = 100;
circle_1.y = 200;
trace("After:");
trace(circle_1.x);
trace(circle_2.y);
```

10. Run the movie. Notice that the Output panel shows the values are still 0 and 0 when the movie starts, but the assignment operator overwrites the initial values with the new coordinates. The Flash runtime then shows them at the desired location (**Figure 1.17**).

    You might ask why you didn't see the circle at the first 0,0 coordinate when the project ran. The simplest explanation is that the operation ran so fast that Flash Player didn't have a chance to display it at the first location. As you continue to work with ActionScript, you'll learn how you can time the changes to properties to display them like an animation.

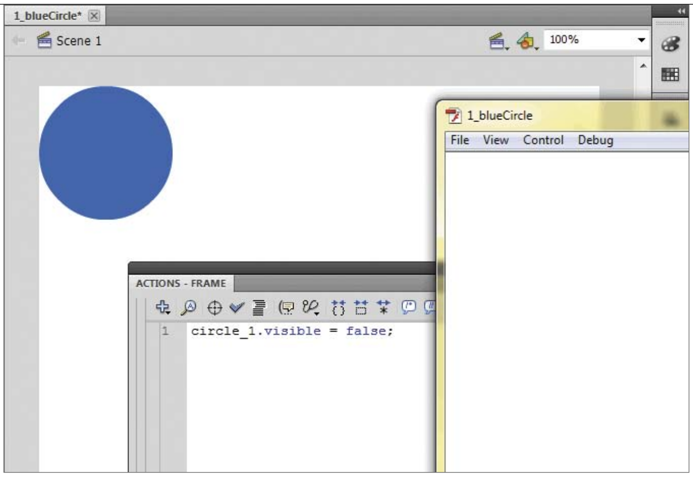
There are a lot of properties that you can modify using ActionScript. Some of the more popular ones are listed in **Table 1.1**.

There are some properties listed in Table 1.1 that use property values that are either "true" or "false." To use these, you need to assign the value `true` or `false` to the property. Take a look at the `visible` property as an example.

**TABLE 1.1** Common MovieClip properties

| PROPERTY | DESCRIPTION | VALUES |
|---|---|---|
| `.alpha` | Sets the transparency level of an object. A low value makes the object more transparent, a high value makes it more opaque. | 0 (Invisible) through 1 (fully opaque) |
| `.width` | Defines the width of an object in pixels. | Decimal number greater than 0 |
| `.height` | Defines the height of an object in pixels. | Decimal number greater than 0 |
| `.x` | Defines the x coordinate of an object, based on its registration point, in pixels. | Decimal number, positive or negative |
| `.y` | Defines the y coordinate of an object, based on its registration point, in pixels. | Decimal number, positive or negative |
| `.scaleX` | Widens an object based on a percentage value. A value of 1 (100%) keeps an object at its natural size. A value of .5 (50%) scales the width down 50%. To double the width, use a value of 2 (200%). | Decimal number greater than 0 |
| `.scaleY` | Changes the height of an object based on a percentage value. A value of 1 (100%) keeps an object at its natural size. A value of .5 (50%) scales the height down 50%. To double the height, use a value of 2 (200%). | Decimal number greater than 0 |
| `.rotation` | Rotates the object, in degrees, in a clockwise direction. | Decimal number, positive or negative |
| `.visible` | Determines if an object is visible or not. | true (visible), false (not visible) |

**FIGURE 1.18** Visibility set to `false`



```
circle_1.visible = false;
```

11. Add the following code to the end of your ActionScript:

