Handling Slider Change Events in Flex 3 and Flex 4^*

R.G. (Dick) Baldwin

This work is produced by OpenStax-CNX and licensed under the Creative Commons Attribution License 3.0^{\dagger}

Abstract

Learn how to write inline event handler code to handle slider change events.

NOTE: Click SliderChangeEvent01¹, SliderChangeEvent02², and SliderChangeEvent03³ to run the Flex programs from this lesson. (Click the "Back" button in your browser to return to this page.)

1 Table of Contents

- Preface (p. 2)
 - · General (p. 2)
 - \cdot Viewing tip (p. 2)
 - * Figures (p. 2)
 - * Listings (p. 2)
 - · Supplemental material (p. 2)
- General background information (p. 2)
- Preview (p. 3)
- Discussion and sample code (p. 9)
 - The Flex 3 project named SliderChangeEvent01 (p. 9)
 - The hybrid Flex3-4 project named SliderChangeEvent02 (p. 17)
 - The Flex 4 project named SliderChangeEvent03 (p. 23)
- Run the programs (p. 23)
- Resources (p. 23)
- Complete program listings (p. 23)
- Miscellaneous (p. 25)

^{*}Version 1.1: Jun 18, 2010 4:58 pm -0500

[†]http://creativecommons.org/licenses/by/3.0/

 $^{^{1}} http://cnx.org/content/m34633/latest/SliderChangeEvent01.html$

 $^{^{2}} http://cnx.org/content/m34633/latest/SliderChangeEvent02.html$

 $^{^{3}} http://cnx.org/content/m34633/latest/SliderChangeEvent03.html$

2 Preface

2.1 General

This lesson is part of a series of tutorial lessons dedicated to programming with Adobe Flex. The material in this lesson applies to both Flex 3 and Flex 4.

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

- Figure 1 (p. 4). Browser image at startup for the Flex 3 project.
- Figure 2 (p. 6) . A toolTip on the slider.
- Figure 3 (p. 8). Changing the height of the image.
- Figure 4 (p. 10). Flex Builder 3 Components tab exposed.
- Figure 5 (p. 11). Drag controls onto the Flex Builder 3 Design tab.
- Figure 6 (p. 13). The Flex Builder 3 Properties tab exposed.
- Figure 7 (p. 18). Browser image at startup for the hybrid project.
- Figure 8 (p. 20) . Flex Builder 4 Components tab exposed.
- Figure 9 (p. 22). Drag controls onto the Flash Builder 4 Design tab.

2.2.2 Listings

- Listing 1 (p. 12). XML code before setting properties for SliderChangeEvent01.
- Listing 2 (p. 14) . Beginning of XML code for SliderChangeEvent01.
- Listing 3 (p. 15). Create and condition the slider.
- Listing 4 (p. 16) . Import an image.
- Listing 5 (p. 22). Mxml code for the layout shown in Figure 9.
- Listing 6 (p. 23). Complete listing of SliderChangeEvent01.
- Listing 7 (p. 24). Complete listing of SliderChangeEvent02.
- Listing 8 (p. 25). Complete listing of SliderChangeEvent03.

2.3 Supplemental material

I also recommend that you 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

If you learn how to program using ActionScript 3, you will probably integrate large amounts of ActionScript code into your Flex projects to provide complex event handling. In the meantime, it is possible to provide simple event handlers in Flex by embedding a very small amount of ActionScript code in your Flex code. I will illustrate and explain that capability in this lesson.

 $^{^{4}}$ http://www.dickbaldwin.com/toc.htm

4 Preview

I encourage you to run (p. 1) the online version of the programs from this lesson before continuing.

Three Flex projects

I will present and explain three Flex projects in this lesson. The first project is named **Slider-ChangeEvent01**. This project was first developed using Flex Builder 3 and later developed using the Flex 3 compiler and Flash Builder 4. The results were essentially the same in both cases. This project uses classes from the Flex 3 (mx) library exclusively. (The screen shots shown in Figure 4, Figure 5, and Figure 6 are from Flex Builder 3.)

