Article 1

Converting from ActionScript to ActionScript 2.0

by William B. Sanders

DEVELOPERS WHO HAVE USED ACTIONSCRIPT from its inception have seen it grow in size and complexity from a few actions to a complete lexicon rivaling even the most developed scripting languages. It has eclipsed languages like JavaScript 1.5 in its scope and size. (ActionScript 2.0 will look very much like JavaScript 2.0, though.) In this latest version of ActionScript, Macromedia has made enough fundamental changes to warrant a change in reference from ActionScript to ActionScript 2.0. All previous versions of ActionScript, but most significantly the version released with Flash MX, are now called ActionScript 1.0.

ActionScript 2.0 can be called an object-oriented programming (OOP) language, whereas previous versions were more modestly referred to as an object-*based* programming language, and that was only with the Flash MX version. Such distinctions are perhaps more hair-splitting than necessary because OOP is as much as an attitude toward programming as it is the built-in features of a language used to generate OOP structures in a script. (See Article 5, "Understanding Object-Oriented Programming in Macromedia Flash MX 2004," by Jeffery Tapper) The change in ActionScript's form is due to the OOP embedded in the ECMA-262 standard, out of which emerged

ECMAScript Edition 4—the proposal for Internet scripting languages. (The latest version of ECMAScript standards is referred to as "Edition" or "proposal" by ECMA. A shorthand for ECMAScript proposal 4 is ECMAScript 4.)

For those familiar with the early days of personal computers, you may remember that every different brand of computer had its own version of the BASIC programming language. Among others were Atari, Commodore, Texas Instruments, Timex-Sinclair, Apple, and IBM. A program developed on an Atari could not be run on an IBM. Instead of coming up with a standard for all PC BASIC languages, the different companies just went their own way until the only ones left were Apple and IBM (running Microsoft BASIC). When the Internet came along, Netscape developed a new language called JavaScript that ran on its browser. Microsoft then developed a slightly different version of JavaScript to run on its browser, Internet Explorer, and so the developers had to write different scripts for the two different browsers.

Wisely, the European Computer Manufacturers' Association (ECMA) encouraged that a standard be developed so that the Internet would not become the Tower of Babel that the early personal computer languages were. Because ECMA sets standards for a number of different computer-related tasks, they number them, and ECMA-262 represents the standard for Internet scripting languages, most notably JavaScript. However, instead of calling it JavaScript, they called it ECMAScript, and the current edition is ECMAScript 4.

The new version of JavaScript under development to conform with ECMAScript 4 is called JavaScript 2.0, and so Macromedia followed suit and called the ECMA standard-meeting version of ActionScript "ActionScript 2.0." (See `http://www.mozilla.org/js/language/es4/index.html` for the full standard.) To be sure, you will find differences between ActionScript 2.0 and the ECMAScript 4 standard. For example, Flash has certain built-in non-ECMA classes such as MovieClip and LoadVars because it *remains* the scripting language for Flash and not something more generic. Otherwise, the language is very close to the ECMAScript 4 standard.

So why bother to adopt a standard that has nothing to do with Flash? Well, ActionScript could either remain a language unlike any other, or it could adopt a way of programming that is familiar to other programmers. By taking the latter course of action, ActionScript can be learned readily by programmers who already know OOP languages such as C++ and Java. Likewise, because JavaScript 2.0 is taking the same route, JavaScript 2.0 programmers will be able to pick up ActionScript quickly as well.

The key to this new direction is the object-oriented model of programming, and by making ActionScript more OOP-like and suitable for object-oriented programming, ActionScript provides a stronger base for well-structured programs with code that is:

- Reusable
- Modular
- Scalable
- Maintainable
- Secure
- Robust

Beginning with Flash MX and the introduction of *prototype* structures, ActionScript began migrating toward the ECMASCript 4 proposal, and that journey is now more complete with ActionScript 2.0. So although the change from ActionScript 1.0 to ActionScript 2.0 is a major one, the result should not be wholly unfamiliar.

As you read the rest of the article, you will find a discussion of what is new with ActionScript 2.0 and tools related to the new ActionScript. First, the article examines the new tools and procedures that affect all ActionScript programming in Flash. You will learn about changes to the Actions panel and the addition of the Behaviors panel and integrated Script window. Also in the initial section you will learn about the new case-sensitivity in writing script and how to create classes and strong data typing. No matter what you're doing with ActionScript 2.0, these new procedures and tools will affect the results.

Second, you will find that several new terms have been added to ActionScript 2.0. Some of these terms are new statements, objects, or general properties. For example, MovieClipLoader is a new class. Other terms, though, are properties or methods added to existing terms, such as the Systems object's new methods and properties. In addition, this section shows whole new ways of doing things in ActionScript 2.0. One of the major additions is the use of a subset of Cascading Style Sheets (CSS) to style text.

Finally, the article briefly examines using ActionScript 2.0 with components. You will find very few of the Flash MX ActionScript techniques for styling components still extant. Using ActionScript 2.0, you will find other fundamental approaches to using components as well. For example, instead of using the `setHandler()` method, listeners are now employed as event detectors.

The purpose of this section is to expand on and more fully examine what you will find in the main dictionary. It also serves as a quick look-up of what's new in ActionScript 2.0. Because the bulk of ActionScript 2.0 is virtually identical to ActionScript 1.0, if you already know ActionScript, this article will show the critical differences and new procedures to be followed, which will save you time and frustration. When you finish this article, you should have a good understanding of the new direction ActionScript 2.0 takes and how to use and integrate it with ActionScript 1.0.

Actions Panel

One of the first things you will notice about the Actions panel is that it is no longer divided into expert and normal modes. In most respects, the expert mode was really a normal mode, and the normal mode was really a learning mode. Most developers never used the normal mode and were glad to see it eliminated. However, many designers relied heavily on it, and even developers used it occasionally to straighten out code formatting.

To get a quick overview of the new Actions panel and the new ActionScript terms, open up a new page using File > General Tab > New Flash Document and press F9 to open the Actions panel. (You will notice that you can open a lot more than just a document.) Now select File > Publish Settings > Flash and change the version to Flash Player 6 and the ActionScript version to ActionScript 1.0. By doing so, all the new ActionScript terms in the Actions toolbox will be highlighted in yellow. Figure 1.1 shows what you can expect to see in the Actions toolbox when new terms are encountered.

The Actions panel itself has changed little from the expert mode of Flash MX. However, some important changes have been introduced, including the following:

- Placement of the Script navigator in the bottom left pane
- Addition of tabs to select ActionScript in different frames and objects
- Placement of the Script Pin button at the bottom of the panel
- Addition of code line markers (red dots that can be toggled on and off by clicking the line numbers)

Figure 1.1 By setting the Publish Settings to the previous version of Flash, all the new terms are highlighted in yellow.

Otherwise, the Actions panel is the same as in Flash MX. Figure 1.2 shows the panel with the Publish Settings set to ActionScript 2.0 and Player 7 with code using the new NetConnection and NetStream classes.

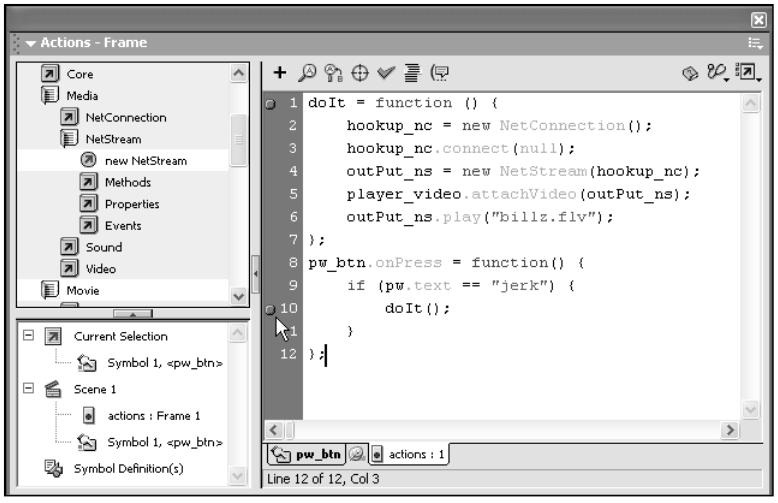

Figure 1.2 Key changes can be found in the new Actions panel.

Designers who have become accustomed to using the helpful normal mode can either transition to the new Actions panel using code hints to help, or they can use the new Behaviors panel. For additional help in creating code and as a substitute for the normal mode, the next section introduces the new Behaviors panel that generates code for specific tasks.

Behaviors Panel Replacing Normal Mode

To aid those who are not developers, the new Flash includes a Behaviors panel. Rather than just helping the user to write code, the Behaviors panel writes chunks of code to accomplish different tasks. For example, the `gotoAndPlay()` and `gotoAndStop()` actions need some kind of event to fire them. Using the Behaviors panel, the user just has to select the object or frame that will use the code, and the Behaviors panel does the rest. For example, consider a typical Flash application that is designed to give the user a choice using buttons. One choice will be correct and the other incorrect, and so one button will order the playhead to go to and stop at the incorrect answer frame and the other to go to and stop at the frame representing the correct answer. Follow these steps to create a simple quiz movie using the Behaviors panel:

1. Open a new Flash document.
2. Create a total of three layers, naming them from top to bottom Items, Buttons, Background.
3. Add a total of ten frames to each layer, and add keyframes in Frames 5 and 10 of each layer.
4. Select the first frame of the Buttons layer, create a button object, and make a copy of it. Place one above the other.
5. Click the first frame of the Items layer, and in the Actions panel, type `stop();`.
6. In the Background layer in the first frame, type the question, **Q: Which version of ActionScript has classes?** Next to the top button, type **ActionScript 1**, and next to the bottom button, type **ActionScript 2**.
7. In the Items layer, click on Frame 5, and in the middle of the stage, type **Incorrect.** Then, click on Frame 10 and type **Correct** in the middle of the stage.
8. Click on the first frame of the Buttons layer and click on the top button. Open the Behaviors panel, click on the "+" button to open a popup menu, and select Movieclip > Goto and Stop at frame or label, as illustrated in Figure 1.3.