```
circle_1.visible = false;
```

Notice that the value `false` isn't placed in quotation marks. This is a special value type called *Boolean* that you'll learn about in a future chapter.

12. Run the movie. Notice that the object is no longer visible. The instance is still there, but you told Flash Player to hide it by changing the `visible` property to be `false` (**Figure 1.18**).

## WRAPPING UP

Using basic ActionScript commands, you can get and set common properties of MovieClips that you have on the Stage. To do this successfully you need to keep the following in mind:

- Give your object an instance name using the Properties panel while the object is selected on the Stage.
- Refer to the instance name when getting or assigning a property value.
- Use dot notation to work with properties that are part of instances.
- Use the assignment operator (=) when you want to assign a value to a property of an object.
- Place semicolons at the end of each ActionScript statement to signal the end of an action to ActionScript.
- Remember to use quotation marks when displaying text, or strings, but not around instance names.

With these general rules, you can start playing with the properties for objects you have on the Stage. In the next chapter, you'll learn how to add objects to the Stage exclusively with ActionScript.

# 2

# DYNAMICALLY **ADDING OBJECTS** TO THE **STAGE**

Now that you know how to access and manipulate objects that are already on the Stage, you'll learn how to add objects dynamically from the Library using ActionScript.

The difference with this method is that the objects you are going to manipulate are not placed on the Stage in the Flash application, also called at *authortime*. Instead, you'll use objects that are in the Library, and after the application is compiled into a SWF, you'll dynamically update the Stage using ActionScript with objects in the Library.

## CREATING NAMED LIBRARY ASSETS

If you remember, in order to access the properties or attributes of items on the Stage, you needed to have an instance name that ActionScript could reference. When you place objects on the Stage from the Library, you need unique instance names as well. You need to give a name to the "stack of sticky notes" in the Library to let ActionScript take an instance of the stack and place it on the Stage.

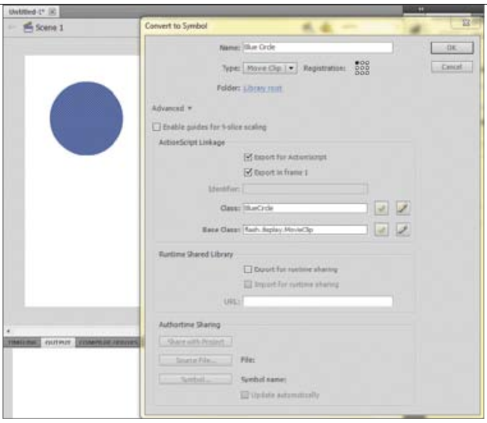
1. Create a new ActionScript 3.0 project in Flash Professional CS5.5.
2. On the Stage, create a simple circle and convert it to a symbol.
3. Name the symbol **Blue Circle** and make sure MovieClip is selected as the symbol type.

   Generally in Flash, objects that you dynamically place on the Stage need to be MovieClips. Before you click OK though, you need to give ActionScript a name it can use to place an instance of the circle on the Stage.

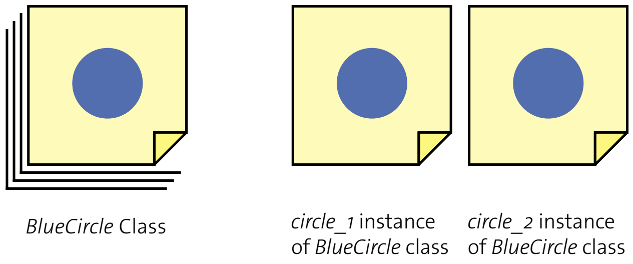
### WHAT IS THE DIFFERENCE BETWEEN A MOVIECLIP AND A GRAPHIC?

When you create a symbol, you'll notice that you have a choice of three symbol types: MovieClip, Graphic, and Button. When working with Action-Script, only MovieClips can be targeted using ActionScript. When you use the Graphic type, you cannot control it with ActionScript. Graphic symbols are for use in animations or designs that need to be encapsulated in a reusable symbol format, but aren't intended to be controlled with ActionScript.

Button is an object type as well, but it is a pared down version of a MovieClip that was commonly used in earlier versions of ActionScript. You can continue to use it, but it is far less flexible than a MovieClip, and as a best practice you should use MovieClips instead of Buttons.



Name: Blue Circle
Type: Movie Clip Registration:
☐ Enable guides for 9-slice scaling
ActionScript Linkage
☑ Export for ActionScript
☑ Export in frame 1
Identifier:
Class: BlueCircle
Base Class: flash.display.MovieClip
Runtime Shared Library
☐ Export for runtime sharing
☐ Import for runtime sharing
URL:
Authortime Sharing
Share with Project...
Source File... File:
Symbol... Symbol name:
☐ Update automatically


*BlueCircle* Class
*circle_1* instance of *BlueCircle* class