The second project

The second project is named **SliderChangeEvent02**. This project was developed using the Flex 4 compiler and Flash Builder 4. This is a hybrid project that uses classes from both the Flex 3 (mx) library and the Flex 4 (spark) library. The behavior of this project is similar to the behavior of the other project, but they look different in several ways that I will explain later.

The third project

The third project is named **SliderChangeEvent03**. This project is a modification of the hybrid project in which the classes used are drawn exclusively from the Flex 4 spark library.

Order of upcoming explanations

I will explain the Flex 3 project named **SliderChangeEvent01** in detail. Then I will explain the differences between that project and the hybrid project named **SliderChangeEvent02**. Finally, I will explain the differences between that project and the project named **SliderChangeEvent03**.

SliderChangeEvent01 output image at startup

The project named **SliderChangeEvent01** starts running in Flash Player with the image shown in Figure 1 appearing in the browser.



Browser image at startup for the Flex 3 project.

Figure 1: Browser image at startup for the Flex 3 project.

The image that you see in Figure 1 consists of two Flex 3 Label controls, one Flex 3 HSlider control, and one Flex 3 Image control arranged vertically and centered in the browser window.

The Application container

All XML documents must have a root element. The root of a Flex 3 application is a container element that is often called the **Application** container. (You can learn all about the **Application** container class at Adobe Flex 3.5 Language Reference 5 .)

Briefly, the **Application** container, (which corresponds to the root element in the Flex XML code), holds all other containers and components.

Vertical layout

By default, the Flex 3 **Application** container lays out all of its children vertically as shown in Figure 1. (As you will see later, this is not the case for the Flex 4 **Application** container.) The default vertical layout occurs when the **layout** attribute is not specified as is the case in this application. According to the Adobe Flex 3.5 Language Reference 6 , the **layout** property:

"Specifies the layout mechanism used for this application. Applications can use "vertical", "horizontal", or "absolute" positioning. Vertical positioning lays out each child component vertically from the top of the application to the bottom in the specified order. Horizontal positioning lays out each child component horizontally from the left of the application to the right in the specified order. Absolute positioning does no automatic layout and requires you to explicitly define the location of each child component. The default value is vertical."

A toolTip on the slider

If you point to the slider with your mouse, a tool tip showing the word *Height* will appear as shown in Figure 2.

 $^{^{5}} http://livedocs.adobe.com/flex/3/langref/index.html$

⁶http://livedocs.adobe.com/flex/3/langref/index.html

A toolTip on the slider.



Figure 2: A toolTip on the slider.

The slider's thumb

The little triangle that you see on the slider in these images is often referred to as the slider's *thumb*. As you will see later, the position of the thumb is intended to represent the height of the image below the slider. The left end of the slider represents a height of 100 pixels and the right end represents a height of 250 pixels (which just happens to be the actual height of the raw image).

Changing the height of the image

If you grab the thumb with the mouse and move it to the left or the right, two obvious visual effects occur. The first is that the value represented by the current position of the thumb is displayed above the thumb as shown in Figure 3.

(As you will see later, the value is also displayed in a Flex 4 application, but by default the appearance is white numerals on a black background.)

Changing the height of the image.

8



http://cnx.org/conte

Figure 3: Changing the height of the image.

The second visual effect

The second visual effect of moving the thumb is that the height of the image changes to the value represented by the position of the thumb on the slider.

(An **Image** object has a property named **maintainAspectRatio**. By default, the value of this property is true. Therefore, when the height is changed, the width changes in a proportional manner.)

Note that the upper-left corner of the image remains anchored to the same point as the height of the image changes as shown in Figure 3.

5 Discussion and sample code

5.1 The Flex 3 project named SliderChangeEvent 01

Creating the layout

Once you create your new Flex 3 Project, there are at least three ways that you can create your layout using Flex Builder 3:

- 1. Select the **Design** tab in the upper-middle panel of the IDE (see Figure 5) and drag your containers, controls, and other components from the **Components** tab onto your design window.
- 2. Select the **Source** tab in Figure 5 and write the raw XML code that defines your layout.
- 3. A combination of 1 and 2 above

Expose the components tab

When you select the **Design** tab in the upper-middle window of the IDE, the lower-left window changes to the appearance shown in Figure 4 with the Flex 3 (mx) **Components** tab exposed.



