

Creating ActionScript 3.0 components in Flash CS3 Professional – Part 4: Events

Jeff Kameron

Adobe

If you skipped the three articles leading up to this one, you might want to begin with Part 1 of the series where you can download the sample files for the entire series. Or if you prefer, you can download the sample files for Part 4 only via the link below to follow along with the topics discussed in this article.

At the end of the last part in this article series, our MenuBar component was just tracing out the name of the menu item when it was selected. Clearly, we'll need to add some events to cause the component to take some action when an item in the menu is selected. For this example, I decided to keep the event support simple and only dispatch an event upon item selection. You could imagine the events that might be called when a drop-down menu is opened, when the user rolls off and the menu is closed, when the user rolls over a menu bar item or drop-down menu item, etc. If you review the Menu and MenuBar components included in Flex 2, you can see examples of components that support these events.

The event classes dispatched by the Flash Player and the User Interface components follow some straightforward patterns that we should follow as a best practice to maintain consistency and usability. These patterns are thoroughly explained in the `MenuEvent` example.

- An event class must inherit from `flash.events.Event`
- An event class should define constant strings for every possible value of the `type` property of the event class
- The constructor should take all of the parameters that its base class's constructor takes, (with the same default arguments) and should add all of its properties to the parameter list with default values
- All public properties should use get functions and, if they are writable, set functions
- An event class should override the method `clone()`
- An event class should override the method `toString()`

Now that we have our set of guidelines established, it is time to begin putting these practices into place. In the next section of this article, we'll see how to extend the Event class with our MenuEvent class to add interactivity.

Implementing MenuEvent

To add events to our MenuBar component, we'll add some code that extends the Event class to our component project. To make these steps clear, I'll walk through my implementation of `fl.example.MenuEvent` to illustrate these principles.

```
package fl.example {  
  
    import flash.events.Event;  
  
    public class MenuEvent extends Event {
```

`MenuEvent` directly extends `Event`, but an event class does not need to extend `Event` directly. There are several reasons you might want to do this. For example, the FLVPlayback component's `fl.video.VideoProgressEvent` needs a `bytesLoaded` property and a `bytesTotal` property, so extending `flash.events.ProgressEvent` was natural. Also, the FLVPlayback component's `fl.video.SkinErrorEvent` extends `flash.events.ErrorEvent`, which adds error handling to the event. By adding this special property, we're ensuring that the Flash Player will throw an error if the event is not handled.

```
        public static const ITEM_SELECTED:String = "itemSelected";
```

There is only one type of `MenuEvent`, `itemSelected`, so I only declared one string constant in the code. The string names for event types should always be in [camel case](#)—the same naming convention used for property and method names. The name for the static constant should be the same as the string name, but it should be in all caps, with each word separated by an underscore.

```
        private var _menuIndex:int;  
        private var _menuLabel:String;  
        private var _itemIndex:int;  
        private var _itemLabel:String;  
  
        public function MenuEvent( type:String, bubbles:Boolean=false,  
cancelable:Boolean=false,  
                                menuIndex:int=-1, menuLabel:String=null,  
itemIndex:int=-1, itemLabel:String=null ) {  
            super(type, bubbles, cancelable);  
            this.menuIndex = menuIndex;  
            this.menuLabel = menuLabel;  
            this.itemIndex = itemIndex;  
            this.itemLabel = itemLabel;  
        }
```

All properties supported by `MenuEvent` have matching private variables with the same name preceded by an underscore. Rather than setting the private variables directly, the constructor invokes the set methods for each property. Also, the constructor's parameter list begins with all of the parameters from the `Event` constructor: `type`, `bubbles` and `cancelable`, and passes these values into the `super` constructor. Notice

that even though two of these parameters, `bubbles` and `cancelable`, will always have the default values in my implementation of the event class, I included them in the constructor's parameter list to follow the best practices. Every event class has these properties in the constructor; always writing your code consistently makes working with events easier and more predictable.

```
public function get menuIndex():int {
    return _menuIndex;
}

public function set menuIndex(value:int):void {
    _menuIndex = value;
}

public function get menuLabel():String {
    return _menuLabel;
}

public function set menuLabel(value:String):void {
    _menuLabel = value;
}

public function get itemIndex():int {
    return _itemIndex;
}

public function set itemIndex(value:int):void {
    _itemIndex = value;
}

public function get itemLabel():String {
    return _itemLabel;
}

public function set itemLabel(value:String):void {
    _itemLabel = value;
}
```