*circle_2* instance of *BlueCircle* class

4. Access the Advanced portion of the window.

   With this view open, you'll see an ActionScript Linkage section in the middle. This section is used to give the object a name that you can use to place it on the Stage.

5. Select the Export for ActionScript check box.

   The Class and Base Class fields will auto fill with **BlueCircle** and **flash.display.MovieClip**.

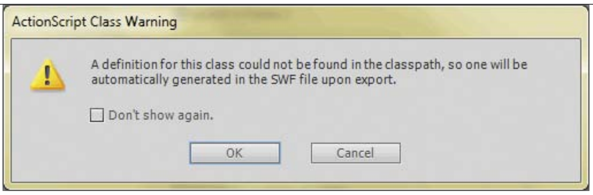
   When you're finished, your Convert to Symbol dialog box should look like the one in **Figure 2.1**.

   Unlike names given to objects in the Library, names used for ActionScript cannot contain spaces. That is why the Class field doesn't have a space. A Class is the way that ActionScript represents the "stack." The "stack" of blue circles is called the BlueCircle class. You can see this analogy if you look again at the diagram from Chapter 1 (**Figure 2.2**).

   In order for Flash to know what type of object it is, it needs to have a base class defined; in this case it's flash.display.MovieClip. You can ignore the items before "MovieClip" for now; you'll learn about those in future chapters. Just know that the stack of blue circles is now referred to as the BlueCircle class.

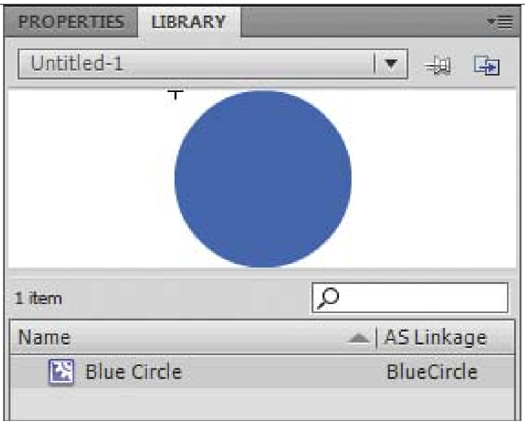
**FIGURE 2.1** Convert to Symbol dialog box, Advanced mode section

**FIGURE 2.2** Another look at the stack and instances with ActionScript names



ActionScript Class Warning

A definition for this class could not be found in the classpath, so one will be automatically generated in the SWF file upon export.



| Name | AS Linkage |
|---|---|
| Blue Circle | BlueCircle |

**FIGURE 2.3** Error message that can be ignored, for now

**FIGURE 2.4** Library panel displaying the Blue Circle object, linked to the BlueCircle class

6. Click OK. You'll get an error message (**Figure 2.3**).

   You can ignore this error message for now. In the future, you'll be creating a special ActionScript file that will define how the BlueCircle class works and behaves.

In the Linkage column of the Library, you'll see that the Blue Circle item is linked to BlueCircle, the class name you'll be using in ActionScript to create instances of the Blue Circle (**Figure 2.4**).

**NOTE:** The error message that displays is notifying you that a special ActionScript file called a Class hasn't been defined for this object. Later in the book, you'll learn how to create these files.

## INTRODUCING THE DISPLAY STACK

In Flash, there are some rules that define how items are displayed in the window. All displayed items are part of a special group called the *display stack*. The display stack is a list from which you can remove or add objects. As you create instances of objects using ActionScript, they won't be displayed until you add them to this special group.

So why have the display stack? At times, you may want to create an object and be able to modify it, but not actually display it for the user. It could be an item that isn't ready for use quite yet, or something that needs to be hidden from time to time. The object is still there, but just can't be seen.

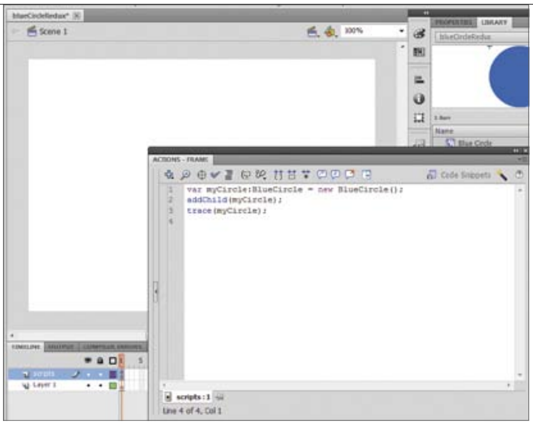
In the past, Flash users would set the `.visible` property to `false` to hide an item, or they would set the `.alpha` value to zero. Either property setting has the same effect, but if you have a significant number of items, it can slow down playback.

### CREATING NEW OBJECTS ON THE STAGE

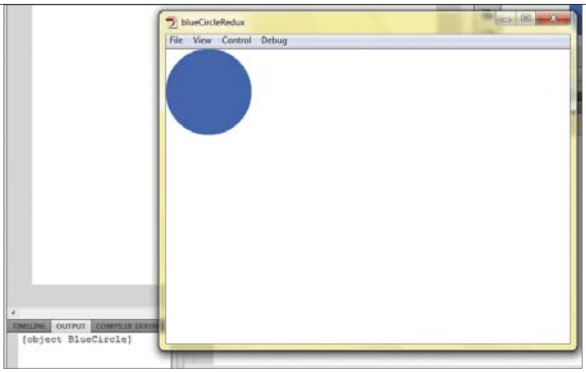
So now that you know about the display stack, it's time to write some code. Creating objects in ActionScript from the Library is pretty straightforward now that you know about classes and the display stack.

Since you are going to place the circle using just code, you need to remove any objects you have on the Stage.

1. Select the circle on the Stage in Flash and delete it.
2. Create a new timeline for your ActionScript code in your now-empty ActionScript 3.0 project.
3. Name the timeline **scripts.**
4. Select the empty frame in the new scripts timeline.
5. Open the Actions panel.