Flex Builder 3 Components tab exposed.

Figure 4: Flex Builder 3 Components tab exposed. $_{\rm http://cnx.org/content/m34633/1.1/}$ The list of available components that you see in Figure 4 also appears when you create a new project in Flash Builder 4 and specify the use of the Flex 3 compiler.

A list of available components

Although they aren't all shown in Figure 4 due to space limitations, the Flex Builder 3 **Components** tab lists all of the components that you can use in your Flex application grouped into the following categories:

- Custom
- Controls
- Layout
- Navigators
- Charts

Expose the design window

Selecting the **Design** tab mentioned above also exposes the Flex Builder 3 design window shown in Figure 5.



Figure 5: Drag controls onto the Flex Builder 3 Design tab.

A similar design window is exposed when you create a new project in Flash Builder 4 specifying the Flex 3 compiler and then select the **Design** tab. The purpose is the same but some of the items at the top of the design window in Flash Builder 4 are different.

Drag components onto the Design tab

You can drag components from the **Components** tab shown in Figure 4 onto the **Design** tab shown in Figure 5 to create your layout in the Flex Builder 3 design mode or in the Flexh Builder 4 design mode. As you do that, the corresponding XML code is automatically generated for you.

For example, Figure 5 shows the results of dragging two **Label** controls, one **HSlider** control, and one **Image** control from the **Components** tab of Figure 4 to the **Design** tab of Figure 5. (No attempt has been made to set property values on any of the controls shown in Figure 5.).

XML code before setting properties

If you select the **Source** tab at this point, you will see the XML code shown in Listing 1.

Listing 1: XML code before setting properties for SliderChangeEvent01.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

<mx:Label text="Label"/> <mx:Label text="Label"/> <mx:HSlider/> <mx:Image/>

</mx:Application>

Compile and run

As you can see, the XML code in Listing 1 is pretty sparse. You could compile and run the application at this point. All you would see would be two labels each containing the text **Label** and a slider covering the default numeric range from 0 to 10.

Put some meat on the bones

We will need to put some meat on the bones of this skeleton mxml code in order to create our Flex application. We can accomplish that by setting attribute values that correspond to properties of the controls.

Setting attribute values

Once again, we have three choices:

- 1. Go hardcore and edit the XML code shown in Listing 1 to add the necessary attributes.
- 2. Stay in **Design** mode, select each component in the **Design** tab, and use the **Flex Properties** tab shown in Figure 6 to set the properties on that component.
- 3. A combination of 1 and 2 above.

	non
1	D:
Te	xt: Label
Enable	ed: 📃 💌
✓ Style	
Style	<default style=""></default>
	Convert to CSS
	Text
A	
	Border/Alpha
4	
r	
- 1 - 2010	.+
Width	Height:
vviuu	neight. j

The Flex Builder 3 Properties tab exposed.



Figure 6: The Flex Builder 3 Properties tab exposed.

The Flex Properties tab

When you select the **Design** tab shown in Figure 5, the **Flex Properties** tab shown in Figure 6 appears in the bottom-right of the IDE.

The appearance of the **Flex Properties** tab depends on which component is selected in the **Design** tab. Figure 5 shows one of the **Label** controls selected, and Figure 6 shows the **Flex Properties** tab corresponding to a **Label** control.

You will see a very similar properties tab if you create a new Flash Builder 4 project and specify use of the Flex 3 compiler. Some of the items are in different locations than Figure 6 but it appears that the Flash Builder 4 properties tab has the same items for a Flex 3 project.

A variety of user input controls

The **Flex Properties** tab contains a variety of user input controls that allow you to specify values for many of the commonly used properties that are allowed for the selected component.

Note, however, that the documentation for the **Label** control lists many properties that are not supported by the **Flex Properties** tab shown in Figure 6. You can increase the number of properties shown in the tab by selecting one of the controls at the top of the tab that converts the display into an alphabetical list of properties. However, even this doesn't seem to show all of the properties defined by and inherited by some components.

