The Flex Event Model

In Java, events and event handling occur by setting up special classes. In Flex, the entire framework is event-oriented making Flex a framework that supports loose-coupling among its components (a good thing!)

Using events

Using events in Flex is a two-step process. First, you write a function or class method, known as an *event listener* or *event handler*, that responds to events. The function often accesses the properties of the Event object or some other settings of the application state. The signature of this function usually includes an argument that specifies the event type being passed in.

The following example shows a simple event listener function that reports when a control triggers the event that it is listening for:

```
<?xml version="1.0"?>
<!-- events/SimpleEventHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
      creationComplete="initApp();">

    <mx:Script><![CDATA[
        import mx.controls.Alert;

        private function initApp():void {
            b1.addEventListener(MouseEvent.CLICK, myEventHandler);
        }

        private function myEventHandler(event:Event):void {
            Alert.show("An event occurred.");
        }

    ]]></mx:Script>

    <mx:Button id="b1" label="Click Me"/>

</mx:Application>
```

As you can see in this example, you also register that function or class method with a display list object by using the addEventListener() method.

Most Flex controls simplify listener registration by letting you specify the listener inside the MXML tag. For example, instead of using the `addEventListener()` method to specify a

listener function for the Button control's `click` event, you specify it in the `click` attribute of the `<mx:Button>` tag:

```
<?xml version="1.0"?>
<!-- events/SimplerEventHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        import mx.controls.Alert;

        private function myEventHandler(event:Event):void {
            Alert.show("An event occurred.");
        }
    ]]></mx:Script>

    <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>

</mx:Application>
```

This is equivalent to the `addEventListener()` method in the previous code example. However, it is best practice to use the `addEventListener()` method. This method gives you greater control over the event by letting you configure the priority and capturing settings, and use event constants. In addition, if you use `addEventListener()` to add an event handler, you can use `removeEventListener()` to remove the handler when you no longer need it. If you add an event handler inline, you cannot call `removeEventListener()` on that handler.

Each time a control generates an event, Flex creates an Event object that contains information about that event, including the type of event and a reference to the dispatching control. To use the Event object, you specify it as a parameter in the event handler function, as the following example shows:

```
<?xml version="1.0"?>
<!-- events/EventTypeHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        import mx.controls.Alert;

        private function myEventHandler(e:Event):void {
            Alert.show("An event of type '" + e.type + "' occurred.");
        }
    ]]></mx:Script>

    <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>

</mx:Application>
```

If you want to access the Event object in an event handler that was triggered by an inline event, you must add the `event` keyword inside the MXML tag so that Flex explicitly passes it to the handler, as in the following:

```
<mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>
```

You are not required to use the Event object in a handler function. The following example creates two event handler functions and registers them with the events of a ComboBox control. The first event handler, `openEvt()`, takes no arguments. The second event handler, `changeEvt()`, takes the Event object as an argument and uses this object to access the `value` and `selectedIndex` of the ComboBox control that triggered the event.

```
<?xml version="1.0"?>
<!-- events/MultipleEventHandlers.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        private function openEvt():void {
            forChange.text="";
        }

        private function changeEvt(e:Event):void {
            forChange.text =
            "Value: " + e.currentTarget.value + "\n" +
            "Index: " + e.currentTarget.selectedIndex;
        }
    ]]></mx:Script>

    <mx:ComboBox open="openEvt()" change="changeEvt(event)">
        <mx:dataProvider>
            <mx:Array>
                <mx:String>AK</mx:String>
                <mx:String>AL</mx:String>
                <mx:String>AR</mx:String>
            </mx:Array>
        </mx:dataProvider>
    </mx:ComboBox>

    <mx:TextArea id="forChange" width="150" height="100"/>

</mx:Application>
```