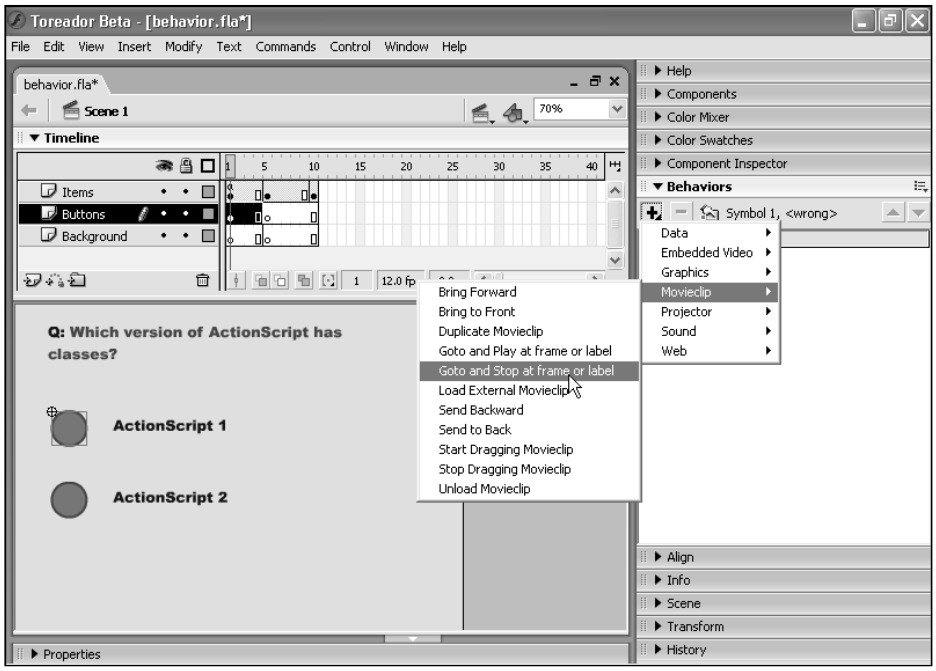

Figure 1.3 Different sets of code can be selected from the Behaviors panel.

9. The Goto and Stop at frame or label dialog box opens. Type the number 5 in the Frame(label) text box, as shown in Figure 1.4.

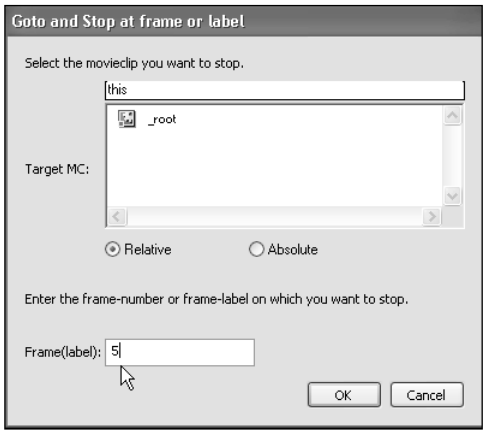

Figure 1.4 All the user has to include is the frame number to create the necessary script.

10. Repeat Steps 8 and 9 for the bottom button, but type 10 for the frame instead of 5. That's it. All the necessary script for the quiz is complete.

When using the Behaviors panel to create code, you cannot see the script, but it is being generated and placed in the Actions panel. To see the code, select the top button and press F9 to open the Actions panel. Figure 1.5 shows the code generated by the Behaviors panel for the top button:

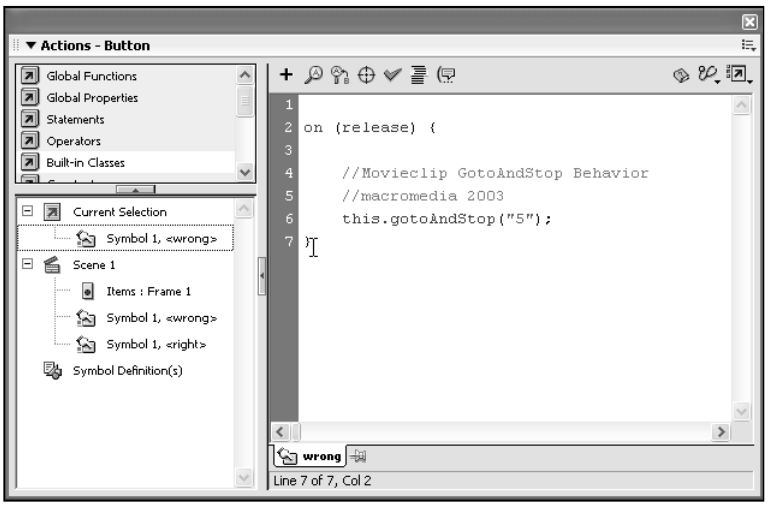

Figure 1.5 Scripts generated with the Behaviors panel appear in the Actions panel.

Although many may miss the helpful structure elements in the normal mode from previous versions of Flash, the Behaviors panel makes it very easy to generate code in much bigger chunks.

When the Behaviors panel generates code, it does so in a case-sensitive manner. As you will see in the next section on case sensitivity, all ActionScript 2.0, no matter how it is generated, (unlike ActionScript 1.0) is case sensitive.

Case Sensitivity

In ActionScript 1.0, all code is case-insensitive. That is, a variable named `alpha` can be accessed as `Alpha` or `aLpHa` or any combination of cases that spell "alpha." However, in ActionScript 2.0, the code is case-sensitive. For example, if you type the code

```
var alpha="Call me, Alphie"
trace(Alpha);
```

instead of seeing "Call me, Alphie" in the Output panel, you will see "undefined." That's because `Alpha` is not defined, even though `alpha` is. If your ActionScript 1.0 files have not paid attention to case sensitivity, you will have to rewrite them if you want to convert your code into files that will run on the Flash 7 player. Otherwise, they will only be able to run on the Flash 6 player.

The change from case-insensitive to case-sensitive code is a *crucial* matter, and until you get used to being sure that all references, definitions, and assignments are case-sensitive, you are likely to encounter bugs in your scripts.

In the same way that ActionScript 2.0 requires you to pay attention to cases in naming and referencing, it has also tightened up on the way data types are employed. The next section explains the entire process of data typing, and although it is not difficult to understand, it requires you to pay attention to a detail not required with ActionScript 1.0.

Strong Data Typing

You may not have been aware of the fact that all data typing prior to ActionScript 2.0 uses *weak data typing*. That means variables and properties can be assigned different types of data at different times, and ActionScript is perfectly happy. That is good because if you want to change a variable from a numeric variable to a string variable, all you needed to do was assign the variable a string value. For example, in ActionScript 1.0 the following script works just fine:

```
var alpha = 521;
trace(2 + alpha); //Output shows: 523
var alpha = "Bloomfield Rocks!";
trace (2 + alpha); //Output shows: 2Bloomfield Rocks!
```

The code also works in ActionScript 2.0, but it's a fairly sloppy practice because the variable changes from one type to another. Also, none of the type checking will be in place. Using *strongly typed data*, however, you are required to include the type of data when you declare a variable. For example, the following declaration creates a numeric variable:

```
var priceItem:Number = 14.95;
```

The variable named `priceItem` is now a numeric (number) variable and cannot be a string or some other kind of variable unless it is retyped. ("Retyping" refers to *providing* a new data type and not keyboard efforts.)

However, if you enter

```
var item:Number = "Hot Dog";
```

you will encounter an error message because the assigned value "Hot Dog" is a string instead of a number.

If you are familiar with languages like Java, numeric variables are broken down into finer categories such as double and integer, but ActionScript 2.0 simply uses Number for any type of numeric value. You can assign the following data types in ActionScript 2.0:

- Accorion Alert
- Array
- Binding
- Boolean
- Button
- Camera
- CheckBox
- Color
- ComboBox
- ComponentMixing
- CustomActions
- DataField
- DataGrid
- DataHolder
- DataSet
- DataType
- Date
- DateChooser
- Delta
- DeltaItem
- DeltaPacket
- Endpoint
- Error
- Function
- Label
- LoadVars
- LocalConnection
- Log
- MediaController
- MediaDisplay
- MediaPlayback
- Menu
- MenuBar
- Microphone
- MovieClip
- MovieClipLoader
- NetConnection
- NetStream
- Number
- Object
- PendingCall
- PopUpManager
- PrintJob
- ProgressBar
- RadioButton
- RadioButtonGroup
- RDBMSResolver
- ScrollPane
- SharedObject
- Slide
- SOAPCall
- Sound
- String
- TextArea
- TextField
- TextFormat
- TextInput
- TextSnapshot

- Tree
- TypedValue
- Video
- Void
- WebServiceConnector
- Window
- XML
- XMLConnector
- XMLNode
- XMLSocket
- XUpdateResolver

In addition, all built-in classes and all custom classes and interfaces can be data types as well.

Most of the new features, including strong typing and the use of classes and OOP, are best understood by seeing how they work in the new integrated Script window, a feature only in the Professional version of Flash. So we now will turn to a discussion of how to use this new editing tool to optimize your use of ActionScript 2.0.

Using the Integrated Script Window to Create a Class

To facilitate coding in ActionScript 2.0, there are now two different scripting windows. First, you have the familiar Actions panel, which is almost identical to the expert mode of the Actions panel in Flash MX. The second one, called the Integrated Script window, works very much like a text editor, such as Notepad. You can write code in it that is saved and used outside of the FLA or SWF file, and that code is addressed and used in the Actions panel. However, unlike Notepad, the Integrated Script window has many of the error-checking and formatting features of the Actions panel. In fact, most of the code editing features found in the Actions panel are duplicated in the Integrated Script window.

You can write external scripts in the Integrated Script window and bring them into the movie using the `#include` action. However, the key purpose of this new script editor is to create classes. Flash MX uses the `prototype` and the `registerClass` functions to create classes, but that method is a bit awkward and divergent from the way classes are created in most other OOP-based languages. So, now classes are created in separate .as files written in the Integrated Script window. A single class is all that can be created in a single .as file, and the .as file name must match the class name. Furthermore, the .as file must be in the same folder as the FLA file that uses the class. The information in the .as file is compiled into the SWF file. Generally, development takes place with the FLA, SWF, and .as file all residing in the same folder, as shown in Figure 1.6.