If you need to set properties that are not supported by the **Flex Properties** tab, you probably have no choice but to select the **Source** tab shown in Figure 5 and write mxml code for those properties.

Will explain the code in fragments

I will explain the code for this Flex application in fragments. A complete listing of the application is provided in Listing 6 near the end of the lesson.

Beginning of XML code for SliderChangeEvent01

The primary purpose of this application is to illustrate the use of inline event handling for Flex 3 slider change events.

The application begins in Listing 2 which shows the beginning of the **Application** element and the two complete **Label** elements shown at the top of Figure 1.

Listing 2: Beginning of XML code for SliderChangeEvent01.

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

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

<mx:Label text="Put your name here" fontSize="16" fontWeight="bold"/>

<mx:Label text="Image Height in Pixels" fontWeight="bold" fontSize="14"/>

Make it easy - drag and drop

The two **Label** elements were created by dragging **Label** controls from the **Components** tab shown in Figure 4 onto the **Design** tab shown in Figure 5. Then the attribute values were set using the **Flex Properties** tab shown in Figure 6.

The attributes shown in Listing 2 represent common properties of a text label and shouldn't require further explanation.

Create and condition the slider

Listing 3 adds a horizontal slider *(HSlider)* control to the application and sets the attributes that control both its appearance and its behavior.

Listing 3: Create and condition the slider.

```
<mx:HSlider minimum="100" maximum="250" value="250"
toolTip="Height"
change="myimg.height=event.currentTarget.value"
liveDragging="true" />
```

The slider is a little more complicated than a label and deserves a more thorough explanation.

The numeric properties

Recall that a slider represents a range of numeric values. The position of the thumb at any instant in time selects a value from that range. The following three attributes shown in Listing 3 deal with the slider and its numeric range:

- minimum the numeric value represented by the left end of a horizontal slider.
- **maximum** the value represented by the right end of a horizontal slider.
- **value** the value that specifies the initial position of the thumb when the slider is constructed and first presented in the application's window.

The toolTip property

As you have probably already guessed, the value of the **toolTip** property specifies the text that appears in the tool tip when it is visible as shown in Figure 2.

The change property

This is where thing get a little more interesting. As the user moves the thumb to the left or right, the slider fires a **continuous** stream of **change** events. You might think of this as the slider yelling out "Hey, the position of my thumb has been changed." over and over as the thumb is being moved. (Also see the discussion of the liveDragging (p. 16) property later.)

An event handler

The value that is assigned to the **change** attribute in Listing 3 is often referred to as an event handler. This value specifies what the application will do each time the slider fires a **change** event.

Three ways to handle events in Flex

There are at least three ways to handle event notifications in Flex:

- Registering an event handler in mxml
- Creating an **inline event handler** in the mxml definition
- Registering an event listener through ActionScript

The XML code in Listing 3 uses the inline approach.

The inline approach

The advantage of using the inline approach, (at least insofar as my **Introduction to XML**, students are concerned), is that it doesn't require you to create a **Script** element within the mxml or to create a separate ActionScript file.

Handling the slider's change event

Now consider the code that begins with the word **change** in Listing 3. The code within the quotation marks can be a little hard to explain, but I will give it a try. (The code in quotation marks is actually an ActionScript code fragment.)

Think of it this way

There is a Flex/ActionScript class named **Event**. The reference to **event** in Listing 3 is a reference to an object of the **Event** class that comes into being each time the slider fires a **change** event .

The **Event** object encapsulates a property named currentTarget, which is described in the Flex 3 documentation as follows:

"The object that is actively processing the Event object with an event listener. For example, if a user clicks an OK button, the current target could be the node containing that button or one of its ancestors that has registered an event listener for that event."

The currentTarget is the slider

In this application, the value of **currentTarget** points to the slider which is firing **change** events as the user moves the thumb.

The value property of an HSlider object

The slider is an object of the **HSlider** class, which has a property named **value**. The **value** property contains the current position of the thumb and is a number between the *minimum* and *maximum* property values.

