Creating Custom Effects*

R.G. (Dick) Baldwin

Abstract

Learn how to create a custom effect and how to play it two different ways.

NOTE: **Click** CustomEffect02 [1] or CustomEffect03 [2] to run the ActionScript programs from this lesson. *(Click the "Back" button in your browser to return to this page.)*

1 Table of Contents

*Version 1.1: May 20, 2010 8:55 pm -0500
†http://creativecommons.org/licenses/by/3.0/
[1]http://cnx.org/content/m34457/latest/CustomEffect02.html
[2]http://cnx.org/content/m34457/latest/CustomEffect03.html

2 Preface

2.1 General

NOTE: All references to ActionScript in this lesson are references to version 3 or later.

This tutorial lesson is part of a series of lessons dedicated to object-oriented programming (OOP) with ActionScript.

Several ways to create and launch ActionScript programs

There are several ways to create and launch programs written in the ActionScript programming language. Many of the lessons in this series will use Adobe Flex as the launch pad for the sample ActionScript programs.

An earlier lesson titled **The Default Application Container** provided information on how to get started programming with Adobe's Flex Builder 3. *(See Baldwin's Flex programming website [3] .)* You should study that lesson before embarking on the lessons in this series.

Some understanding of Flex MXML will be required

I also recommend that you study all of the lessons on Baldwin's Flex programming website in parallel with your study of these ActionScript lessons. Eventually you will probably need to understand both Action-Script and Flex and the relationships that exist between them in order to become a successful ActionScript programmer.

Will emphasize ActionScript code

It is often possible to use either ActionScript code or Flex MXML code to achieve the same result. Insofar as this series of lessons is concerned, the emphasis will be on ActionScript code even in those cases where Flex MXML code may be a suitable alternative.

2.2 Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

2.2.1 Figures

2.2.2 Listings

[3] http://www.dickbaldwin.com/tocFlex.htm

2.3 Supplemental material

I recommend that you also study the other lessons in my extensive collection of online programming tutorials. You will find a consolidated index at www.DickBaldwin.com [4] .

3 General background information

In an earlier lesson titled **Events, Triggers, and Effects** , I taught you how to use the triggers and effects that are built into the ActionScript language.

In this lesson, I will teach you how to create your own custom effects. I will also explain two different programs that use a custom effect of my own design. I recommend that you run (p. 1) the online version of each of the two programs before continuing with the lesson.

4 Preview

Program output at startup

Figure 1 shows the screen output of both programs at startup.

[4] http://www.dickbaldwin.com/toc.htm

Program output at startup.

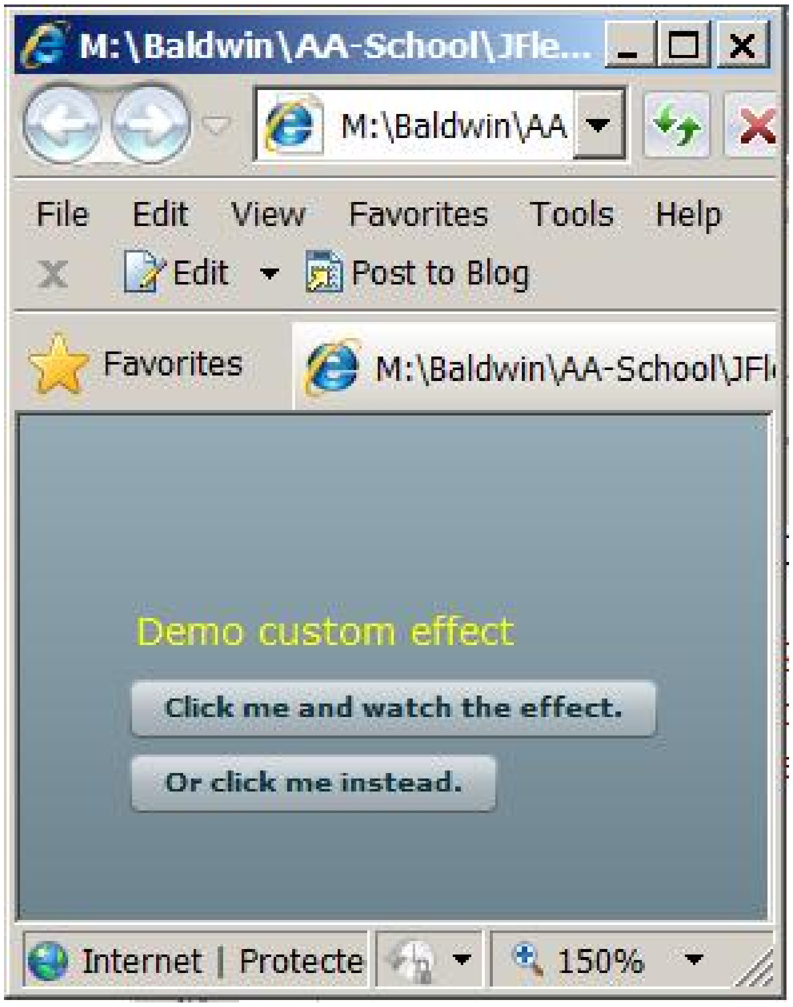

Figure 1: Program output at startup.