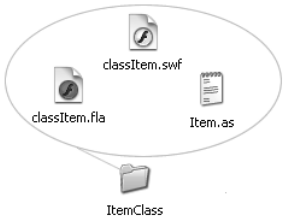

Figure 1.6 Class definitions saved as .as files must be in the same folder as the .FLA files that use the classes.

When you define a class, you can make it part of the authoring environment by placing it in the Classes folder within the First Run folder. The full path in Windows XP, for example, is Program Files > Macromedia > Flash 2004 > en > First Run > Classes. So if you create a class you're likely to use in several different programs, you can save time by placing it in the Classes folder.

To see how to use the Integrated Script window and create and use a class in ActionScript 2.0, follow these steps to create a simple movie:

1. Open a new Flash document and create a total of three layers, naming them from top to bottom Actions, Text field, and Background.
2. In the Text field layer, add two dynamic text fields next to one another, naming the one on the left itemOut and the one on the right checkOut.
3. In the Background layer, type the label **Product** over the left text field and the label **Total** over the right text field. Save the file as classItem.fla in a folder named ItemClass.
4. Select File > New > ActionScript File and enter the following script in Listing 1.1:

Listing 1.1 **Creating a Class**

```
class Item {
     var describe:String;
     var price:Number;
     var tax:Number;
     function Item(describe, price, tax) {
          this.describe = describe;
          this.price = price;
          this.tax = tax;
     }
function selectedItem() {
          return describe;
     }
     function kaChing():Number {
          return price += (price*tax);
     }
}
```

5. Save the file as Item.as in the ItemClass folder. Note that the class name "Item" must match the name of the file. Figure 1.7 shows the completed Integrated Script window with the Actions panel ghosted in the foreground.

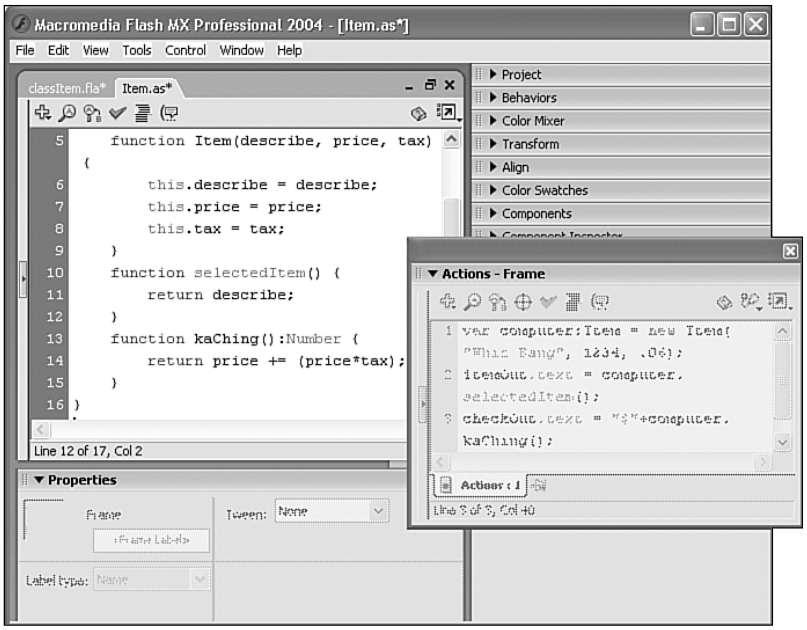

Figure 1.7 All class definitions must be created in the Integrated Script window.

6. Click the classItem.fla tab at the bottom of the Integrated Script window to select the FLA document and open the Actions panel. Type the following script:

```
var computer:Item = new Item("Whiz Bang", 1234, .06);
itemOut.text = computer.selectedItem();
checkOut.text = "$"+computer.kaChing();
```

Figure 1.8 shows the FLA document window with the related code and stage. Note the tab in the upper left corner to toggle back to the Integrated Script window. (The Macintosh version of Flash has separate windows with no tabs.)

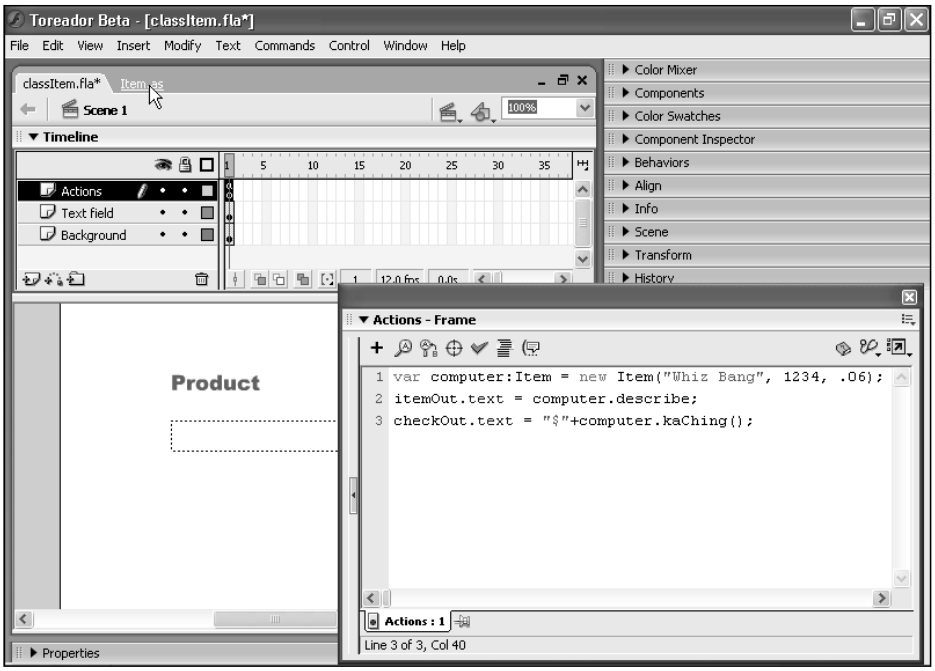

Figure 1.8 Only the Actions panel can be used to instantiate the class developed in the Integrated Script Window.

7. Save the script and press Ctrl+Enter to test the movie (Command-Return on the Mac). You should see the current value of the `Item.describe` property in the left window and the output of the `Item.kaChing()` method in the right window. Figure 1.9 shows the output of testing the movie. Note the tabs in the lower left corner of the figure.

After going through all that for the simple output and calculations, the new system for creating classes may seem to be a step backwards in both productivity and coding. However, one of the main purposes of OOP is to encourage reusable code. The Item class can be called by any script simply by placing the Item.as file in the same folder as the SWF file calling it. By creating a library of classes in this fashion, production time can be reduced considerably.

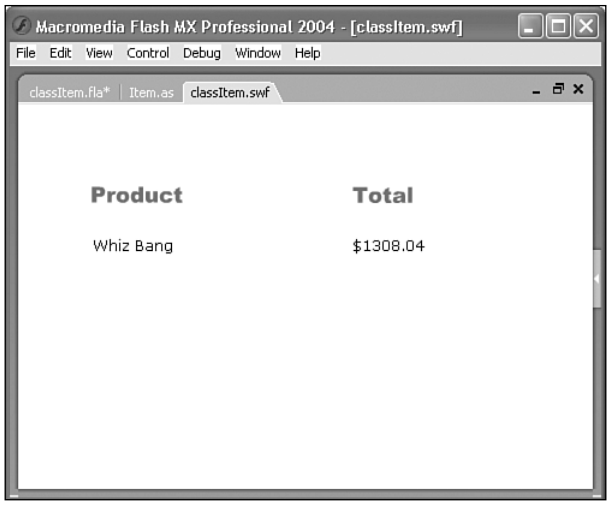

Figure 1.9 The output requires both the code from the .as file and the Actions panel.

Using the New Terms and Term Groups

In addition to new tools and procedures, ActionScript 2.0 has several new terms, and the terms are grouped differently in the Actions toolbox in the Actions panel. The groupings of the different terms are now in the following top-level folders in the Actions toolbox:

- Global Functions
- Global Properties
- Statements
- Operators
- Built-in Classes
- Constants
- Compiler Directives
- Types
- Deprecated
- Data
- Components

In rethinking the top-level folders and their subfolders, Macromedia completely restructured the organization of the Actions toolbox. So, instead of being able to provide a set of equivalence folders to Flash MX, you have to go through the

different folders and see how the terms are organized. Some folders like Constants and Operators are almost identical to Flash MX, and Objects in Flash MX are now Built-in Classes. However, Global Functions and Global Properties have very little in common with the Functions and Properties folders in Flash MX.

In discussing the new terms below, each one is identified by the path to its folder and subfolder. Each is discussed with examples to supplement its discussion in the main dictionary portion (Part II, "Dictionary") of this book.

NetConnection and NetStream

One of the most important new features of ActionScript 2.0 is the addition of terms that can be used to stream Flash Video (FLV) files. The streaming is actually a progressive download but appears indistinguishable from streaming with Flash Communication Server. (This feature is available only in the Professional version of Flash.) However, this is an important new feature because it means that you no longer have to place an external video directly into a Flash movie or movie clip. Instead, all you have to do is call up the video like you would an external text or graphic file.

Two key objects are involved: NetConnection and NetStream (Built-in Classes > Media). NetConnection has a single method, `connect`. However, NetStream is a bit more robust, and its features are summarized in Table 1.1.

Table 1.1 **NetStream Properties, Methods and Event**

Methods	**Description**
`close()`	Stops play altogether, and resumption of play starts at the beginning of the file.
`pause()`	A toggle method that stops and resumes play of stream in position where stop/start occurs.
`play()`	Begins streaming play to designated output device.
`seek(n)`	Moves to the stream position in seconds (`n`).
`setBufferTime(n)`	Establishes the number of seconds (`n`) the stream is placed into buffer before dropping frames.
Properties	**Description**
`bufferLength`	Current number of seconds in the buffer.
`bufferTime`	Seconds specified in the `setBufferTime()` method.
`currentFps`	Frames per second in current stream.
`time`	How long in seconds the stream has been playing.
Event handler	**Description**
`onStatus`	Whenever an error or status change occurs, this event fires.