Get the current value of the thumb

Therefore, each time the slider fires a **change** event, the code on the right side of the assignment operator within the highlighted quotation marks in Listing 3 gets the numeric value that indicates the current position of the thumb.

Cause the image to be resized

This value is assigned to the **height** property of the image, causing the overall size of the image to be adjusted, if necessary, to match the current position of the slider's thumb. (I could go into more detail as to the sequence of events that causes the size of the image to change, but I will leave that as an exercise for the student.)

The liveDragging property

That leaves one more attribute or property for us to discuss in Listing 3: **liveDragging**. This one is much easier to understand.

The Flex 3 documentation has this to say about the **liveDragging** property:

"Specifies whether live dragging is enabled for the slider. If false, Flex sets the value and values properties and dispatches the change event when the user stops dragging the slider thumb. If true, Flex sets the value and values properties and dispatches the change event continuously as the user moves the thumb. The default value is false."

If liveDragging is false...

If you were to modify the code in Listing 3 to cause the value of the **liveDragging** property to be false (or simply not set the attribute value to true), the slider would only fire change events each time the thumb stops moving (as opposed to firing a stream of events while the thumb is moving). This, in turn, would cause the size of the image to change only when the thumb stops moving instead of changing continuously (p. 15) while the thumb is moving.

An Image control

The Flex 3 documentation tells us:

The Image control lets you import JPEG, PNG, GIF, and SWF files at runtime. You can also embed any of these files and SVG files at compile time by using @Embed(source='filename').

The primary output that is produced by compiling a Flex application is an swf file that can be executed in Flash Player.

The documentation goes on to explain that by using **@Embed**, you can cause resources such as images to be embedded in the swf file.

The advantage to embedding is that embedding the resource eliminates the requirement to distribute the resource files along with the swf files. The disadvantage is that it causes the swf file to be larger.

Import an image

Listing 4 imports an image from the file named **myimage.jpg** that is located in the **src** folder of the project tree. This image is embedded in the swf file when the Flex application is compiled.

Listing 4: Import an image.

```
<mx:Image id="myimg" source="@Embed('myimage.jpg')"
height="250">
</mx:Image>
```

</mx:Application>

The *id* property

Setting the **id** property on the image to **myimg** makes it possible to refer to the image in the change-event code in Listing 3.

(Note that there is no requirement to set the value of the *id* property to be the same as the name of the image file as was done in Listing 4.)

The *height* property

Setting the **height** property of the image to 250 pixels in Listing 4 causes the image height to be 250 pixels when it is first displayed as shown in Figure 1.

The end of the application

 $\label{eq:listing4} \mbox{Listing4} \mbox{ contains the closing tag for the } {\bf Application} \mbox{ element signaling the end of the Flex 3 application} \\ {\bf named} \ \ {\bf SliderChangeEvent01} \ .$

5.2 The hybrid Flex3-4 project named SliderChangeEvent02

The mxml project code

The mxml code for this project is shown in its entirety in Listing 7. If you examine this code you will see that:

- It uses a Flex 4 spark **s:Application** element instead of a Flex 3 **mx:Application** element.
- It declares the standard set of Flex 4 namespaces.
- It uses a spark s:VGroup element as the container for the following Flex 3 components. (Note that the Flex 3 project in Listing 6 doesn't require another container in addition to the mx:Application container.) :
 - \cdot mx:Label
 - \cdot mx:Label
 - · mx:HSlider
 - · mx:Image

Otherwise, the mxml code for this project is the same as the code for the Flex 3 project shown in Listing 6. The mixture of spark and mx components causes this to be a hybrid Flex 3-4 project.

Visual appearance of the project

If you run (p. 1) the online version of the project named **SliderChangeEvent02**, you should see an initial screen display similar to Figure 7.



Browser image at startup for the hybrid project.

Figure 7: Browser image at startup for the hybrid project.

Compare with the Flex 3 project

By comparing this screen output for the hybrid project with the screen output for the Flex 3 project in Figure 1, you can immediately spot several significant differences:

- The background is white instead of gray.
- The labels, the slider, and the image are not centered horizontally in the browser window.
- The appearance of the thumb on the slider is a circle instead of a triangle.