A common custom effect

Both programs apply the same custom effect to both buttons. However, the program named **CustomEffect02** applies the effect in such a way that it is played on both buttons simultaneously if either button is clicked. The program named **CustomEffect03** applies the custom effect in such a way that it plays individually on each button when the button is clicked. I will explain the reason for this difference later.

One run is worth a thousand pictures

Hopefully by now you have been able to run (p. 1) the online version of both programs because the effect is difficult to explain. Basically, the custom effect consists of the parallel execution of three types of standard ActionScript effects:

- WipeRight

- Glow
- Rotate

The effect appears to cause the buttons to leave their positions, change color, and fly around for a short while before settling back into their normal positions.

CustomEffect02 output after clicking a button

Figure 2 shows the output from the program named CustomEffect02 shortly after clicking one of the buttons.

CustomEffect02 output after clicking a button.

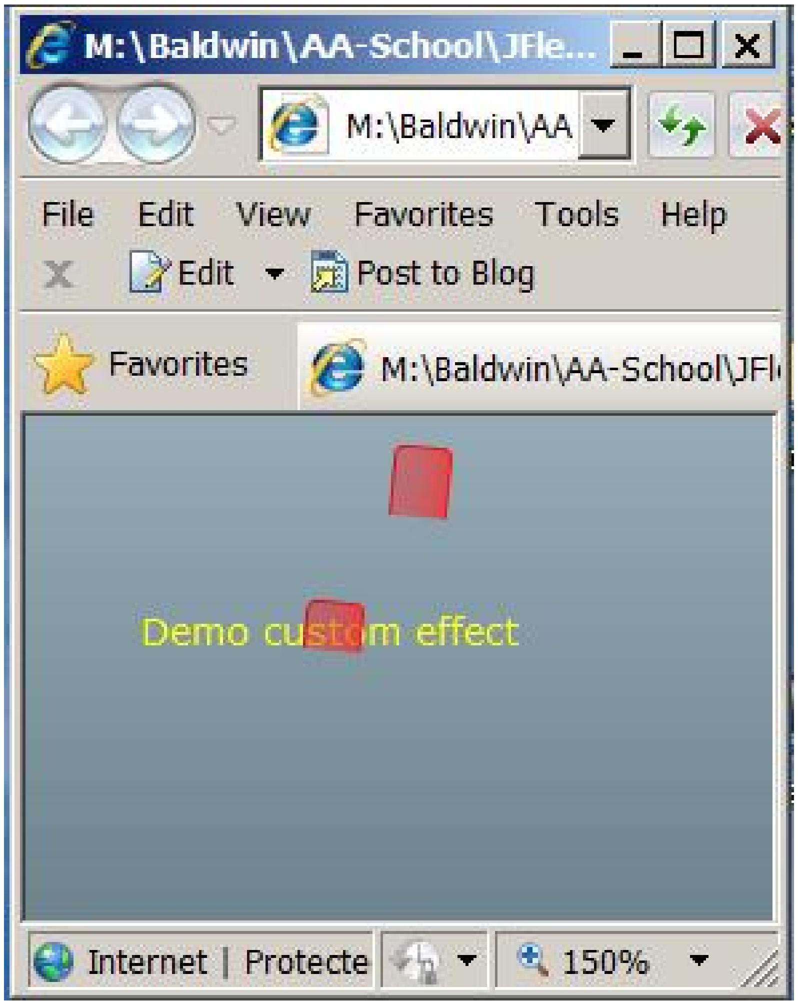

Figure 2: CustomEffect02 output after clicking a button.

---

5 Discussion and sample code

Will discuss in fragments

I will discuss and explain these two programs in fragments. Complete listings of the MXML code and the ActionScript code are provided near the end of the lesson beginning with Listing 18.

The MXML files

Both programs use the same simple MXML code shown in Listing 1.

Listing 1: Common MXML Code.

```
<?xml version="1.0" encoding="utf-8"?>

<!--CustomEffect02 11/26/09
Illustrates a custom effect, which is the parallel
playing of three standard effects:

WipeRight
Rotate
Glow

The effect is applied to two buttons each time either
button is clicked.

This version sets the targets on the effect and calls the
play method on the effect.

See the Flex 3 Cookbook, page 363
Also see http://livedocs.adobe.com/flex/3/html/help.html?
content=createeffects_2.html#178126
Also see http://livedocs.adobe.com/flex/3/html/help.html?
content=behaviors_04.html#275399
-->

<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:cc="CustomClasses.*">

  <cc:Driver/>

</mx:Application>
```

An object of the class named Driver

As you can see, this MXML file simply instantiates an object of the class named **Driver** . That's because almost all of the code in these two programs is written in ActionScript instead of MXML.