To see how the two new media objects work in a Flash movie, this next example uses all the objects' methods, properties, and events. All the code is in Frame 1. Figure 1.10 shows the stage running a video with the different objects and their instance names.

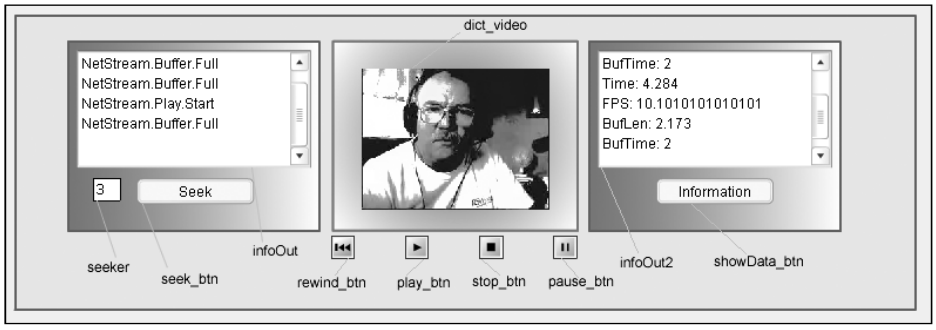

Figure 1.10 A good deal of information about the status of a stream is available using the NetStream class.

The embedded video object shown in Figure 1.10 is named `dict_video` and is shown as a rectangle with an "X" in the middle when viewed on the development stage. The two text boxes to the left and right of the video are TextArea components, but the text field named `seeker` is simply an input text field. The Seek and Information buttons are Button components, and the four buttons beneath the video are modified button symbols found in Window > Other Panels > Common Libraries > Buttons > Playback. (The pause button was created by removing the arrow from the playback-loop button and replacing it with the parallel vertical bars.)

Add five layers—Actions, Video, Text fields, Buttons, and Background. Place the elements shown in Figure 1.10 on an 800×600 stage and add the following script in Listing 1.2 to Frame 1 of the Actions layer.

Listing 1.2 **Playing an FLV File**

```
//Make a connection
var hookup_nc:NetConnection = new NetConnection();
hookup_nc.connect(null);
//Create a NetStream instance
var showTime_ns:NetStream = new NetStream(hookup_nc);
//Attach the NetStream to the video on stage
dict_video.attachVideo(showTime_ns);
//Set buffer to 2 seconds
```

continues

Listing 1.2 **Continued**

```
showTime_ns.setBufferTime(2);
//Play video
play_btn.onPress = function() {
      showTime_ns.play("dict.flv");
};
//Rewind
rewind_btn.onPress = function() {
      showTime_ns.seek(0);
};
//Stop video
stop_btn.onPress = function() {
      showTime_ns.close();
};
//Pause video (toggle)
pause_btn.onPress = function() {
      showTime_ns.pause();
};
//Seek position
seek_btn.label = "Seek";
seekVid = new Object();
seekVid.click = function() {
      showTime_ns.seek(parseInt(seeker.text));
};
seek_btn.addEventListener("click", seekVid);
//Check the current NetStream status
showTime_ns.onStatus = function(info) {
      infoOut.text += info.code+newline;
      infoOut.vPosition = infoOut.maxVPosition;
};
showStream = new Object();
showStream.click = function() {
      infoOut2.text += "Time: "+showTime_ns.time+newline;
      infoOut2.text += "FPS: "+showTime_ns.currentFps+newline;
      infoOut2.text += "BufLen: "+showTime_ns.bufferLength+newline;
      infoOut2.text += "BufTime: "+showTime_ns.bufferTime+newline;
};
showData_btn.addEventListener("click", showStream);
showData_btn.label = "Information";
```

The only other piece required is an FLV file. I created one using Sorensen Squeeze, but they can be generated using Flash Communication Server directly from camera input as well. If you have an existing movie file, such as AVI or MOV, you can use Flash to transform it to an FLV file. Import the file into the Library panel, select the icon in the Library panel, and right-click (Control-click on the Mac) it to open the context menu. In the context menu, select Properties > Export, and the movie will be saved in FLV format.

ID3

MP3 files may contain information tags known as ID3 tags. These tags were originally 256 bytes, but now the latest version of the tag, known as ID3v2, can be up to 256 megabytes. Located at the beginning of the audio file, the ID3v2 tag can have a good deal of information, including encapsulated pictures. In the latest version of Flash, you can access up to 39 of the properties that make up ID3v2 files, as shown in Table 1.2.

Table 1.2 **ID3v2 Properties**

Property	Description	Property	Description
COMM *comment	Comment	TALB *album	Album/Movie/Show title
TBPM	beats per minute	TCOM *artist	Composer
TCON *genre	Content type	TCOP	Copyright message
TDAT	Date	TDLY	Play list delay
TENC	Encoded by	TEXT	Lyricist/Text writer
TFLT	File type	TIME	Time
TIT1	Content group description	TIT2 *songname	Title/song name/ content description
TIT3	Subtitle/ Description refinement	TKEY	Initial key
TLAN	Language(s)	TLEN	Length
TMED	Media type	TOAL	Original album/ movie/ show title
TOFN	Original filename	TOLY	Original lyricist/ text writer
TOPE	Original artist/ performer	TORY	Original release year
TOWN	File owner/licensee	TPE1 *artist	Lead performer Soloist
TPE2	Band/orchestra/ accompaniment	TPE3	Conductor/performer refinement
TPE4	Interpreted, remixed, or otherwise modified by	TPOS	Part of a set
TPUB	Publisher	TRCK *track	Track number/Position in set
TRDA	Recording dates	TRSN	Internet radio station name
TRSO	Internet radio station owner	TSIZ	Size
TSRC ISRC	International Standard Recording Code	TSSE	Software/Hardware and settings used for encoding
TYER *year	Year	WXXX	URL link frame

*Backward compatibility to IDv1 properties in Flash 6 player.

The two new ID3 terms are `Sound.id3.property` and `onID3`, (Built-in Classes > Media > Sound > Objects). Both terms are used with the Sound object using MP3 files. The new terms can be employed to provide the viewer more information about an MP3 file when it begins to be played.

To see how to use both the `id3` property and the `onID3` event handler, the following example uses three objects on the stage (the italicized word is the instance name).

- 1 button *showIt*
- 1 input text field *feature*
- 1 dynamic text field *showFeature*

The whole movie can be done in two layers. Place the button and text field in the bottom layer and the code in the top layer. When you test the script, use the property names from Table 1.2 to examine the different elements of the MP3 file embedded in the ID3 tag (see Listing 1.3).

Listing 1.3 **ID3 Feature Display**

```
showIt.onPress = function() {
     coolBreeze_sound = new Sound();
     id3Feature = feature.text;
     coolBreeze_sound.onID3 = function() {
           showFeature.text = coolBreeze_sound.id3[id3Feature];
     };
     coolBreeze_sound.loadSound("chillyJazz.mp3", true);
};
```

Context Menu

The context menu appears when a menu-sensitive element is right-clicked (Control-clicked on the Mac) in a Flash movie. ActionScript 2.0 provides objects, properties, methods, and event handlers for context menus. The ContextMenu and ContextMenuItem classes are at the crux of the context menu usage in ActionScript 2.0. Table 1.3 provides a summary of the objects and their respective properties, methods, and event handlers.

Specialized context menus and their specified items can be added to any `MovieClip.menu`, `Button.menu` or `TextField.menu`. After instantiating a ContextMenu class, using the `customItems` array property, you can add different labels (captions) and callbacks to the menu that uses the ContextMenuItem class.

Table 1.3 **ContextMenu and ContextMenuItem Class Properties, Methods and Event Handlers**

ContextMenu	Description
`new ContextMenu ([callBackFunction])`	Constructor.
`copy()`	Copies the specified ContextMenu class.
`hideBuiltInItems()`	Hides the built-in items.
`builtInItems`	Object members corresponds to built-in menu items.
`customItems`	An undefined array containing ContextMenuItem classes.
`onSelect`	The callback handler called when the context menu object invoked.
ContextMenuItem	**Description**
`new ContextMenuItem (caption, callBack, [separatorBefore,] [enabled,] [visible])`	Constructor.
`copy()`	Copies the specified ContextMenuItem class.
`caption`	Provides a label for the menu item.
`enabled`	Boolean value that determines whether item is enabled or not.
`separatorBefore`	Boolean value that determines if a separator bar appear above the menu item.
`visible`	Boolean value that determines whether the item is visible or not.
`onSelect`	The callback handler called when the menu item is selected.

To see how to use the context menu terms, the following example uses a single movie clip with the instance name `star_mc`. Place a star-shaped movie clip in the middle of the stage, and in Frame 1, add the following script in Listing 1.4.

Listing 1.4 **Movie Clip Context Menu**

```
//Create Context Menu
var starMen_cm = new ContextMenu();
//Hide the built-in items in context menu
starMen_cm.hideBuiltInItems();
//Add custom items
starMen_cm.customItems.push(new ContextMenuItem("Left Corner", lefty));
starMen_cm.customItems.push(new ContextMenuItem("Right Corner", righty));
starMen_cm.customItems.push(new ContextMenuItem("Center", center));
```

continues

Listing 1.4 **Continued**

```
//Set up functions that fire with item selection
function lefty(obj:Object, item) {
     star_mc._x = 0;
     star_mc._y = 0;
}
function righty(obj:Object, item) {
     star_mc._x = 550;
     star_mc._y = 400;
}
function center(obj:Object, item) {
     star_mc._x = 165.2;
     star_mc._y = 90.5;
}
//Connect the menu to Object.menu
_level0.star_mc.menu = starMen_cm;
```

When you test the movie, place the cursor and right-click (Control-click on the Mac) on the movie clip object. As you can see in Figure 1.11, a context menu appears with the new items added in the script. Even though most of the built-in items are hidden because of the line `starMen_cm.hideBuiltInItems()`, in the script, the Settings and Debugger items remain. (The Debugger menu item will appear only if you have the debugger plug-in installed in your browser.)