Behavior of the project

If you move the slider with the mouse, you will see that the behavior is essentially the same as the Flex 3 version of the project, including the display of a tool tip as shown in Figure 2 and the display of the slider value as shown in Figure 3.

The spark s:Application element

The Flex 4 spark **s:Application** element differs from the Flex 3 **mx:Application** element in several ways including the following:

- **Default layout:** Unlike the **mx:Application** element, the **s:Application** element does not have a default vertical (p. 5) layout. By default, all components placed in the **s:application** element are placed in the upper-left corner. (*The* **s:VGroup** container element was used in Listing 7 to resolve this issue.)
- **Default background color:** The default background color of the **s:Application** element is white, whereas the default background color of the **mx:Application** element is gray.
- Horizontal positioning: Unlike the mx:Application element which centers it components horizontally by default, the default position of components placed in the s:Application element is the upper-left corner.

The spark s:VGroup element

As shown in Listing 7, the **s:VGroup** element can be used to arrange the components in a vertical sequence from top to bottom. However, placing components in an **s:VGroup** element does not cause them to be centered horizontally. Instead, by default, the components end up on the left side of the container as shown in Figure 7. If you want the components to be centered, you must write additional code to cause that to happen.

The Flash Builder 4 Components tab

When you create a new Flex project in Flash Builder 4, if you specify the use of the Flex 3 compiler, the **Components** tab in the resulting IDE will look like Figure 4. However, if you specify the Flex 4 compiler when you create the new project, the **Components** tab will look like Figure 8.



Flex Builder 4 Components tab exposed.

 $\begin{array}{c} {}^{\rm http://cnx.org/content/m34633/1.1/} \\ {\bf Figure 8:} \ {\rm Flex \ Builder \ 4 \ Components \ tab \ exposed.} \end{array}$

Numerous differences

If you compare Figure 8 with Figure 4, you will see numerous differences between the two lists. Some of the names are the same and some of the names are different. Even though some of the names are the same, most of the components that you see in Listing 8 are Flex 4 spark components and the components that you see in Figure 4 are Flex 3 mx components and they are represented by different classes in the class library.

However, as you will see later, the **Image** component in the Flex 4 Component list is actually a Flex 3 mx component. That may also be the case for some of the other components as well.

Same name doesn't guarantee same appearance or same behavior

While the appearance and behavior of a Flex 4 spark component may be the same as the appearance and behavior of a Flex 3 mx component with the same name, there is no guarantee that will be the case. They are entirely different components and the only way you can be sure is to study the documentation.

As you saw earlier, the appearance of an **mx:HSlider** used inside an **s:Application** element is different from an **mx:HSlider** used inside an **mx:Application** element. Therefore, if the appearance and behavior of the components in your project are really critical, you should probably avoid mixing Flex 3 and Flex 4 components.

No drag-and-drop support for hybrid projects

Another ramification of the fact that the components in Figure 8 are spark components is that you cannot create hybrid projects using drag-and-drop programming alone. If you drag the components in Figure 8 into the **design** pane of Flash Builder 4, your project will be populated with Flex 4 spark components. If you want the project to be populated with Flex 3 mx components, you will have to manually edit the mxml code to accomplish that. That may be another reason to avoid hybrid projects.

Drag-and-drop results

Figure 9 shows the results of dragging one VGroup layout, two Label controls, one HSlider control, and one Image control from the Flash Builder 4 Components panel into the Design panel.



Drag controls onto the Flash Builder 4 Design tab.

Figure 9: Drag controls onto the Flash Builder 4 Design tab.

Compare Figure 9 with Figure 5 to see the differences in background color and layout. Mxml code for the layout shown in Figure 9

Listing 5 shows the Flex 4 mxml code that corresponds to the layout shown in Figure 9.