5.1 Creating a custom effect

You must define two classes to create a custom effect. One class is a *factory* class that extends the class named **Effect** . The other class is an *instance* class that extends the class named **EffectInstance** .

The instance class plays the effect

When the time comes to play the effect on a component, the factory class instantiates an object of the instance class to actually play the effect. If the same effect is played on two or more components at the same time, a different object of the instance class is instantiated to play the effect on each component.

Play three effects in parallel

As explained in Listing 1, the custom class that I designed for use in this lesson plays the following three effects in parallel:

- WipeRight
- Rotate
- Glow

You learned about something similar to this in my earlier lesson titled **Events, Triggers, and Effects** . However, in that lesson I didn't combine the three effects into a single custom effect the way that I will in this lesson.

Knowledge of OOP is required

I will do the best that I can to explain this code. Even at that, you are likely to need a pretty good understanding of object-oriented programming to understand the code required to create a custom effect. As you will see later, the required code is steeped in overridden methods, interfaces, and other object-oriented concepts.

5.2 The class named CustomEffect

The class named **CustomEffect** begins in Listing 2. A complete listing of the class is provided in Listing 19 near the end of the lesson.

Listing 2: Beginning of the class named CustomEffect.

```
  package CustomClasses{
import mx.effects.Effect;
import mx.effects.IEffectInstance;
import mx.events.EffectEvent;

public class CustomEffect extends Effect{

  //Would prefer to make these private and use implicit
  // setter methods, but I decided to leave them public
  // to simplify the code.
  public var theDuration:Number = 2000;//default value
  public var rotateAngleFrom:Number = 0;//default value
  public var rotateAngleTo:Number = 360;//default value
  public var wipeShowTarget:Boolean = true;//default
  public var glowColor:uint = 0xFF0000;//default value
  public var glowInner:Boolean = true;//default value
  public var glowStrength:Number = 255;//default value
```

The factory class

Of the two required classes, this is the factory class that I mentioned earlier. This class must extend the class named **Effect** , and will override methods inherited from that class.

Public instance variables

Listing 2 declares and initializes seven public instance variables that will be used to set properties on the **WipeRight** object, the **Rotate** object, and the **Glow** object. I provided default values for these variables so that the program will work even if the driver program fails to provide the required values.

Could use implicit setter methods

As I mentioned in the comments, I would prefer to make these variables private and provide an implicit setter method for each variable. However, I decided to make them public to simplify the code and make it easier to explain.

The constructor for the class named CustomEffect

The constructor is shown in its entirety in Listing 3.

Listing 3: The constructor for the class named CustomEffect.

```
    public function CustomEffect(target:Object=null){
  super(target);
  instanceClass = CustomEffectInstance;
} //end constructor
```

The incoming parameter

The incoming parameter for the constructor is the generic type **Object** . This parameter must specify the component on which the effect is to be played.

If no target is passed as a parameter to the constructor, the default null value prevails and the **target** property of the object must be set. As you will see later, an alternative property named **targets** can be set to cause the effect to be played on multiple targets at the same time.

Call the superclass constructor

Without attempting to explain why, I am going to tell you that it is frequently necessary in OOP to cause the constructor for a class to make a call to the constructor of its superclass as the first statement in the constructor. This constructor is no exception to that rule.

The first statement in Listing 3 is a call to the constructor for the **Effect** class passing a reference to the target component(s) as a parameter. When that constructor returns control, the second statement in Listing 3 is executed.

The instanceClass property

This class inherits a property named **instanceClass** from the class named **Effect** . According to About creating a custom effect [5] , the factory class that you define must set the value of this property to the name of the instance class that will be used to play the effect.

In this program, the name of the instance class is **CustomEffectInstance** , as shown in Listing 3. I will explain the code in that class after I finish explaining the code in this class.

This inherited property provides the mechanism that ties the instance class to the factory class.

Override the initInstance method

Listing 4 overrides an inherited method named **initInstance** .

Listing 4: Override the initInstance method.

```
    override protected function initInstance(
                         instance:IEffectInstance):void{
  super.initInstance(instance);

  CustomEffectInstance(instance).theDuration =
                                    this.theDuration;
  CustomEffectInstance(instance).rotateAngleFrom =
                                this.rotateAngleFrom;
  CustomEffectInstance(instance).rotateAngleTo =
                                  this.rotateAngleTo;
```

[5] http://livedocs.adobe.com/flex/3/html/help.html?content=createeffects_2.html#178126

```
      CustomEffectInstance(instance).wipeShowTarget =
                                this.wipeShowTarget;
      CustomEffectInstance(instance).glowColor =
                                     this.glowColor;
      CustomEffectInstance(instance).glowInner =
                                     this.glowInner;
      CustomEffectInstance(instance).glowStrength =
                                  this.glowStrength;

    } //end initInstance
```

Set the property values in the instance object

According to About creating a custom effect[6] , the purpose of this method is to copy property values from the factory class to the instance class. Flex calls this method from the **Effect.createInstance()** method. You don't have to call it yourself, but you must prepare it to be called.