```
var myCircle:BlueCircle = new BlueCircle();
addChild(myCircle);
trace(myCircle);
```



```
[object BlueCircle]
```

**FIGURE 2.5** Actions and Timeline panels with new code added

**FIGURE 2.6** Dynamically placed BlueCircle object

6. In the Actions panel, type in the following code:

```
var myCircle:BlueCircle = new BlueCircle();
addChild(myCircle);
trace(myCircle);
```

   When you're finished, your project should look like the one shown in **Figure 2.5**:

7. Run the code.

   You'll see that part of your blue circle is in the upper-left corner of the display (**Figure 2.6**). This is because the registration point for the circle is in the upper-right corner, and the Flash runtime positions the circle based on the coordinate 0,0, which is the default location for objects when you create them.

So let's step through the code to see how it all works. The first line

```
var myCircle:BlueCircle = new BlueCircle();
```

can be read like this: A new object named myCircle has been created, which is an instance of the BlueCircle class. That object is assigned a new instance of the BlueCircle class. The process is shown in **Figure 2.7**.

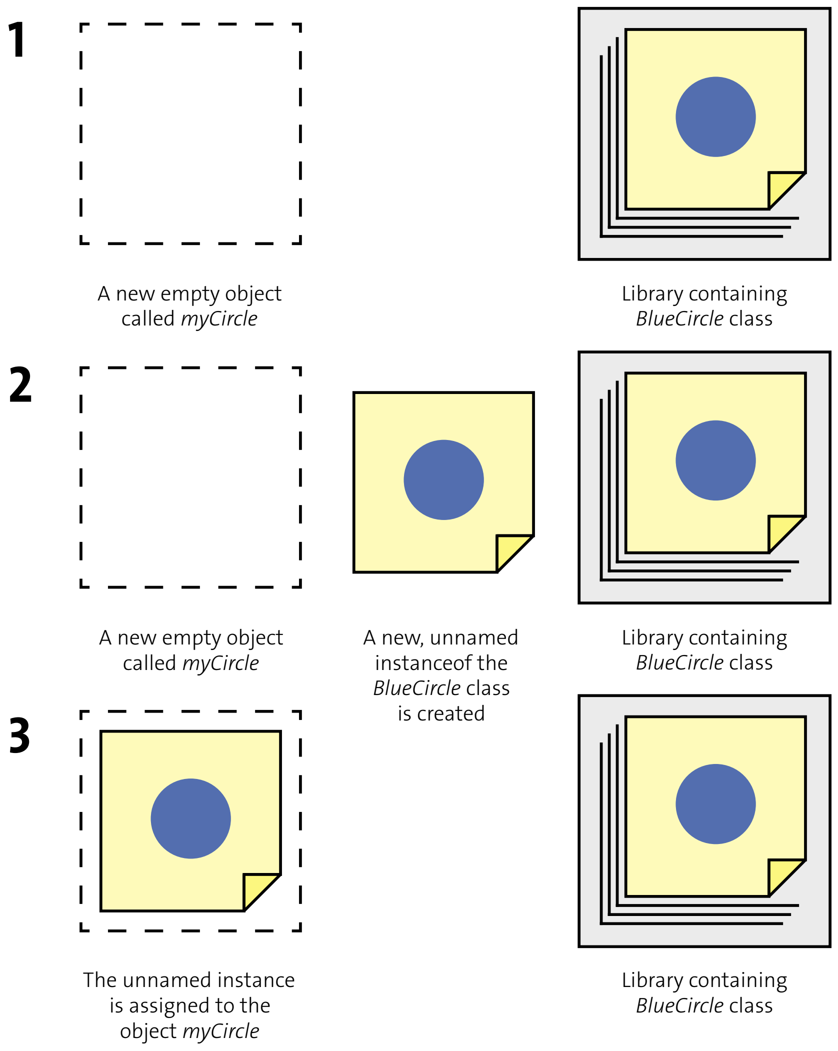

1
A new empty object called *myCircle*
Library containing *BlueCircle* class
2
A new empty object called *myCircle*
A new, unnamed instanceof the *BlueCircle* class is created
Library containing *BlueCircle* class
3
The unnamed instance is assigned to the object *myCircle*
Library containing *BlueCircle* class

**FIGURE 2.7** Creating a new object instance using ActionScript

In this illustration, the first row shows that you are creating a new named object called myCircle. You'll learn more about the `var` statement in an upcoming chapter, but just remember that the name of the new instance you created is called myCircle. You also have the BlueCircle "stack of sticky notes," or class, available in the Library to use.