I made all of the properties read/write. Since all of the getter and setter implementations are trivial, I could have declared the properties as public variables and eliminated the getter and setter functions altogether, but this would not have been following the best practice. By implementing the properties this way, a subclass is used to override a property, which a user of the component may wish to do—even if I will never change the defaults. In general it is a best practice to implement all public properties this way in all of your classes, to allow maximum flexibility for yourself and the other developers who will use your code.

```
override public function toString():String {
    return formatToString("MenuEvent", "type", "bubbles",
"cancelable", "eventPhase", "menuIndex", "menuLabel", "itemIndex",
"itemLabel");
}
```

Because of the `toString()` implementation, the output of `trace(new MenuEvent())` looks like this:

```
[MenuEvent type="itemSelected" bubbles=false cancelable=false eventPhase=2
menuIndex=-1 menuLabel=null itemIndex=-1 itemLabel=null]
```

You can use the `toString()` method as a very useful debugging aid. The `formatString()` method is a public method of `Event`, which is specifically designed to help implement `toString()` methods for events classes.

```
        override public function clone():Event {
            return new MenuEvent(type, bubbles, cancelable, menuIndex,
menuLabel, itemIndex, itemLabel);
        }
    }
```

An implementation of the `clone()` method should always call the constructor with all of the parameters listed, as shown above. The Flash Player calls the `clone()` method when an event is redispached, so it is a very important method to implement. When an event is handled, the handler method can call `dispatchEvent` on the event object it received. When an event is dispatched a second time, a copy is made of it and the `target` and `currentTarget` properties are updated to reflect the object that dispatched the event.

Next we'll begin adding mouse events, to add interactivity to the `MenuBar` component.

Dispatching MenuEvent

If you've been following along with each part of this article series, you'll remember that up until now I had been handling the `mouseUp` event on my drop-down menus with this code:

```
// we aren't doing any real event dispatching yet, just tracing
trace(cellRenderer.data.label);
closeMenuBar();
```

In the code above, I was only tracing the label, so the next step was to write code to get the menu bar item index and label, as well as the drop-down menu index. When you examine the code below, be sure to read the comments for the details on what the code is doing. As I worked on this section, I was able to figure out how to make this work by researching the documentation in the [fl.controls.listClasses.ICellRenderer](#) and [fl.controls.listClasses.ListData](#) sections of the [ActionScript 3.0 Language and Components Reference](#).

```
// we will dispatch an event!
// first get the list that this was selected from, so we can use it to
determine
// which menu bar item this drop-down menu comes from
```

```

var theMenu:List = cellRenderer.listData.owner as List;
// the menu bar index matches the index of the List in myMenus array
var menuIndex:int = theMenus.indexOf(myMenus);
// menuIndex to get the label from myMenuBar's dataProvider
var menuLabel:String = myMenuBar.dataProvider.getItemAt(menuIndex).label;
// the drop-down menu index is in the listData property
var itemIndex:int = cellRenderer.listData.index;
// the drop-down menu label is in the data property
var itemLabel:String = cellRenderer.data.label;
// dispatch an event of type itemSelected with bubbles and cancelable both
set to false
dispatchEvent(new MenuEvent( MenuEvent.ITEM_SELECTED, false, false,
menuIndex, menuLabel, itemIndex, itemLabel));
closeMenuBar();

```

Once an event is created, dispatching it is very simple. I just had to call `dispatchEvent()` with my event object and the Flash Player takes care of the rest.

Finally, I added a little code to my test.fla file to ensure that my ActionScript code was working correctly:

```

import fl.example.*;
menuBar.addEventListener(MenuEvent.ITEM_SELECTED, handleItemSelected);
function handleItemSelected(e:MenuEvent):void {
    trace(e);
}

```

Where to go from here

In Part 5 of this article series we'll get into aesthetics and learn how to customize the appearance of the MenuBar component with styles and skins. In the next installment you'll learn how to adjust the look and feel of the components that you build. Using the same process, you can also change the display of the components included with Flash CS3 to match your projects.

If you are more familiar with ActionScript 2.0 and want to get up to speed on using ActionScript 3.0, be sure to visit the [ActionScript 2.0 Migration page](#).

To learn more about working with events in ActionScript 3.0 check out these Developer Center articles:

[Introduction to event handling in ActionScript 3.0](#)
[ActionScript 3.0 overview](#)

About the author

Jeff Kameron is a computer scientist at Adobe Systems who has worked on the Flash authoring team since 2002.