A reference to the instance object as type iEffectInstance

The **initInstance** method receives a reference to the instance object as the interface type **IEffectInstance** . The objective is to write values into the properties belonging to the instance object. However, the **IEffectInstance** interface doesn't know anything about properties having those names. Therefore, it is necessary to cast the instance object's reference to the type of the instance object before making each assignment. One such cast operation is shown by the statement that begins with **CustomEffectInstance(instance)** in Listing 4.

Call the initInstance method of the superclass

Also note that you must call the **initInstance** method of the superclass in your overridden method as shown in Listing 4.

This method provides the mechanism by which required property values make it all the way from the driver class to the instance class.

Override the getAffectedProperties method

According to About creating a custom effect[7] , you must override the inherited method named **getAffectedProperties** in such a way as to return an array of strings. Each string is the name of a property of the target object that is changed by the effect. If no properties are changed, you must return an empty array.

Listing 5 shows my overridden version of the **getAffectedProperties** method.

Listing 5: Override the getAffectedProperties method.

```
        override public function
                          getAffectedProperties():Array{
      return ["rotation","rotationX","rotationY","x","y"];
    } //end getAffectedProperties
    //--------------------------------------------------//
  } //end class
} //end package
```

This is a little difficult

It is a little difficult to know exactly which properties belonging to the target component will be modified by the effect, particularly when the custom effect is a composite of existing effects. Also, I don't know whether a change must be permanent or whether a temporary change in the value of a property requires

[6]http://livedocs.adobe.com/flex/3/html/help.html?content=createeffects_2.html#178126
[7]http://livedocs.adobe.com/flex/3/html/help.html?content=createeffects_2.html#178126

that it be returned by the **getAffectedProperties** method. There are several target property values that are temporarily changed by this custom effect.

In this program, the target component is a **Button** object but it could be any component. I went through the list of properties belonging to a button and came up with the five shown in Listing 5 as those most likely to be modified.

The end of the CustomEffect class

Listing 5 also signals the end of the class named **CustomEffect** . In addition to the methods that were overridden above, the following two inherited methods may optionally be overridden as well:

- **effectStartHandler** - called when the effect instance starts playing.
- **effectEndHandler** - called when the effect instance finishes playing.

As the names and descriptions of these two methods suggest, they can be overridden to provide any special behavior that you need when the effect starts and finishes playing.

5.3 The class named CustomEffectInstance

The class named **CustomEffectInstance** begins in Listing 6. A complete listing of the class is provided in Listing 20 near the end of the lesson.

Listing 6: Beginning of the CustomEffectInstance class.

```
  package CustomClasses{
import mx.effects.EffectInstance;
import mx.effects.Glow;
import mx.effects.Parallel;
import mx.effects.Rotate;
import mx.effects.WipeRight;
import mx.events.FlexEvent;

public class CustomEffectInstance
                                 extends EffectInstance{
  //Instantiate the individual effects that will be
  // combined in parallel to produce the custom effect.
  private var wipeEffect:WipeRight = new WipeRight();
  private var rotateEffect:Rotate = new Rotate();
  private var glowEffect:Glow = new Glow();
```

Extends the class named EffectInstance

This class extends the class named **EffectInstance** . As before, the code in this class will override methods inherited from the **EffectInstance** class.

Instantiate and save references to standard effects

Listing 6 instantiates and saves references to the **WipeRight** , **Rotate** , and **Glow** effect classes. They will be combined to run concurrently later in the program.

Declare variables for storage of effect properties

Listing 7 declares a set of variables that will be used to store the properties for the three different effects that are combined to create a composite effect.

Listing 7: Declare variables for storage of effect properties.

```
        //Variables for the storage of effect properties.
    public var theDuration:Number;
    public var rotateAngleFrom:Number;
    public var rotateAngleTo:Number;
    public var wipeShowTarget:Boolean;
    public var glowColor:uint;
    public var glowInner:Boolean;
    public var glowStrength:Number;
```

Property values are stored in these variables by the code in the **initInstance** method shown in Listing 4.

The constructor for the CustomEffectInstance class

The constructor for the class is shown in its entirety in Listing 8.

Listing 8: The constructor for the CustomEffectInstance class.

```
        public function CustomEffectInstance(
                                        theTarget:Object){
      super(theTarget);

      //Set the target for all three individual effects.
      rotateEffect.target = theTarget;
      wipeEffect.target = theTarget;
      glowEffect.target = theTarget;
    } //end constructor
```

The target component

As was the case in Listing 3, this constructor receives a parameter of the generic type **Object** , which specifies the component on which the effect will be played.

A different EffectInstance object for each target component

If multiple target components are specified by setting the **targets** property of the **CustomEffect** class, different objects of the **CustomEffectInstance** class are instantiated and targeted to the different components in the list of target components.

Target the WipeRight, Rotate, and Glow effects to the target component

The code in the constructor sets the specified target to be the target for the three types of standard effects that will be played concurrently.