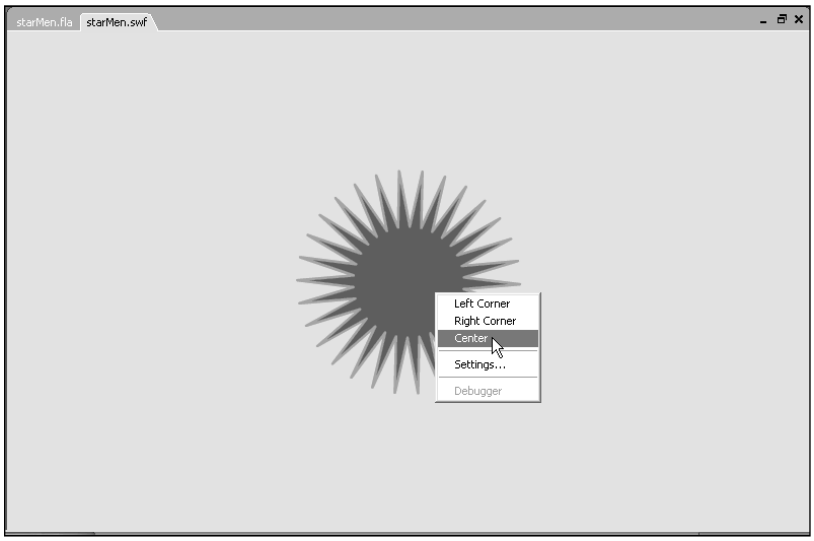

Figure 1.11 The customized menu items can be used to launch any function required.

TextField.StyleSheet

An important addition to ActionScript 2.0 is the TextField.StyleSheet built-in class. Using this new class, you can create a style sheet or even use a Cascading Style Sheet (CSS). The new class has five methods and a callback handler, as shown in Table 1.4.

Table 1.4 **TextField.StyleSheet Methods and Callback Handler**

Method	Description
`getStyle(styleName)`	Gets the named style (for example, "bodyText") and its property names and values.
`getStyleNames()`	Gets the array containing the style names.
`load()`	Begins loading CSS file into StyleSheet.
`parseCSS(cssSheet)`	Parses CSS in cssSheet and loads style sheet.
`setStyle(name, style)`	Inserts a new style with [`name`] and given characteristics [`style`].
Callback Handler	**Description**
`onLoad`	Used in conjunction with `StyleSheet.load()` indicating a successful load with a Boolean true.

The importance of using CSS style sheets is that they are standardized as part of the ECMA proposal, not only for HTML, but for XML and other ECMA languages. At this time, though, Flash supports only a subset of the full CSS set of formats. Table 1.5 shows the CSS formatting terms and acceptable values.

Table 1.5 **CSS Terms and Flash Values**

CSS Format	Flash Assignment and Values
`text-align`	Alignment to left, center, and right.
`font-size`	Use only numeric value. (Points/pixels work the same.)
`text-decoration`	Only values are none and underline.
`margin-left`	Use only numeric value. (Points/pixels work the same.)
`margin-right`	Use only numeric value. (Points/pixels work the same.)
`font-weight`	Recognized values are normal and bold.
`font-style`	Recognized values are normal and italic.
`text-indent`	Use only numeric value. (Points/pixels work the same.)
`font-family`	Mono, sans-serif, and serif are available. Mono is converted to _typewriter (courier-like), sans-serif is converted to _sans (Arial-like), and serif is converted to _serif (Times-like).
`color`	Use #hhhhhh hexadecimal values only. Names (for example, green) are not supported.
`display`	Only values are inline, block, and none.

To see how to integrate CSS with a Flash movie, first create a CSS file. Listing 1.5 uses a Flash-legal set of terms and associated values. (Save the file as dictStyle.css.)

Listing 1.5 **CSS Style Sheet**

```
.highLight {
         color: #983803;
         font-family: sans-serif;
         font-weight: bold;
         font-size: 11;
}
.bodyFont {
         margin-left: 9;
         font-family: serif;
         color: #000000;
         font-size: 12;
}
.header {
         color: #983803;
         text-align: center;
         font-weight: bold;
         font-size: 16;
         font-family: sans-serif;
}
```

To apply the CSS external style sheet, it must be loaded using the `TextField.StyleSheet.load("fileName.css")` method. So, first create an instance of the TextField.StyleSheet class, and then using the instance, use the `load()` method to set the style to the contents of the CSS file. To check to see if the CSS file loads successfully, you can use the `onLoad` event handler to create a function that passes a Boolean true if the file loads. In Listing 1.6, a small dynamic text field (instance name "loadMe") and a large dynamic text field (instance name "praise") serve to display whether the file loaded successfully and to display the CSS-formatted text on the screen.

Listing 1.6 **Applying CSS Style Sheet to Text**

```
//Create StyleSheet
var willStyleSheet = new TextField.StyleSheet();
willStyleSheet.onLoad = function(cssUp) {
     if (cssUp) {
          _level0.loadMe.text = "CSS is Up";
     } else {
          _level0.loadMe.text = "Load Error";
     }
};
```

```
praise.styleSheet = willStyleSheet;
var speech:String = '<p class=\"header\">Mark Antony on Making a Point</p>';
speech += '<p class=\"bodyFont\">';
speech += "<br>I am no orator, as Brutus is;\n";
speech += "But, as you know me all, a plain blunt man, \n";
speech += "That love my friend; and that they know full well \n";
speech += "That gave me public leave to speak of him:\n";
speech += "For I have neither wit, nor words, nor worth,\n";
speech += "Action, nor utterance, nor the power of speech,\n";
speech += 'To stir men\'s blood: I only speak <span class=\"highLight\"> right on;
➥</span>\n';
speech += "I tell you that which you yourselves do know;";
speech += "</p>";
praise.htmlText = speech;
willStyleSheet.load("dictStyle.css");
```

Figure 1.12 shows the output. Note the different fonts and font styles used in the output.

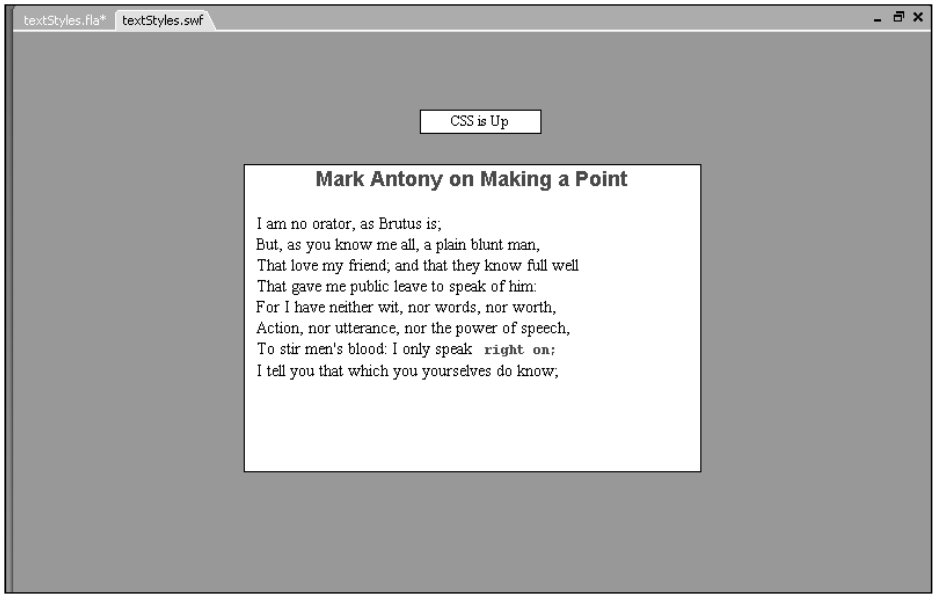

Figure 1.12 Text is formatted using an external CSS Style Sheet.

In addition to using external CSS style sheets, ActionScript 2.0 enables the user to employ the same style sheet characteristics using a special set of terms similar to those in Table 1.5. Table 1.6 shows the Flash inline terms used to assign CSS-like values to different text properties.

Table 1.6 **Flash CSS Inline Terms**

textAlign	fontSize	textDecoration	marginLeft
marginRight	fontWeight	fontStyle	textIndent
fontFamily	color	display	

To see how to employ the inline CSS terms, Listing 1.7 shows how to create a user-class and how to assign a value to a tag. Also, it illustrates how to use some of the other ActionScript 2.0 terms associated with the TextField.StyleSheet class.

Listing 1.7 **Inline Styles Applied to Text**

```
//Dynamic Example
var dynamicSheet = new TextField.StyleSheet();
dynamicSheet.setStyle("a:hover", {color:"#009966"});
dynamicSheet.setStyle(".noted", {fontFamily:'Verdana', fontSize:24,
fontWeight:'bold'});
createTextField("dreamField", 0, 200, 150, 200, 35);
dreamField.html = true;
dreamField.styleSheet = dynamicSheet;
dreamField.htmlText = "<p class='noted'><a
href='http://www.sandlight.com'>Sandlight</a></p>";
```

When you test the movie, you will see that a single word, "Sandlight," appears in the middle of the page. When you pass the mouse pointer over it, it turns from black to green, demonstrating the hover effect. The "noted" class is applied over the <a> tag style.

PrintJob Class

The PrintJob class is the central class for printing out dynamically generated materials, including databases and user-generated information. The class has a constructor and three methods shown in Table 1.7.

Table 1.7 **PrintJob Methods**

Method	Description
addPage()	Specifies the print target and optional print area, a bitmap Boolean, and frame number.
send()	Spooled pages sent to printer.
start()	Opens the OS's print dialog window and initiates spooling.

