

# **Introduction to Adobe Flex 3**

## **Guide to Custom Events**

## **Trademarks**

ChikaraDev and the ChikaraDev logo are trademarks of ChikaraDev. Such trademarks may be registered in the United States or in other jurisdictions, including internationally. This manual may include trademarks, service marks, or trade names of Adobe Systems Incorporated, Inc. and other companies. Such trademarks, service marks, or trade names may be registered in the United States or in other jurisdictions, including internationally.

## **Third-Party Information**

This manual contains information such as links to third-party websites that are not under the control of ChikaraDev, and ChikaraDev is not responsible for the content on any linked site. If you access a third-party website mentioned in this manual, then you do so at your own risk. ChikaraDev has provided these links only as a convenience, and the inclusion of the link does not imply that ChikaraDev endorses or accepts any responsibility for the content on those third-party sites.

Copyright © 2008 ChikaraDev. All rights reserved.

The software described in this manual is provided under an agreement with Adobe Systems Incorporated, and such software can only be used in accordance with the terms of the agreement provided by Adobe Systems. Software code described and provided in this manual is provided under an agreement with ChikaraDev. The software can only be used in accordance with the terms of the agreement.

No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, photocopying, manual, optical, recording, or otherwise, outside the license agreement accompanying these materials, without the prior written permission of ChikaraDev. ChikaraDev claims copyright in this program and documentation as an unpublished work, revisions of which were first licensed on the date indicated in the foregoing notice. Claim of copyright does not imply waiver of other rights of ChikaraDev and its subsidiaries.

Information in this manual may change without notice and does not represent a commitment on the part of ChikaraDev.

## **NOTICE OF LIABILITY**

This information in these training materials is distributed on an “AS IS” basis, without warranty of any kind, either express or implied. While every precaution has been taken in the preparation of these materials, neither ChikaraDev nor its licensors shall have any liability to any person or entity with respect to liability, loss, or damage caused or alleged to be caused directly or indirectly by the instructions contained in these materials or by the computer software and hardware products described herein.

First Edition: July 2008 - ChikaraDev - Cupertino, CA 95014 USA

# Overview of Custom Events in Adobe Flex

Most Flex applications should be designed using MXML or ActionScript components that can be reused in multiple applications. However, hard-coding communication between the components and the main application using `Application.application`, `parentDocument`, and `parentApplication`, can result in a “tightly-coupled” application that is more difficult to maintain and extend.

Custom events can be used in Adobe Flex 3 to design an application that is “loosely coupled”. Something happens somewhere in the application, a custom event is dispatched, and the main application or another component is listening for that event.

A good example of using custom events is an application making use of a login component. If the user successfully logs in, the main application might display a welcome message.

## Simple Custom Event Example

The example code for this example involves code in these files:

- `CustomEvents1.mxml` (main application file)
- `com/myCompany/classes/events/LoginEvent.as` (custom event file)
- `components/Login.mxml` (login component file)

### Points to Note about the Code

The main application is a bare bones application with only an `ApplicationControlBar` that displays a welcome message if the user successfully logs in. Communication between the login component and the main application is achieved by the login component dispatching a custom event upon successful login, and the main app listening for that event.

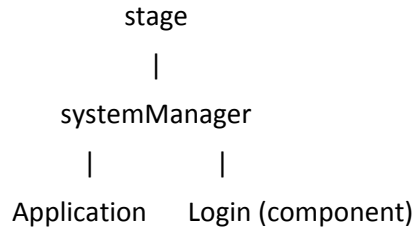
Login failure is handled by the Login component. The application should not be involved in login, except to display a welcome message for successful login, which is an indirect result of login.

The `LoginEvent` class extends the Flex `Event` class. The constructor calls the parent class constructor using `super()`, passing in the custom event type (`LoginEvent.SUCCESS`). The event type is necessary for creating an instance of any event. This makes it possible to identify what type of event was dispatched, so the proper event listeners can respond to the event.

```
super(LoginEvent.SUCCESS);
```

In the following code, we use `systemManager.addEventListener` rather than `this.addEventListener` ('this' means the application in this case) because the event will be dispatched by a popup, and pop-ups and the application share the systemManager, but are otherwise not in the same display list hierarchy.

```
systemManager.addEventListener(LoginEvent.SUCCESS, handleLogin, true);
```



**This is an important concept, because if you are listening in the Application for events dispatched by pop-ups, you must listen at the systemManager level, not the Application level.**

The third argument of the `addEventListener` method above is 'true' so the `systemManager` listens during the 'capture' event propagation phase. By default, no container listens during the capturing phase. The default value of the `use_capture` argument is false. The only way to add a listener during this phase is to pass true for the `use_capture` argument when calling the `addEventListener()` method.