Override the inherited play method

You may have noticed that the code in the **CustomEffect** class didn't include any of the operational details regarding the nature of the custom effect. Those details are programmed into an overridden **play** method that begins in Listing 9.

Listing 9: Override the inherited play method.

```
        override public function play():void{
      super.play();

      //Note: The following values cannot be set in the
      // constructor because the variables aren't stable
      // at that point in time.

      //Configure the rotate effect
      rotateEffect.angleFrom = rotateAngleFrom;
      rotateEffect.angleTo = rotateAngleTo;
      rotateEffect.duration = theDuration;

      //Configure the wipe effect.
      wipeEffect.showTarget = wipeShowTarget;
      wipeEffect.duration = theDuration;

      //Configure the glow effect.
      glowEffect.color = glowColor;
      glowEffect.duration = theDuration;
      glowEffect.inner = glowInner;
      glowEffect.strength = glowStrength;
```

The overridden play method produces the desired effect

Later on you will see that the driver class for this program instantiates an object of the custom effect class and calls the **play** method on that object. At that point, the driver class will be calling the method that begins in Listing 9.

Set the required properties on the three standard effects

Listing 9 uses the values that were stored in the variables in Listing 7 by the initInstance method in Listing 4 to set the required properties for each of the three individual effects that will be combined to produce this custom effect.

Ordinarily, you might think that this could have been accomplished in the constructor for the class. However, the values in the variables in Listing 7 aren't stable until the constructor has finished constructing the object. Therefore, it is necessary to defer the assignments in Listing 9 until after the construction of the object is complete.

Play the three effects in parallel

You learned how to use an object of the **Parallel** class to play two or more effects in parallel in the earlier lesson titled **Events, Triggers, and Effects** .

Listing 10: Play the three effects in parallel.

```
          //Play all three effects in parallel.
      var parallel:Parallel = new Parallel();
      parallel.addChild(rotateEffect);
      parallel.addChild(glowEffect);
      parallel.addChild(wipeEffect);
      parallel.play();
    } //end play
    //--------------------------------------------------//
  } //end class
} //end package
```

Therefore, you shouldn't have any difficulty understanding the code in Listing 10.

Steps required to create a custom effect

You must define a factory class and an instance class. The following steps are required to create and prepare the factory class:

- Define a factory class that extends the **Effect** class.
- Declare variables in the factory class, if any are required, to store the property values for the custom effect.
- Define a constructor for the factory class that calls the constructor for the superclass and also sets the name of the instance class into the inherited variable named **instanceClass** .

- Override the **initInstance** method in the factory class to store the property values into variables in the instance class. Also call the **initInstance** method of the superclass in that overridden method.
- Override the **getAffectedProperties** method in the factory class to return a list of target component properties that will be modified by the effect. Return an empty array if none will be modified.

Define and prepare the instance class

Having defined the factory class using the steps listed above, define the instance class by performing the following steps:

- Define an instance class that extends the **EffectInstance** class.
- Declare public instance variables for the storage of effect property values if any are required.
- Define a constructor for the instance class that deals appropriately with the target component.
- Override the inherited **play** method to implement the actual behavior of the custom effect.

The end of the CustomEffectInstance class

Listing 10 signals the end of the class named **CustomEffectInstance** .

5.4 The Driver class for the program named CustomEffect02

The two classes discussed above constitute the whole of the custom effect. I will provide and explain two different driver classes that use the same custom effect but use it in different ways. The driver class for the program named **CustomEffect02** begins in Listing 11. A complete listing of this class is provided in Listing 21 near the end of the lesson.

Two ways to play effects

You learned in the earlier lesson titled **Events, Triggers, and Effects** that there are at least two different ways to cause an effect to be played on a component in an ActionScript program. One way is to call the **setStyle** method on the component and associate an effect trigger with an effect. With that approach, the effect will be played each time the effect trigger fires.

The second way

The second way to play an effect on a component is to target an **Effect** object to the component and then call the **play** method on the effect object. This approach doesn't make explicit use of the effect trigger.

I will illustrate the second approach in the program named **CustomEffect02** , and will illustrate the first approach later in the program named **CustomEffect03** .

Beginning of the Driver class for CustomEffect02

The Driver class begins in Listing 11.

Listing 11: Beginning of the Driver class for CustomEffect02.

```
  package CustomClasses{

import mx.containers.VBox;
import mx.controls.Button;
import mx.controls.Label;
import mx.controls.Spacer;
import flash.events.MouseEvent;

public class Driver extends VBox{
  //Instantiate and save references to all of the
  // objects needed by the program.
  private var title:Label = new Label();
  private var btnA:Button = new Button();
```

```
    private var btnB:Button = new Button();
    private var spacer:Spacer = new Spacer();
    private var theEffect:CustomEffect =
                                   new CustomEffect();
```

The code in Listing 11 extends the **VBox** class and instantiates objects for all of the components that will be required to produce the GUI shown in Figure 1. In addition, Listing 11 instantiates an object of the new **CustomEffect** class.