The second step says that you are "tearing off a sticky note" from the stack using the `new` statement, creating a new instance of the BlueCircle class. At this point, the instance doesn't have a name, but it exists in your program.

The third step takes that unnamed instance of the BlueCircle class and assigns it to the named object myCircle. You need to do this in order to send any actions or make property changes on the object since it requires an instance name before you can work with it.

This process is the same as dragging an object from the Library, placing it on the Stage, and giving it an instance name of myCircle.

Take a look at the second line of code:

```
addChild(myCircle);
```

This line of code adds the shiny new blue circle to the display stack so it can be visible to the user. The `addChild` statement takes the item referenced in the parentheses, and adds it to the list of items that are displayed.

When the Flash runtime plays, it continually checks the contents of the display stack and renders the objects within it on the screen. Now that the blue circle is in the display stack, you can see it on the screen.

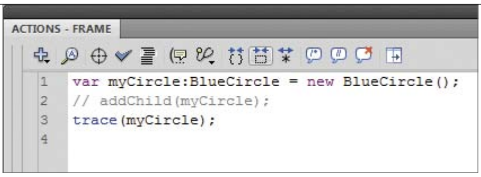
Take a look at the last line and the unusual text that appears in the Output panel:

```
trace(myCircle);
```

The last line traces the entire object to the Output panel. Notice that the panel displays the following:

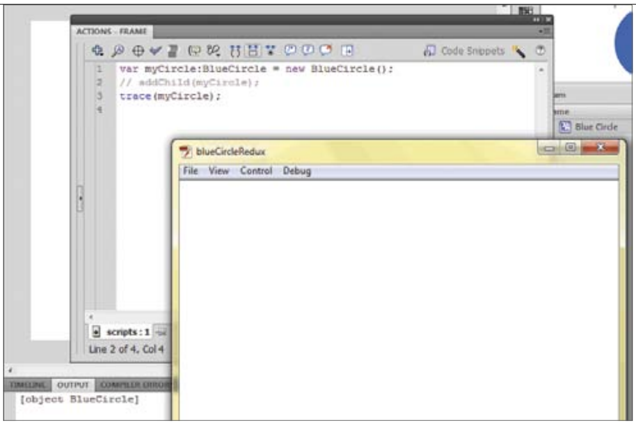
```
[object BlueCircle]
```

Since the object itself doesn't have any meaningful textual or numerical value, this line of code is sending a statement that the object being referenced is an object of the BlueCircle class. The `trace` statement then sends that to the Output panel.