Of these three methods, only `addPage()` needs more than a little explanation. At its most simple use, a single parameter prints the entire current page. For example, the following script prints the entire page:

```
printNow_pj = new PrintJob();
printNow_.pj.addPage(0);
```

However, you can specify a movie clip as a target and further specify what area is to be printed. For example, in the PrintJob script below, the print area for the movie clip is

```
{xMin:0, xMax:225, yMin:0, yMax:200},
```

Those parameters outline the upper left print area beginning at 0,0 and the lower right at 225,200 of the target—not the page. The `addPage()` bitmap parameter is an option triggered by `{printAsBitmap:true}` that enables bitmap printing for those pages with bitmap graphics. If the option is not used, the default printing uses vector graphics.

Listing 1.8 shows how a user can enter dynamic information through UI Components (InputText and Radio Buttons), which is then displayed to the screen in a movie clip and printed out (see Figure 1.13):

Listing 1.8 **Printing Dynamic Data**

```
//Initialize Radio Buttons
la_rb.label = la_rb.data="Los Angeles";
ny_rb.label = ny_rb.data="New York";
at_rb.label = at_rb.data="Atlanta";
bf_rb.label = bf_rb.data="Bloomfield";
//Hide Print Button
purchase_btn._visible = false;
//Register
register_btn.onRelease = function() {
     //Generate Registration number
     var conNum:Number = Math.round(Math.random(1)*1000000);
     level0.ticketClip_mc.confirm.text = "Confirmation number: FL-AS2"+conNum;
     //Place information into MC that will be printed
     ticketClip_mc.ticket.text += name_ip.text+newline;
     ticketClip_mc.ticket.text += address_ip.text+newline;
     ticketClip_mc.ticket.text += CtStZp_ip.text+newline;
     ticketClip_mc.ticket.text += email_ip.text+newline;
     ticketClip_mc.ticket.text += "Location: "+radioGroup.selection.data;
     //Show Print Button
     purchase_btn._visible = true;
};
//Print
purchase_btn.onRelease = function() {
```

continues

Listing 1.8 **Continued**

```
    ticketPrint_pj = new PrintJob();
    var confirmRes = ticketPrint_pj.start();
    confirmRes = ticketPrint_pj.addPage("ticketClip_mc", {xMin:0, xMax:225,
    ➥yMin:0, yMax:200}, {printAsBitmap:false}, 1);
    ticketPrint_pj.send();
    delete ticketPrint_pj;
};
```

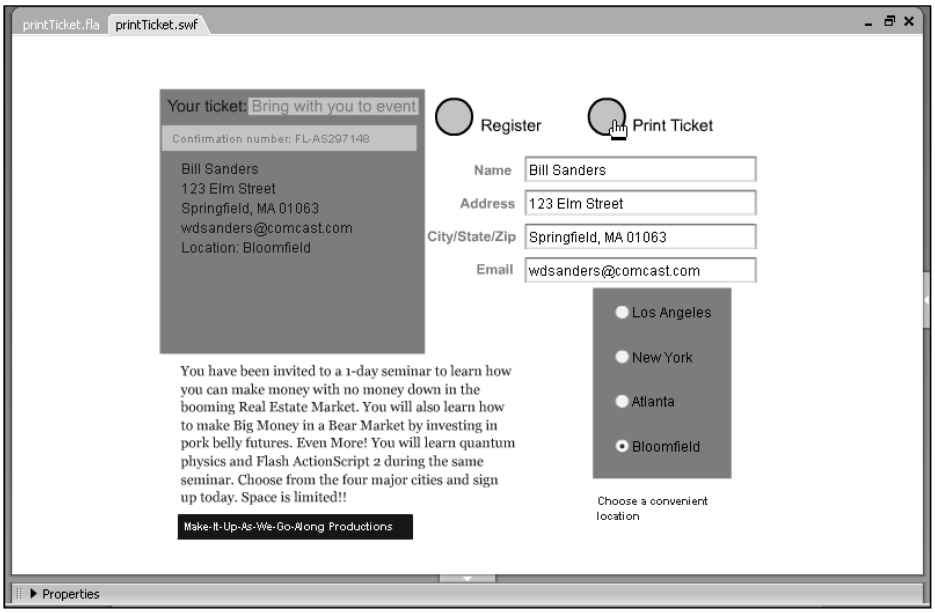

Figure 1.13 Text in a movie clip to be printed.

MovieClipLoader

The new MovieClipLoader class provides a new way of loading and "listening to" a movie being loaded. The MovieClipLoader is instantiated exactly like any other class using the `new` statement. As is the case with several of the new classes, a listener callback is used in conjunction with the class and related methods and properties. Table 1.8 shows the methods, properties, and listener callback functions used with the MovieClipLoader class.

Table 1.8 **MovieClipLoader Methods, Properties, and Listener Callback Functions**

Method	Description
`loadClip(url,target)`	Begins the process of loading an SWF file into the movie clip target.
`unloadClip(target)`	Terminates a current download.
`getProgress(target)`	Retrieves the current number of bytes loaded and total number of bytes.
Properties	**Description**
`bytesLoaded`	Returned object property of `getProgress` method of current number of bytes loaded.
`bytesTotal`	Returned object property of `getProgress` method of current number of bytes loaded.
Listener Callback Functions	**Description**
`onLoadStart(target)`	Called when download begins.
`onLoadProgress(target,BL,BT)`	Implemented as an alternative to `getProgress` method and can be used as single function for tracking bytes loaded (`BL`) and progress meters needing the total number of bytes (`BT`).
`onLoadComplete(target)`	Called when the download is finished.
`onLoadInit(target)`	Called when the download is complete *and* the actions in the first frame are complete.
`onLoadError(target, errorCode)`	Called when download fails or is terminated by unloadClip method.

To see how the MovieClipLoader class is instantiated and used to load an external movie into an existing movie clip, this next movie loads three different movies. As each movie is loaded, the listeners display the beginning and end of the load and the name of the loaded movie (see Listing 1.9).

Listing 1.9 **Loading a Movie Clip**

```
//Instantiate the MovieClipLoader
var mcLoader:MovieClipLoader = new MovieClipLoader();
var loadListner:Object = new Object();
//Listen for Start
loadListner.onLoadStart = function(movieClip) {
     _level0.loadInfo.text = "Begin";
};
loadListner.onLoadComplete = function(movieClip) {
     _level0.loadInfo.text += "--Complete " + launch;
};
mcLoader.addListener(loadListner);
```

continues

Listing 1.9 **Continued**

```
//Establish Functions for ComboBox
alpha = function () {
     mcLoader.loadClip("alpha.swf", loader_mc);
};
beta = function () {
     mcLoader.loadClip("beta.swf", loader_mc);
};
gamma = function () {
     mcLoader.loadClip("gamma.swf", loader_mc);
};
//Set up the ComboBox
var comboListener:Object = new Object();
chooseLoad_cb.addItem("Select One");
chooseLoad_cb.addItem("Alpha", "alpha");
chooseLoad_cb.addItem("Beta", "beta");
chooseLoad_cb.addItem("Gamma", "gamma");
comboListener.change = function(eventObj) {
     launch = chooseLoad_cb.selectedItem.data;
     if (launch == "alpha") {
          alpha();
     } else if (launch == "beta") {
          beta();
     } else if (launch == "gamma") {
          gamma();
     }
};
chooseLoad_cb.addEventListener("change", comboListener);
```

Figures 1.14 and 1.15 show how the different movies are loaded using the combo box UI component.

Figures 1.14 and 1.15 Different movie clips loaded into the screen.

Exceptions and the Error Class

In the Exceptions folder in the Actions panel, you will find the terms that are used with the Error class in ActionScript 2.0. The statements in the Exceptions folder are used in conjunction with the Error class instances. When an error occurs, an Error object is thrown using a `throw` statement. A `catch` statement is used to handle the error, typically with a statement specifying the error. A `try` statement looks for the error condition, and within the `try` block, the `throw` statement is initiated if the error condition is met. The `try` and `catch` statements are bound with a `finally` statement; thus, an exception is composed of three blocks. The `catch` statement contains an error argument, typically `e` or `error`. Within the `try...catch...finally` sequence, the `finally` block always executes whether the error condition is met or not.

The Error object is instantiated not with a variable, but with a message argument using the format

```
throw new Error("Error message");
```

within a `try` block.

The Error class has a single method and two properties, as shown in Table 1.9.

Table 1.9 **Error Method and Properties**

Method	**Description**
`toString()`	Transforms the Error object into a string.
Properties	**Description**
`message`	The error message associated with the error.
`name`	Exception type or name of error.

The Error class can be used to report built-in errors, such as the failure of an external file to load, or you can create user errors. In Listing 1.10, a new user error is created that traps numeric values under 65. It employs a UI List component for obtaining the data that will or will not create the error conditions. Two dynamic text fields display the data value (`data_txt`) and the error message, plus the ever-present value generated in the finally block (`errorMsg_txt`).

Listing 1.10 **Catching Defined Errors**

```
_global.style.setStyle("color", 0xa699e6);
_global.style.setStyle("highlightColor", 0xa6a6a6);
_global.style.setStyle("shadowColor", 0xa6a6a6);
```

continues

Listing 1.10 **Catching Defined Errors**

```
_global.style.setStyle("fontWeight", "bold");
//List box labels and data
errList_lb.addItem("Name", "Clark Kent");
errList_lb.addItem("Address", "123 Elm");
errList_lb.addItem("Age", 64);
errList_lb.addItem("Weight", 160);
//Create listener object
errorListen = new Object();
errorListen.change = function(eventObject) {
     errorMsg_txt.text = "";
     data_txt.text = errList_lb.selectedItem.data;
     var tester = errList_lb.selectedItem.data;
     //Exception
     try {
          //This next line is the custom error condition
          if (tester<=64 && typeof (tester) != "string") {
               throw new Error("Under the limit");
          }
     } catch (error) {
          if (error.message == "Under the limit") {
               errorMsg_txt.text = "Must 65 or older.";
          }
     } finally {
          errorMsg_txt.text += "Ok";
     }
};
errList_lb.addEventListener("change", errorListen);
```

Figure 1.16 shows how the error message is displayed, along with the "Ok" from the `finally` block:

Figure 1.16 The error throw is displayed in a text field.