No target is passed to the constructor

As you can see from Listing 11, a target component was not passed to the constructor for the **CustomEffect** class. Instead, an alternative approach that sets the **targets** property will be used.

Beginning of the constructor for the Driver class

The constructor for the Driver class begins in Listing 12.

Listing 12: Beginning of the constructor for the Driver class.

```
      public function Driver(){//constructor
    //Make some space at the top of the display.
    spacer.height = 40;
    addChild(spacer);

    //Set title properties and add to the VBox.
    title.setStyle("color","0xFFFF00");
    title.setStyle("fontSize",14);
    title.text = "Demo custom effect";
    addChild(title);

    //Instantiate two buttons and add them to the VBox.
    // Register the same event listener on both of
    // them.
    btnA.label = "Click me and watch the effect.";
    btnA.addEventListener(MouseEvent.CLICK,handler);
    addChild(btnA);

    btnB.label = "Or click me instead.";
    btnB.addEventListener(MouseEvent.CLICK,handler);
    addChild(btnB);
```

There is nothing new in Listing 12 so further explanation shouldn't be required. It is worth noting, however, that the same **click** event listener is registered on both buttons.

Set properties on the custom effect

Listing 13 shows the code that sets properties on the custom effect.

Listing 13: Set properties on the custom effect.

```
        //Specify both buttons to be the target for the
    // same effect.
    theEffect.targets = [btnA,btnB];

    //Set various properties needed by the effect.
    theEffect.theDuration = 4000;
```

```
      theEffect.rotateAngleFrom = 0;
      theEffect.rotateAngleTo = 720;
      theEffect.wipeShowTarget = true;
      theEffect.glowColor = 0xFF0000;
      theEffect.glowInner = true;
      theEffect.glowStrength = 255;

    } //end constructor
```

With the exception of the property named **targets** , the values that are assigned in Listing 13 are stored in the variables that are declared in Listing 2.

The targets property

The **targets** property is inherited into the **CustomEffect** class from the **Effect** class. Note that both buttons are passed to the **targets** property in the form of an array containing references to the two buttons. This causes the custom effect to be played on both buttons at the same time.

Listing 13 also signals the end of the constructor for the **Driver** class.

The common click event handler

The click event handler that is registered on both buttons is shown in Listing 14.

Listing 14: The common click event handler.

```
      public function handler(event:MouseEvent):void{
      theEffect.play();
    }//end handler

  } //end class
} //end package
```

The event handler calls the **play** method on the custom effect object whenever either of the buttons shown in Figure 1 is clicked. This causes the **play** method defined in Listing 9 to be executed.

The end of the program

Listing 14 also signals the end of the program named **CustomEffect02** .

5.5 The Driver class for the program named CustomEffect03

The **Driver** class for the program named **CustomEffect03** begins in Listing 15. A complete listing of the class is provided in Listing 22 near the end of the lesson. This program uses the first approach (p. 13) for playing an effect.

Listing 15: Beginning of the Driver class for CustomEffect03.

```
    package CustomClasses{

  import mx.containers.VBox;
  import mx.controls.Button;
  import mx.controls.Label;
  import mx.controls.Spacer;

  public class Driver extends VBox{
    //Instantiate and save references to all of the
    // objects needed by the program.
  private var title:Label = new Label();
  private var btnA:Button = new Button();
  private var btnB:Button = new Button();
  private var spacer:Spacer = new Spacer();
  private var theEffect:CustomEffect =
                                   new CustomEffect();
  //--------------------------------------------------//

  public function Driver(){//constructor
    //Make some space at the top of the display.
    spacer.height = 40;
    addChild(spacer);

    //Set title properties and add to the VBox.
    title.setStyle("color","0xFFFF00");
    title.setStyle("fontSize",14);
    title.text = "Demo custom effect";
    addChild(title);

    //Instantiate two buttons and add them to the VBox.
    // Register the same event listener on both of
    // them.
    btnA.label = "Click me and watch the effect.";
    addChild(btnA);

    btnB.label = "Or click me instead.";
    addChild(btnB);
```

Very similar to the previous code

The code in Listing 15 matches the code in Listing 11 and Listing 12 with a few exceptions:

- There is no import directive for the **MouseEvent** class.
- There are no **click** event handlers registered on the buttons.

Set properties on the custom effect

Listing 16 sets the properties on the custom effect.

Listing 16: Set properties on the custom effect.

```
        //Set various properties needed by the effect.
    theEffect.theDuration = 4000;
    theEffect.rotateAngleFrom = 0;
    theEffect.rotateAngleTo = 720;
    theEffect.wipeShowTarget = true;
    theEffect.glowColor = 0xFF0000;
    theEffect.glowInner = true;
    theEffect.glowStrength = 255;
```

Once again, this code is very similar to the code in Listing 13. There is one major difference, however. The **targets** property for the effect is not explicitly set to the buttons as is the case in Listing 13.

Apply the effect to the two buttons individually

Listing 17 shows the major difference between the two programs.