Listing 5: Mxml code for the layout shown in Figure 9.

```
</fx:Declarations>
<s:VGroup x="0" y="0" width="200" height="200">
<s:Label text="Label"/>
<s:Label text="Label"/>
<s:HSlider/>
<mx:Image/>
</s:VGroup>
</s:Application>
```

Mostly Flex 4 spark components

The most important thing to note about Listing 5 is that the **VGroup**, **Label**, and **HSlider** components that were dragged from the **Component** tab shown in Figure 8 into the **Design** panel shown in Figure 9 are all declared using the spark (s) namespace. Curiously, however, the **Image** control shows up in the code as **mx:Image** instead of **s:Image**.

5.3 The Flex 4 project named SliderChangeEvent03

Updating the mxml code in Listing 5 by applying the properties from Listing 7 to the **Label** and **HSlider** components produces the complete Flex 4 project named **SliderChangeEvent03** shown in Listing 8.

If you run (p. 1) the online versions of **SliderChangeEvent0** 2 and **SliderChangeEvent03** sideby-side in different browser windows, you will probably notice a few subtle differences in the look and feel of the two programs. Here are some of the differences that I have noticed:

- There is less space between the labels and the slider in the Flex 4 version.
- There is less space between the edge of the Flash window and the top and left ends of the labels and the slider in the Flex 4 version.
- The overall length of the slider is shorter in the Flex 4 version.
- The treatment of the little popup window that shows the value of the slider is different between the two. It has black letters on a cream-colored background in the hybrid version as in Figure 3, but it has white letters on a black background in the Flex 4 version.

6 Run the programs

I encourage you to run (p. 1) the online versions of the programs from this lesson. Then copy the code from Listing 6, Listing 7, and Listing 8. Use that code to create Flex projects of your own. Compile and run your 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 Flex resources as a separate document. Search for Flex Resources in the Connexions search box.

8 Complete program listing

Complete listings of the programs discussed in this lesson are shown in Listing 6, Listing 7, and Listing 8 below.

Listing 6: Complete listing of SliderChangeEvent01.

```
<?xml version="1.0" encoding="utf-8"?>
<!--
SliderChangeEvent01
Illustrates the use of inline event handling for slider
change events.
-->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Label text="Put your name here" fontSize="16"
       fontWeight="bold"/>
    <mx:Label text="Image Height in Pixels"
        fontWeight="bold" fontSize="14"/>
    <mx:HSlider minimum="100" maximum="250" value="250"
        toolTip="Height"
        change="myimg.height=event.currentTarget.value"
        liveDragging="true" />
    <mx:Image id="myimg" source="@Embed('myimage.jpg')"
       height="250">
    </mx:Image>
</mx:Application>
```

Listing 7: Complete listing of SliderChangeEvent02.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx">
  <s:VGroup>
    <mx:Label
      text="Put your name here" fontSize="16"
      fontWeight="bold"/>
    <mx:Label
      text="Image Height in Pixels"
     fontWeight="bold" fontSize="14"/>
    <mx:HSlider
     minimum="100"
     maximum="250"
      value="250"
      toolTip="Height"
      change="myimg.height=event.currentTarget.value"
      liveDragging="true" />
```

```
<mx:Image
id="myimg" source="@Embed('myimage.jpg')"
height="250">
</mx:Image>
</s:VGroup>
```

```
</s:Application>
```

Listing 8: Complete listing of SliderChangeEvent03.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"</pre>
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               minWidth="955" minHeight="600">
  <s:VGroup x="0" y="0" width="200" height="200">
    <s:Label
      text="Put your name here" fontSize="16"
      fontWeight="bold"/>
    <s:Label
      text="Image Height in Pixels"
      fontWeight="bold" fontSize="14"/>
    <s:HSlider
      minimum="100"
      maximum="250"
      value="250"
      toolTip="Height"
      change="myimg.height=event.currentTarget.value"
      liveDragging="true" />
    <mx:Image id="myimg" source="@Embed('myimage.jpg')"
              height="250"/>
  </s:VGroup>
</s:Application>
```

9 Miscellaneous

NOTE: Housekeeping material

- Module name: Handling Slider Change Events in Flex 3 and Flex 4
- Files:
 - \cdot Flex0104\Flex0104.htm
 - \cdot Flex0104\Connexions\FlexXhtml0104.htm

NOTE: **PDF disclaimer:** Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

OpenStax-CNX module: m34633

-end-