```
var myCircle:BlueCircle = new BlueCircle();
// addChild(myCircle);
trace(myCircle);
```

**FIGURE 2.8** Showing comment coloring in the Actions panel



```
var myCircle:BlueCircle = new BlueCircle();
// addChild(myCircle);
trace(myCircle);
```

```
[object BlueCircle]
```

**FIGURE 2.9** Result when removing `addChild` from the code

## MESSING WITH THE DISPLAY STACK

Now, you'll update the code to show what the display stack does.

1. Change the code to read like this:

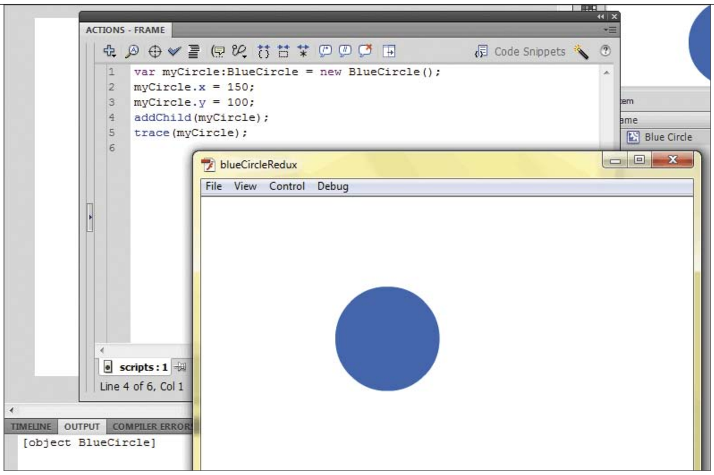
   ```
   var myCircle:BlueCircle = new BlueCircle();
   // addChild(myCircle);
   trace(myCircle);
   ```

   When you add the two slashes before the second line, you are doing what programmers call "commenting out." You'll learn more about using comments in the next section, but for now just know that the two forward slashes hide the code from Flash, so it skips it, ignoring any commands or actions on that line. Notice that the code in Flash turns grey. That is the default color for showing comments (**Figure 2.8**).

   Without the `addChild` statement, the myCircle object is not added to the display stack.

2. Run this code. You'll see an empty page.

   Check the Output panel. You'll see that the `trace` statement still works as expected because the object is still there; it just isn't part of the display stack (**Figure 2.9**).



```
var myCircle:BlueCircle = new BlueCircle();
myCircle.x = 150;
myCircle.y = 100;
addChild(myCircle);
trace(myCircle);
```

```
[object BlueCircle]
```

**FIGURE 2.10** Change of location after modifying the x and y properties

This is a common mistake that a lot of new coders make: creating an instance of an object and forgetting to add it to the display stack. You'll see that you get no errors when you run the project, because the object is there, but you haven't specifically told Flash to render it on the screen.

3. Add the `addChild` statement back in by removing the leading double slashes.
4. Run the program again. You'll see that you restored the object.

## ASSIGNING PROPERTIES TO DYNAMICALLY CREATED INSTANCES

If you want to position the object at a certain location when you place it, you can assign values to the instance properties x and y after the object is created.

1. Update the code to assign values to these properties after the line that creates the initial instance with the `new` statement:

```
var myCircle:BlueCircle = new BlueCircle();
myCircle.x = 150;
myCircle.y = 100;
addChild(myCircle);
trace(myCircle);
```

2. Display the object. Its location will be at the coordinate 150,100 (**Figure 2.10**).

# WORKING WITH COMMENTS

Comments are a huge part of your programming, and are critical for you and your team members to understand the code that you are writing. Comments allow programmers to add notes, hints, and explanations within their code, so they or others can understand it.

Even if you are working alone, comments are extremely important to help you understand your code after you stop working on a project for a while and need to get back to it. Many developers wrestle with confusing code that made perfect sense when they originally wrote it, but can't remember how it worked when they went back to it months later.

## CREATING COMMENTS

You have two basic ways to comment in Flash. The first is a single-line comment. To create one, start the line of code with a double forward slash, `//`. Any contents after these, on that line only, are hidden by Flash and aren't executed. For example:

```
// This code creates a blue circle on the screen
var myCircle:BlueCircle = new BlueCircle();
addChild(myCircle);
trace(myCircle);
```

In this example, the first line is a comment, and is ignored by Flash when it runs the code.

The second type of comment is a multi-line comment. To create this type, you need to mark the beginning and ending of the comment using two special two-character sequences: `/*` and `*/`. You'll wrap the text you want to hide from Flash using these two sequences. Here is an example:

```
/* This code creates a blue circle
    on the screen at 150,100 */
var myCircle:BlueCircle = new BlueCircle();
myCircle.x = 150;
myCircle.y = 100
addChild(myCircle);
trace(myCircle);
```

The contents inside the opening /* and the closing */ are hidden from Flash, and the contents between them can be as long as you want. You might see some code use these opening and closing symbols in various ways, but they always do the same thing: start and end a multi-line comment.

### WORKING WITH SPACES

Generally, you can add whitespace characters (spaces, tabs, and carriage returns) in your code. Sometimes these are helpful to segment related pieces of code together, usually with a comment above the segment to identify what the code does. There are some best practice rules for using tabs in your ActionScript, which you'll learn later in the book.

The blue circle example could be expanded to make the code's function very clear and explicit:

```
/* This code creates a blue circle
    on the screen at 150,100 */
// Create a new instance of the BlueCircle class
var myCircle:BlueCircle = new BlueCircle();
// Position the myCircle on the screen
myCircle.x = 150;
myCircle.y = 100
// Add myCircle to the display stack
addChild(myCircle);
// Send myCircle to the Output panel to confirm it is there
trace(myCircle);
```

## WRAPPING UP

Using a few new ActionScript commands, you are able to dynamically add objects to the Stage. When working with object instances using ActionScript, make sure you do the following:

- Make sure that the object in your Library is named and has been configured for "Export to ActionScript" using the Advanced mode section of the Convert to Symbol dialog box.
- Create a named object to hold your new instance using the `var` statement.
- Generate a new instance of the object from the Library using the `new` statement, assigning the object to the named object using the = assignment operator.
- Add the object to the Flash display stack using `addChild` to render the object on the screen.
- Remember to add comments using single-line or multi-line comments to document what your code does for others to understand, or for you to understand when you return to your project later.

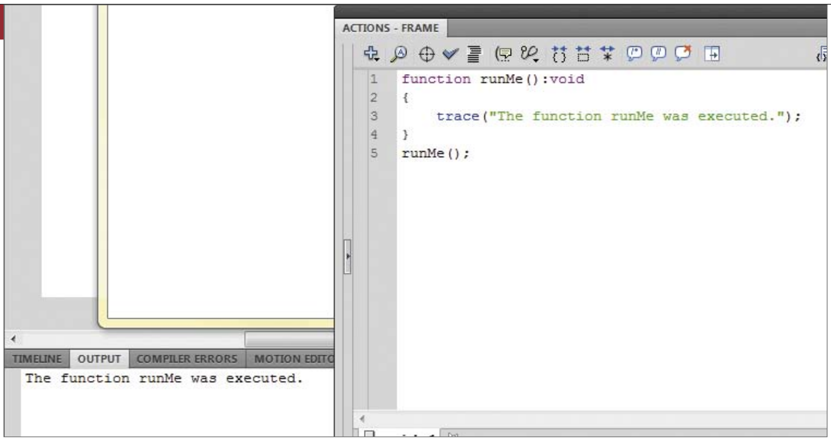
3

# WORKING WITH FUNCTIONS

When you work with ActionScript, there are sequences of code that you may want to execute multiple times. To do this, you need a way to group the code into a logical block and give it a name that will tell the Flash runtime to execute the code.

Functions are the way to do exactly that. Using functions, you can group commonly used commands for repetitive use throughout your application. In this chapter, you'll learn the commands and syntax to create a basic function, and then learn how you can extend the use of your functions to send and receive data in and out of them.

## FUNDAMENTALS OF FUNCTIONS



```
function runMe():void
{
    trace("The function runMe was executed.");
}
runMe();
```

```
The function runMe was executed.
```

**FIGURE 3.1** Results from your first function

To get started with functions, let's look at the following example.

1. Create a new ActionScript 3.0 Flash file and enter the following code into the frame script on the timeline:

```
function runMe():void
{
    trace("The function runMe was executed.");
}
runMe();
```

2. Run the code. The phrase "The function runMe was executed" appears in the Output panel in Flash (**Figure 3.1**).