Listing 17: Apply the effect to the two buttons individually.

```
        btnA.setStyle("mouseUpEffect",theEffect);
      btnB.setStyle("mouseUpEffect",theEffect);

    } //end constructor
    //--------------------------------------------------//
  } //end class
} //end package
```

Use the setStyle method and the effect trigger

Whereas the previous program explicitly sets the buttons as targets of the effect and calls the **play** method on the effect, this program uses the **setStyle** approach and associates the custom effect with a **mouseUpEffect** trigger on each button individually. As a result, when the mouse button is released while the mouse pointer is over one of the buttons, the effect is played on that button alone.

May not be possible to specify multiple targets

I don't know of any easy way to use this approach to cause the effect to be played on two or more components at the same time. The documentation hints that this may not be possible.

The end of the program

Listing 17 also signals the end of the **Driver** class and the end of the program.

6 Run the program

I encourage you to run (p. 1) this program from the web. Then copy the code from Listing 18 through Listing 22. Use that code to create Flex projects. Compile and run the projects. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

7 Resources

I will publish a list containing links to ActionScript resources as a separate document. Search for ActionScript Resources in the Connexions search box.

8 Complete program listings

Complete listings of the Flex applications discussed in this lesson are provided below.

Listing 18: Common MXML code used for both programs.

```
    <?xml version="1.0" encoding="utf-8"?>

<!--CustomEffect02 11/26/09
Illustrates a custom effect, which is the parallel
playing of three standard effects:

WipeRight
Rotate
Glow
```

```
The effect is applied to two buttons each time either
button is clicked.

This version sets the targets on the effect and calls the
play method on the effect.

See the Flex 3 Cookbook, page 363
Also see http://livedocs.adobe.com/flex/3/html/help.html?
content=createeffects_2.html#178126
Also see http://livedocs.adobe.com/flex/3/html/help.html?
content=behaviors_04.html#275399
-->

<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:cc="CustomClasses.*">

  <cc:Driver/>

</mx:Application>
```

Listing 19: Source code for the class named CustomEffect.

```
    package CustomClasses{
  import mx.effects.Effect;
  import mx.effects.IEffectInstance;
  import mx.events.EffectEvent;

  public class CustomEffect extends Effect{

    //Would prefer to make these private and use implicit
    // setter methods, but I decided to leave them public
    // to simplify the code.
    public var theDuration:Number = 2000;//default value
    public var rotateAngleFrom:Number = 0;//default value
    public var rotateAngleTo:Number = 360;//default value
    public var wipeShowTarget:Boolean = true;//default
    public var glowColor:uint = 0xFF0000;//default value
    public var glowInner:Boolean = true;//default value
    public var glowStrength:Number = 255;//default value

    public function CustomEffect(target:Object=null){
      super(target);
      instanceClass = CustomEffectInstance;
    } //end constructor

    override protected function initInstance(
                          instance:IEffectInstance):void{
      super.initInstance(instance);
      CustomEffectInstance(instance).theDuration =
                                      this.theDuration;
      CustomEffectInstance(instance).rotateAngleFrom =
                                  this.rotateAngleFrom;
      CustomEffectInstance(instance).rotateAngleTo =
                                    this.rotateAngleTo;
      CustomEffectInstance(instance).wipeShowTarget =
                                   this.wipeShowTarget;
      CustomEffectInstance(instance).glowColor =
                                        this.glowColor;
      CustomEffectInstance(instance).glowInner =
                                        this.glowInner;
      CustomEffectInstance(instance).glowStrength =
                                     this.glowStrength;

    } //end initInstance
    //----------------------------------------------------//

    override public function
                        getAffectedProperties():Array{
      return ["rotation","rotationX","rotationY","x","y"];
    } //end getAffectedProperties
    //----------------------------------------------------//
  } //end class
} //end package
```

Listing 20: Source code for the class named CustomEffectInstance.

```
    package CustomClasses{
  import mx.effects.EffectInstance;
  import mx.effects.Glow;
  import mx.effects.Parallel;
  import mx.effects.Rotate;
  import mx.effects.WipeRight;
  import mx.events.FlexEvent;

  public class CustomEffectInstance
                                   extends EffectInstance{
    //Instantiate the individual effects that will be
    // combined in parallel to produce the custom effect.
    private var wipeEffect:WipeRight = new WipeRight();
    private var rotateEffect:Rotate = new Rotate();
    private var glowEffect:Glow = new Glow();

    //Variables for the storage of effect properties.
    public var theDuration:Number;
    public var rotateAngleFrom:Number;
    public var rotateAngleTo:Number;
    public var wipeShowTarget:Boolean;
    public var glowColor:uint;
    public var glowInner:Boolean;
    public var glowStrength:Number;

    public function CustomEffectInstance(
                                    theTarget:Object){
      super(theTarget);

      //Set the target for all three individual effects.
      rotateEffect.target = theTarget;
      wipeEffect.target = theTarget;
      glowEffect.target = theTarget;
    } //end constructor

    override public function play():void{
      super.play();

      //Note: The following values cannot be set in the
      // constructor because the variables aren't stable
      // at that point in time.

      //Configure the rotate effect
      rotateEffect.angleFrom = rotateAngleFrom;
      rotateEffect.angleTo = rotateAngleTo;
      rotateEffect.duration = theDuration;

      //Configure the wipe effect.
      wipeEffect.showTarget = wipeShowTarget;
      wipeEffect.duration = theDuration;

      //Configure the glow effect.
      glowEffect.color = glowColor;
      glowEffect.duration = theDuration;
      glowEffect.inner = glowInner;
      glowEffect.strength = glowStrength;

      //Play all three effects in parallel.
      var parallel:Parallel = new Parallel();
      parallel.addChild(rotateEffect);
      parallel.addChild(glowEffect);
      parallel.addChild(wipeEffect);
      parallel.play();
    } //end play
    //--------------------------------------------------//
  } //end class
} //end package
```