Other New Terms in ActionScript 2.0

In addition to the new classes, ActionScript 2.0 has some other new key terms, some of which are associated with other classes. Each is reviewed briefly

with examples and explanations. They are covered in the main dictionary but like the other new terms are given further attention here.

The following section examines the different classes with new properties, methods, or other features in no special order. They are discussed here because they are new in some fashion and constitute enhancements to existing classes.

Array Constants

Five new constants are associated with the Array class and are used in conjunction with `Array.sort()` and `Array.sortOn()` methods. The constants are all uppercase, and because ActionScript 2.0 is case-sensitive, this feature is important. The following provides a brief description of each.

- **CASEINSENSITIVE**. Ignores case sensitivity when sorting, so words beginning with lowercase letters are not all placed after words beginning with uppercase letters. For example, Listing 1.11 generates two sorts with different outcomes, depending on whether the CASEINSENSITIVE constant is employed:

Listing 1.11 **A Case Insensitive Sort**

```
var folks:Array = new Array();
folks.push("Jesse", "Bill", "Linda", "Jeff", "aYo")
trace("First: " + folks.sort())
var newOrder:Array = folks.sort(Array.CASEINSENSITIVE);
trace("Second: " + newOrder);
_lockroot property
```

The following output is generated:

```
First: Bill,Jeff,Jesse,Linda,aYo
Second: aYo,Bill,Jeff,Jesse,Linda
```

As can be seen, the second sort puts the names in correct alphabetical order because the case is ignored.

- **DESCENDING**. Basically, this is a backward sort. The highest value is placed first and the lowest last. You can use the constants with `Array.sortOn()` as well, as Listing 1.12 shows:

Listing 1.12 **Sorting in Descending Order**

```
var cowPokes:Array = new Array();
cowPokes.push({name:"Wild Bill Hitchcock", classify:"Lawman"});
cowPokes.push({name:"Billy the Kid", classify:"Outlaw"});
cowPokes.push({name:"Wyatt Earpp", classify:"Lawman"});
```

continues

Listing 1.12 **Continued**

```
cowPokes.push({name:"Jesse James", classify:"Outlaw"});
cowPokes.push({name:"Sundance Kid", classify:"Outlaw"});
var cowboy:Array = cowPokes.sortOn ("name",Array.DESCENDING);
for (x=0; x<cowboy.length; x++) {
trace(cowboy[x]["name"]);
}
```

The script generates the following output:

```
Wyatt Earpp
Wild Bill Hitchcock
Sundance Kid
Jesse James
Billy the Kid
```

Note that in using a constant with `Array.sortOn()` that the field name precedes the constant.

```
var cowboy = cowPokes.sortOn ("name",Array.DESCENDING);
```

- **NUMERIC.** The numeric sort is designed for sorting when numbers are being compared, but if numbers are not in the array, string comparisons are used.
- **RETURNINDEXEDARRAY.** This constant keeps the array in unsorted order but returns the index value of each element in the position it will be in when sorted. For example, the following script

```
var folks:Array = new Array();
folks.push("Jesse", "Bill", "Linda", "Jeff", "aYo")
trace(folks.sort(Array.RETURNINDEXEDARRAY));
```

would display the following in the Output panel:

```
1,3,0,2,4
```

using a case-sensitive sort.

Keeping in mind that "aYo" begins with a lowercase letter and will be last in a sorted list where all the other names begin with an uppercase letter, you can see how this index works. The following shows the sorted list of names with the original index value next to each name:

Bill [1]
Jeff [3]
Jesse [0]
Linda [2]
aYo [4]

- **UNIQUESORT.** Every element in the array must be unique or an error is thrown. For example, if you have an array with the elements Jones, Smith, Jones, Allen, Johnson, and use the UNIQUESORT constant, instead of getting a sorted output, a 0 is returned.

_lockroot

When using several levels created by having multiple movie clips, it can be easy to make a call to the root level, forgetting that the root may be several levels up. To make life easier, especially when loading SWF files into movie clips using MovieClipLoader class, use `_lockroot`. To keep the root level where you want it, you can use the new `_lockroot` property. For example, if the following were put into the first frame of a movie and then loaded into another movie, the root target would be the movie that was loaded, not the movie into which it is loaded:

```
this._lockroot = true;
if (x == undefined || x>80) {
         var x:Number = 0;
}
x++;
_root.cool_mc._alpha = x+20;
_root.cool_mc._rotation = x;
```

To test the use of `_lockroot`, create a movie with two frames with a movie clip with the instance name "cool_mc" that can be seen to rotate—a simple line is fine. Test it using the above script so that you can see the movie clip in the movie rotate and change alpha levels. Then load the movie into another movie using MovieClipLoader. You should see the same actions. Then comment out the first line as shown:

```
//this._lockroot = true;
```

Now try it again. Without the `_lockroot` statement, neither the change in rotation nor the alpha level operate correctly when the movie is loaded into another movie clip.

MouseWheel

The addition of a mouse wheel event handler provides a mechanism to build movies that recognize and use the input from a mouse wheel. The `onMouseWheel` event handler recognizes changes in the mouse wheel position. The amount and direction (positive or negative) of the mouse wheel value can

be passed to a function and captured in the function argument. The following script shows how a movie clip object on the stage can be moved left and right using the script:

```
wheelMover = new Object();
wheelMover.onMouseWheel = function(mover) {
        bar_mc._x += mover;
};
Mouse.addListener(wheelMover);
```

In addition to the event handler, the TextField class includes a new property, `mouseWheelEnabled`. The line

```
someTextField.mouseWheelEnabled=true;
```

will enable the scroll bar to move when the mouse wheel is turned.

Movie Clip Depths

Two new movie clip methods are `getNextHighestDepth()` and `getInstanceAtDepth()`. The former method is employed to find what the next highest depth is for a loading movie clip. If no movie clip has been loaded, the next depth is 0. As each new movie clip is loaded, the depth increases by 1. The latter method, expressed as `movieClipName.getInstanceAtDepth()`, finds the name of the movie clip loaded at the specified depth. To see how the methods work, Listing 1.13 has a single movie clip on the stage with the instance name pinkie_mc and movie clips in the Library panel with the linkage names grayguy and greenie.

Listing 1.13 **Dynamically Changing Movie Clip Depths**

```
depth1_btn.onPress = function() {
     var someDepth = pinkie_mc.getNextHighestDepth();
     pinkie_mc.attachMovie("grayguy", "grayguyUp", someDepth);
     first_txt.text = pinkie_mc.getInstanceAtDepth(someDepth);
     trace(someDepth);
};
depth2_btn.onPress = function() {
     var someDepth = pinkie_mc.getNextHighestDepth();
     pinkie_mc.attachMovie("greenie", "greenieUp", someDepth)._x = -20;
     second_txt.text = pinkie_mc.getInstanceAtDepth(someDepth);
     trace(someDepth);
};
```

Figure 1.17 shows the results of pressing the top and bottom buttons several times. Each time the buttons are pressed, the depth increases by one, and the two movie clips loaded from the Library take turns overlapping each other.

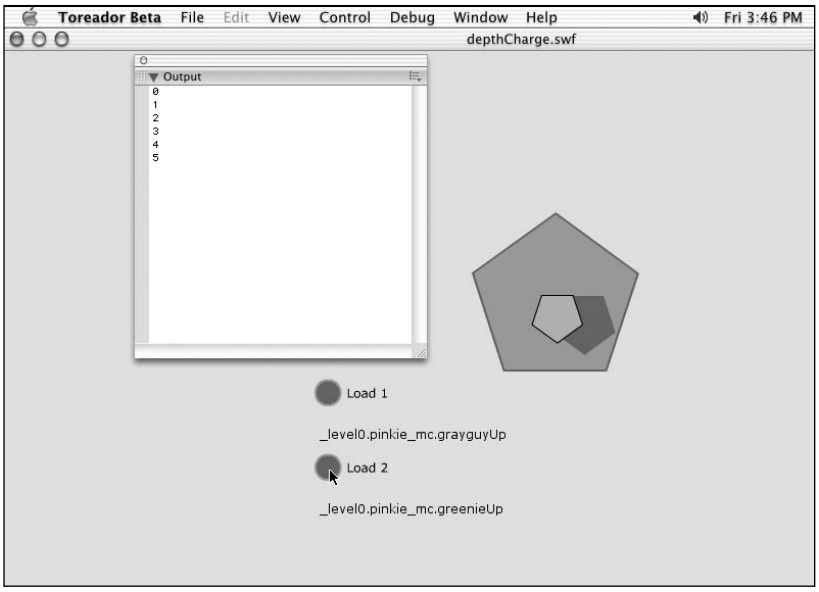

Figure 1.17 The Output panel shows the increasing depths each time the functions are fired.

Getting the Player Version

Another new movie clip method is `getSWFVersion`. It returns an integer with the player version (for example, 5, 6, or 7) the SWF was published as. For instance,

```
trace(_root.getSWFVersion());
```

shows the version of the SWF file in the Output panel.

Added System Capabilities Properties

Flash MX introduced `System.capabilities` to the ActionScript lexicon, and the new properties in ActionScript 2.0 primarily add to those capabilities. Like the original, many return a Boolean value. In Listing 1.14, six of the nine properties displayed have Boolean values.