Listing 21: Driver class for the program named CustomEffect02.

```
    /*CustomEffect02 11/26/09
```

```
*********************************************************/

package CustomClasses{

  import mx.containers.VBox;
  import mx.controls.Button;
  import mx.controls.Label;
  import mx.controls.Spacer;
  import flash.events.MouseEvent;

  public class Driver extends VBox{
    //Instantiate and save references to all of the
    // objects needed by the program.
    private var title:Label = new Label();
    private var btnA:Button = new Button();
    private var btnB:Button = new Button();
    private var spacer:Spacer = new Spacer();
    private var theEffect:CustomEffect =
                                   new CustomEffect();
    //--------------------------------------------------//

    public function Driver(){//constructor
      //Make some space at the top of the display.
      spacer.height = 40;
      addChild(spacer);

      //Set title properties and add to the VBox.
      title.setStyle("color","0xFFFF00");
      title.setStyle("fontSize",14);
      title.text = "Demo custom effect";
      addChild(title);

      //Instantiate two buttons and add them to the VBox.
      // Register the same event listener on both of
      // them.
      btnA.label = "Click me and watch the effect.";
      btnA.addEventListener(MouseEvent.CLICK,handler);
      addChild(btnA);

      btnB.label = "Or click me instead.";
      btnB.addEventListener(MouseEvent.CLICK,handler);
      addChild(btnB);

      //Specify both buttons to be the target for the
      // same effect.
      theEffect.targets = [btnA,btnB];

      //Set various properties needed by the effect.
      theEffect.theDuration = 4000;
      theEffect.rotateAngleFrom = 0;
      theEffect.rotateAngleTo = 720;
      theEffect.wipeShowTarget = true;
      theEffect.glowColor = 0xFF0000;
      theEffect.glowInner = true;
      theEffect.glowStrength = 255;

    } //end constructor
    //--------------------------------------------------//

    public function handler(event:MouseEvent):void{
      theEffect.play();
    }//end handler

  } //end class
} //end package
```

Listing 22: Driver class for the program named CustomEffect03.

```
    /*CustomEffect03 11/27/09
*********************************************************/

package CustomClasses{

  import mx.containers.VBox;
  import mx.controls.Button;
  import mx.controls.Label;
  import mx.controls.Spacer;

  public class Driver extends VBox{
    //Instantiate and save references to all of the
    // objects needed by the program.
    private var title:Label = new Label();
    private var btnA:Button = new Button();
    private var btnB:Button = new Button();
    private var spacer:Spacer = new Spacer();
    private var theEffect:CustomEffect =
                                   new CustomEffect();
    //--------------------------------------------------//

    public function Driver(){//constructor
      //Make some space at the top of the display.
      spacer.height = 40;
      addChild(spacer);

      //Set title properties and add to the VBox.
      title.setStyle("color","0xFFFF00");
      title.setStyle("fontSize",14);
      title.text = "Demo custom effect";
      addChild(title);

      //Instantiate two buttons and add them to the VBox.
      // Register the same event listener on both of
      // them.
      btnA.label = "Click me and watch the effect.";
      addChild(btnA);

      btnB.label = "Or click me instead.";
      addChild(btnB);

      //Set various properties needed by the effect.
      theEffect.theDuration = 4000;
      theEffect.rotateAngleFrom = 0;
      theEffect.rotateAngleTo = 720;
      theEffect.wipeShowTarget = true;
      theEffect.glowColor = 0xFF0000;
      theEffect.glowInner = true;
      theEffect.glowStrength = 255;

      //Apply the effect to the two buttons individually.
      btnA.setStyle("mouseUpEffect",theEffect);
      btnB.setStyle("mouseUpEffect",theEffect);

    } //end constructor
    //--------------------------------------------------//
  } //end class
} //end package
```

9 Miscellaneous

This section contains a variety of miscellaneous materials.

NOTE: **Housekeeping material**

- Module name: Creating Custom Effects
- Files:
  - ActionScript0118\ActionScript0118.htm
  - ActionScript0118\Connexions\ActionScriptXhtml0118.htm

-end-