Listing 1.14 **System Capabilities Displayed**

```
trace("Printing: "+System.capabilities.hasPrinting);
trace("Screen playback: "+System.capabilities.hasScreenPlayback);
```

continues

Listing 1.14 **Continued**

```
trace("Streaming audio: "+System.capabilities.hasStreamingAudio);
trace("Screen Broadcast: "+System.capabilities.hasScreenBroadcast);
trace("Streaming video: "+System.capabilities.hasStreamingVideo);
trace("Debugger: "+System.capabilities.isDebugger);
trace("Player type: "+System.capabilities.playerType);
trace("Server string: "+System.capabilities.serverString);
trace("Version: "+System.capabilities.version);
```

In addition to the Boolean values, `playerType` and `version` return fairly terse values. However, the new `serverString` property specifies values for each of the capabilities in URL-encoded format. Figure 1.18 shows all the output for the above script:

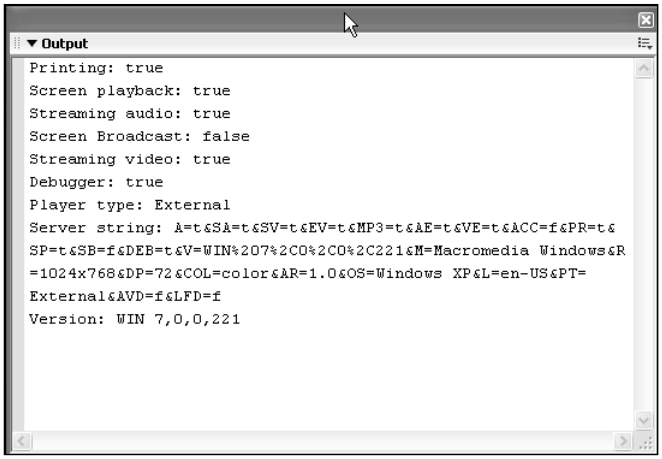

Figure 1.18 Most of the new `System.capabilities` properties are Boolean values.

Another new property added to the System class is `useCodePage`. This property is assigned a Boolean value to determine whether the system uses Unicode or the operating system's code page. The default setting is *not* to use Unicode, and so if you want to use Unicode, use the following script:

```
System.useCodepage=true;
```

The property is treated separately because it is not part of the properties belonging to the `System.capabilities`.

New System Methods

Two new System methods have been introduced in ActionScript 2.0. First, `System.security.allowDomain()` provides a way to override the default security system that disallows inter-domain scripting. For example, if your domain is `www.domainAlpha.com` and you have an SWF file in `www.domainBeta.com`, you can allow access so that you can load the movie in `www.domainBeta.com` using the line

```
System.security.allowDomain("domainBeta.com")
```

You can also use the formats "`http://domain`" or "`http://ip address`."

A second new System method, the `showSettings()` method, allows access to the different Macromedia Flash Player Settings for privacy, local storage, microphone, and camera. You can set your script to call one of the specific settings, as illustrated in Listing 1.15:

Listing 1.15 **Viewing System Settings**

```
//Privacy=0
priv_btn.onPress = function() {
     System.showSettings(0);
};
//Local Storage=1
store_btn.onPress = function() {
     System.showSettings(1);
};
//Microphone=2
mic_btn.onPress = function() {
     System.showSettings(2);
};
//Camera=3
cam_btn.onPress = function() {
      System.showSettings(3);
};
```

Each of the different settings is addressed by a value from 0 to 3 as indicated in the script. Figure 1.19 shows the setting shown when the Camera button is pressed.

Figure 1.19 Each of the four different Macromedia Flash Player Settings can be displayed with the `showSettings()` method.

Adding and Changing HTTP Request Headers

Both the LoadVars and XML classes have a new method, `addRequestHeader()`. Using an instance of either class, you can add or change HTTP request headers. The method accepts two arguments: a header name and a header value associated with the header name. For example,

```
var Loadie_0 = new LoadVars();
Loadie_0.addRequestHeader("Content-Type", "'Special'");
```

would specify "Content-Type" in the HTTP header with the value "Special." The XML class uses identical formatting with the LoadVars class.

All the new terms discussed in this section apply to existing classes. In the next section, ActionScript 2.0 has been applied to UI components as well; however, you will see that ActionScript 2.0 usage with UI components is very different from Flash MX.

UI Components and ActionScript 2.0

ActionScript 2.0 has several new terms and changes in addressing UI components. Also, several new UI component classes have their own ActionScript 2.0 methods, properties, and event handlers, and many of the old UI components have changed the ActionScript associated with their class as well. Rather than

attempting to examine all the terms associated with all the UI components, this section will focus on key new ActionScript 2.0 ways of dealing with UI components.

Styling UI Components

One of the major changes in ActionScript 2.0 in dealing with components is how they are styled, especially on a global level. Each of the UI components can be considered a class, and so global styles are assigned to the class. You can now assign a class style sheet for any class of component. Using the format

```
var comStyleSheetName = _global.styles.UIComponent=new
mx.styles.CSSStyleDeclaration();
```

you set the style sheet for the class. Table 1.10 shows the UI component class names that you can assign.

Table 1.10 **Component Coding Names**

Alert	Button	CheckBox	ComboBox	DataGrid
Label	List	Menu	NumericStepper	ProgressBar
RadioButton	ScrollPane	TextArea	TextInput	Tree
Window				

Listing 1.16 shows examples of scripting the Button, CheckBox, and ComboBox classes:

Listing 1.16 **CSS Styling Buttons**

```
//Button Styles
var btnStyle = _global.styles.Button=new mx.styles.CSSStyleDeclaration();
btnStyle.color = 0xBF3F00;
btnStyle.fontFamily = "Verdana";
btnStyle.fontWeight = "bold";
//Check Box Styles
var ckBxStyle = _global.styles.CheckBox=new mx.styles.CSSStyleDeclaration();
ckBxStyle.color = 0xA8C2B5;
ckBxStyle.fontFamily = "Georgia";
ckBxStyle.fontWeight = "bold";
//Combo Box Styles
comBx_cb.addItem("Alpha");
comBx_cb.addItem("Beta");
comBx_cb.addItem("Gamma");
var comBxStyle = _global.styles.ComboBox=new mx.styles.CSSStyleDeclaration();
comBxStyle.color = 0xBF3F00;
```

continues

Listing 1.16 **Continued**

```
comBxStyle.backgroundColor = 0xA8C2B5;
comBxStyle.background = 0xD4BBA1;
comBxStyle.fontFamily = "Courier";
comBxStyle.fontWeight = "bold";
```

Individual Style Settings

The ActionScript 2.0 for individual component styling is little changed from ActionScript 1.0. However, you will find a few changes. ActionScript 1.0 used the method

```
componentInName.setStyleProperty(styleProp, value);
```

However, ActionScript 2.0 uses

```
componentInName.setStyle(styleProp, value);
```

The difference is small, but small differences are the kinds that cause the most trouble because they are so difficult to detect. The following script shows how color and background color in a TextArea UI component are changed.

```
write_txt.text = "Nothing is perfect, and so nothing can be perfectly imperfect.";
write_txt.setStyle("color", 0xdddd00);
write_txt.setStyle("backgroundColor", 0xaa0000);
```

As can be seen, the difference is quite small. In the above example, the text colored is that assigned to the instance of the TextArea class, whereas in most other UI components the colored text is its label.

Styles Supported in ActionScript 2.0

The UI component style properties in ActionScript 2.0 have followed the general CSS style format adopted in other ActionScript 2.0 style properties. For example, instead of using the following line to set text color

```
light_btn.setStyleProperty("textColor", 0xdd00dd);
```

as is done in ActionScript 1.0, the line

```
light_btn.setStyle ("color", 0xdd00dd);
```

uses the `color` property, a term familiar to CSS users. Table 1.11 shows the complete set of UI component styles supported in ActionScript 2.0.

Table 1.11 **ActionScript 2.0 Component Style Names**

`backgroundColor`	component's background color
`borderColor`	3-D border's black section or 2-D section's color
`borderStyle`	*none, inset, outset*, or *solid* are recognized styles
`buttonColor`	3-D section and button face color
`color`	text color
`disabledColor`	color when text disabled
`fontFamily`	font family name, in quotes (for example, "Verdana," "Georgia")
`fontSize`	numeric value only for font size—no unit of measure (for example, pt or px)
`fontStyle`	*italic* or *normal* are recognized styles
`fontWeight`	*bold* or *normal* are recognized styles
`highlightColor`	three-dimensional border section
`marginLeft`	a numeric value only for left margin—no unit of measure (for example, pt or px)
`marginRight`	a numeric value only for right margin—no unit of measure (for example, pt or px)
`scrollTrackColor`	track color in a scrollbar
`shadowColor`	3-D border section
`symbolBackgroundColor`	radio button or check box background color
`symbolBackgroundDisabledColor`	radio button or check box background color when disabled
`symbolBackgroundPressedColor`	radio button or check box background color when pressed
`symbolColor`	color of radio button dot or check box check
`symbolDisabledColor`	disabled color of radio button or check box
`textAlign`	*left, right,* or *center* are acceptable values
`textDecoration`	*none* or *underline* are acceptable values
`textIndent`	a numeric value only for text indent—no unit of measure (for example, em, pt or px)

Making the Transition

The two primary areas of significant change between ActionScript 1.0 and ActionScript 2.0 can be found in the use of classes and CSS-like style terminology. This shift reflects the maturation of ActionScript into an ECMAScript 4-compliant language. As an object-oriented language, ActionScript now has the framework for building structures using encapsulated code, inheritance, and polymorphism.

This newest version of Flash, besides introducing ActionScript 2.0, can be used in interaction with other languages, primarily JavaScript, to create other structures that interact with ActionScript 2.0. See Article 4, "Creating Hybrid Flash Applications (HTML, JavaScript, and Flash)" by aYo Binitie on using Flash with multiple languages.

In this article, we have seen that although ActionScript 2.0 has different features and tools than ActionScript 1.0, the differences can be broken down into a few simple categories. First, we have the new tools like the Actions panel without the normal mode, the Integrated Script window (Flash Professional Only), and the Behaviors panel. Second, a few new classes have been added, and instead of using the `prototype` function, you can now create and reference classes in true object-oriented programming style. Third, some new features have been added to the existing classes you were familiar with from Flash MX. Finally, the UI components use ActionScirpt in their styling.

To help guide you through the transition, we have focused on the new features in ActionScript 2.0. And, for your convenience, all the scripts in this article are available online at `www.sandlight.com`; you can freely download them from there. I think you'll find that ActionScript 2.0 contains a number of welcomed changes to make your Flash creations even better.