We could have instead allowed the `systemManager` to listen during the 'bubble' event propagation phase, but then the third parameter to the `addEventListener()` method would have been omitted (default is false), and we would have the following line in the custom event class:

```
super(LoginEvent.SUCCESS, true);
```

The second argument 'true' indicates the custom event 'bubbles'.

In the custom event file, we are required to override the `Event.clone()` method. The `clone()` method returns a cloned copy of the event object by setting the type property and any new properties in the clone. Typically, you define the `clone()` method to return an event instance created with the new operator.

### CustomEvents1.mxml (main application file)

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  creationComplete="init();">
  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      import mx.managers.PopUpManager;
      import components.*;
      import com.myCompany.classes.events.*;

      private function init():void{
        // Need to add the event listener to the systemManager, not to 'this'
```

```

// (the application), because the event is dispatched from a popup,
// and pop-ups and the application share the systemManager, but are
// otherwise not in the same display list hierarchy.

// Passing 'true' for the third parameter indicates that this event listener
// listens for events in the 'capture' phase of event propagation.
systemManager.addEventListener(LoginEvent.SUCCESS, handleLogin, true);
login();
}

private function login():void{
    var loginComp:Login = new Login();
    PopUpManager.addPopUp(loginComp, this, true);
}

private function logInOut(evt:MouseEvent):void{
    if(evt.currentTarget.label == "Login"){
        login();
    }else{
        welcomeLbl.visible = false;
        loginBtn.label = "Login";
    }
}

private function handleLogin(evt:LoginEvent):void{
    welcomeLbl.visible = true;
    loginBtn.label = "Logout";
}
]]>
</mx:Script>
<mx:ApplicationControlBar width="100%" dock="true">
    <mx:Canvas width="100%" height="100%">
        <mx:Label id="welcomeLbl" left="50" verticalCenter="0"
            fontSize="20" color="0xFFFFFFFF" text="Welcome !!!"
            visible="false"/>
        <mx:Label id="titleLbl" horizontalCenter="0" verticalCenter="0"
            text="Sample Application" fontSize="24"/>
        <mx:Button id="loginBtn" right="30" label="Login"
            verticalCenter="0" click="logInOut(event);"/>
    </mx:Canvas>
</mx:ApplicationControlBar>
</mx:Application>

```

### **com/myCompany/classes/events/LoginEvent.as (custom event file)**

```

package com.myCompany.classes.events
{
    import flash.events.Event;

    public class LoginEvent extends Event{
        public static const SUCCESS:String = "success";

        public function LoginEvent(){
            super(LoginEvent.SUCCESS);
        }

        // Override the inherited clone() method.
        override public function clone():Event {
            return new LoginEvent();
        }
    }
}

```

```
}  
}  
}
```

### components/Login.mxml (login component file)

```
<?xml version="1.0" ?>  
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"  
  title="Login" horizontalScrollPolicy="off"  
  verticalScrollPolicy="off" creationComplete="init();"  
  defaultButton="{loginBtn}">  
  <mx:Script>  
    <![CDATA[  
      import mx.controls.Alert;  
      import com.myCompany.classes.events.*;  
      import mx.collections.ArrayCollection;  
      import mx.managers.PopUpManager;  
  
      private var testUser:String = "user1";  
      private var testPwd:String = "testme";  
  
      private function init():void{  
        userTxt.text = "";  
        passwordTxt.text = "";  
        PopUpManager.centerPopUp(this);  
      }  
  
      private function login(evt:Event):void{  
        if(userTxt.text == testUser && passwordTxt.text == testPwd){  
          this.dispatchEvent(new LoginEvent());  
          PopUpManager.removePopUp(this);  
        }else{  
          mx.controls.Alert.show("Login Failed!!!");  
        }  
      }  
    ]]>  
  </mx:Script>  
  <mx:Form id="loginForm">  
    <mx:FormItem label="Username:">  
      <mx:TextInput id="userTxt"/>  
    </mx:FormItem>  
    <mx:FormItem label="Password:">  
      <mx:TextInput id="passwordTxt"  
        displayAsPassword="true"/>  
    </mx:FormItem>  
  </mx:Form>  
  <mx:ControlBar horizontalAlign="center">  
    <mx:Button label="Login" id="loginBtn" click="login(event);"/>  
  </mx:ControlBar>  
</mx:Panel>
```

# Custom Event Handler Defined in MXML Example

The example code for this example involves code in these files:

- CustomEvents2.mxml (main application file)
- com/myCompany/classes/events/LoginEvent.as (custom event file)
- components/Login.mxml (login component file)

## Points to Note about the Code

In this example, the main application does not define an event listener for the custom event using the `addEventListener()` method, but instead defines the listener in the MXML tag for the custom component, similar to defining a listener for the Button “click” event:

```
<mx:Button label="Submit" click="myClickHandler(event);"/>
```

The Login.mxml component must have the following code to support defining an event listener for the custom component in the MXML tag:

```
<mx:Metadata>  
    [Event(name="success", type="com.myCompany.classes.events.LoginEvent")]  
</mx:Metadata>
```

And then an event listener can be defined for the custom event in the custom component MXML tag in the main application file:

```
<comp:Login id="loginComp" success="handleLogin(event);"/>
```

**Important: the event name in the custom component tag must match the value of the name="success" in the Event metadata tag. This same string must be used in the custom event class, as seen below:**

```
public static const SUCCESS:String = "success";  
super(LoginEvent.SUCCESS, true);
```

If the same string (in this case “**success**”) is not used in all these three areas, you will not catch the custom event. The choice to use the string “**success**” is arbitrary, and can be any string, according to your application design needs.

Because the Login component is not being used as a popup, but as a component in the main application, a number of code changes were necessary. The text inputs for username and password are reset each time the Login component appears. As a popup this can be achieved using the `creationComplete` event, but as a component in the main application, we need to use the “`show`” event. This also means that we no longer use

**PopUpManager.centerPopUp(this)** and **PopUpManager.removePopUp(this)**. Instead we use the main application **verticalAlign="middle"** and **horizontalAlign="center"** properties to center the Login component, and we use the **visible** property to show and hide the login form.

Note: when the Login component is used as a modal popup, the main application is blurred, and the user cannot interact with it while the popup is displayed. This behavior does not occur when the login component is part of the main app. Implementing this behavior is left as an exercise for the student.

**Note: compare the following code with that of the previous example to see what has changed.**

### CustomEvents2.mxml (main application file)

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:comp="components.*" verticalAlign="middle" horizontalAlign="center">
  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      import com.myCompany.classes.events.*;

      private function logInOut(evt:MouseEvent):void{
        if(evt.currentTarget.label == "Login"){
          loginComp.visible = true;
        }else{
          welcomeLbl.visible = false;
          loginBtn.label = "Login";
          loginComp.visible = true;
        }
      }

      private function handleLogin(evt:LoginEvent):void{
        welcomeLbl.visible = true;
        loginBtn.label = "Logout";
      }
    ]]>
  </mx:Script>
  <comp:Login id="loginComp" success="handleLogin(event);"/>
  <mx:ApplicationControlBar width="100%" dock="true">
    <mx:Canvas width="100%" height="100%">
      <mx:Label id="welcomeLbl" left="50" verticalCenter="0"
        fontSize="20" color="0xFFFFFFFF" text="Welcome !!!"
        visible="false"/>
      <mx:Label id="titleLbl" horizontalCenter="0" verticalCenter="0"
        text="Sample Application" fontSize="24"/>
      <mx:Button id="loginBtn" right="30" label="Login"
        verticalCenter="0" click="logInOut(event);"/>
    </mx:Canvas>
  </mx:ApplicationControlBar>
```

```
</mx:Application>
```

### **com/myCompany/classes/events/LoginEvent.as (custom event file)**

```
package com.myCompany.classes.events
{
    import flash.events.Event;

    public class LoginEvent extends Event{
        public static const SUCCESS:String = "success";

        public function LoginEvent(){
            super(LoginEvent.SUCCESS, true);
        }

        // Override the inherited clone() method.
        override public function clone():Event {
            return new LoginEvent();
        }
    }
}
```

### **components/Login.mxml (login component file)**

```
<?xml version="1.0" ?>
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"
    title="Login" horizontalScrollPolicy="off"
    verticalScrollPolicy="off" show="init();"
    defaultButton="{loginBtn}">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import com.myCompany.classes.events.*;
            import mx.collections.ArrayCollection;

            private var testUser:String = "user1";
            private var testPwd:String = "testme";

            private function init():void{
                userTxt.text = "";
                passwordTxt.text = "";
            }

            private function login(evt:Event):void{
                if(userTxt.text == testUser && passwordTxt.text == testPwd){
                    this.dispatchEvent(new LoginEvent());
                    this.visible = false;
                }else{
                    mx.controls.Alert.show("Login Failed!!!");
                }
            }
        ]]>
    </mx:Script>
    <mx:Metadata>
        [Event(name="success", type="com.myCompany.classes.events.LoginEvent")]
    </mx:Metadata>
    <mx:Form id="loginForm">
```

```

    <mx:FormItem label="Username:">
        <mx:TextInput id="userTxt"/>
    </mx:FormItem>
    <mx:FormItem label="Password:">
        <mx:TextInput id="passwordTxt"
            displayAsPassword="true"/>
    </mx:FormItem>
</mx:Form>
<mx:ControlBar horizontalAlign="center">
    <mx:Button label="Login" id="loginBtn" click="login(event);"/>
</mx:ControlBar>
</mx:Panel>

```

## Multiple Custom Events Example

The example code for this example involves code in these files:

- CustomEvents3.mxml (main application file)
- com/myCompany/classes/events/LoginEvent.as (custom event file)
- com/myCompany/classes/events/LoginFailureEvent.as (custom event file)
- com/myCompany/classes/events/LoginSuccessEvent.as (custom event file)
- components/Login.mxml (login component file)

### Points to Note about the Code

In this example, three custom event classes are defined:

LoginEvent.as (base custom event class)

LoginFailureEvent.as (custom event extending base class for login failure)

LoginSuccessEvent.as (custom event extending base class for login success)

One key difference in the code is in the Login.mxml component:

```

private function login(evt:Event):void{
    if(userTxt.text == testUser && passwordTxt.text == testPwd){
        this.dispatchEvent(new LoginSuccessEvent());
        PopUpManager.removePopUp(this);
    }else{
        this.dispatchEvent(new LoginFailureEvent());
        mx.controls.Alert.show("Login Failed!!!");
    }
}

```

If the user enters a valid username and password, the custom event **LoginSuccessEvent** is dispatched, otherwise the custom event **LoginFailureEvent** is dispatched.

The main application registers event listeners with the systemManager for these two event types:

```
systemManager.addEventListener(LoginEvent.SUCCESS, handleLogin, true);
systemManager.addEventListener(LoginEvent.FAILURE, handleLogin, true);
```

Notice that the addEventListener() method is provided constants of the **LoginEvent** class (LoginEvent.SUCCESS and LoginEvent.FAILURE) to indicate the event type, though alternatively the constants could be defined in the individual custom event sub-classes.

And the main application uses a single event handler, executing differently depending on the value of the dispatched event "type" property:

```
private function handleLogin(evt:LoginEvent):void{
    if(evt.type == LoginEvent.SUCCESS){
        welcomeLbl.text = "Welcome !!!";
        welcomeLbl.visible = true;
        loginBtn.label = "Logout";
    }else if(evt.type == LoginEvent.FAILURE){
        welcomeLbl.text = "LOGIN FAILED !!!";
        welcomeLbl.visible = true;
    }
}
```

**Note: compare the following code with that of the previous two examples to see what has changed.**

#### **CustomEvents3.mxml (main application file)**

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="init();">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.managers.PopUpManager;
            import components.*;
            import com.myCompany.classes.events.*;

            private function init():void{
                // Need to add the event listener to the systemManager, not to 'this'
                // (the application), because the event is dispatched from a popup,
                // and pop-ups and the application share the systemManager, but are
                // otherwise not in the same display list hierarchy.

                // Passing 'true' for the third parameter indicates that this event listener
```

```

        // listens for events in the 'capture' phase of event propagation.
        systemManager.addEventListener(LoginEvent.SUCCESS, handleLogin, true);
        systemManager.addEventListener(LoginEvent.FAILURE, handleLogin, true);
        login();
    }

    private function login():void{
        var loginComp:Login = new Login();
        PopUpManager.addPopUp(loginComp, this, true);
    }

    private function logInOut(evt:MouseEvent):void{
        if(evt.currentTarget.label == "Login"){
            login();
        }else{
            welcomeLbl.visible = false;
            loginBtn.label = "Login";
        }
    }

    private function handleLogin(evt:LoginEvent):void{
        if(evt.type == LoginEvent.SUCCESS){
            welcomeLbl.text = "Welcome !!!";
            welcomeLbl.visible = true;
            loginBtn.label = "Logout";
        }else if(evt.type == LoginEvent.FAILURE){
            welcomeLbl.text = "LOGIN FAILED !!!";
            welcomeLbl.visible = true;
        }
    }
}
]]>
</mx:Script>
<mx:ApplicationControlBar width="100%" dock="true">
    <mx:Canvas width="100%" height="100%">
        <mx:Label id="welcomeLbl" left="50" verticalCenter="0"
            fontSize="20" color="0xFFFFFFFF" text="Welcome !!!"
            visible="false"/>
        <mx:Label id="titleLbl" horizontalCenter="0" verticalCenter="0"
            text="Sample Application" fontSize="24"/>
        <mx:Button id="loginBtn" right="30" label="Login"
            verticalCenter="0" click="logInOut(event);"/>
    </mx:Canvas>
</mx:ApplicationControlBar>
</mx:Application>

```

### **com/myCompany/classes/events/LoginEvent.as (custom event file)**

```

package com.myCompany.classes.events
{
    import flash.events.Event;

```

```
/* This custom event is the base class from which other login related
 * events are derived. This makes it possible to dispatch a login related
 * event and set the loginEventType property, and then in the event handler,
 * check this property and execute based on the login event type.
 */
```

```
public class LoginEvent extends Event{
    public static const SUCCESS:String = "success";
    public static const FAILURE:String = "failure";

    public function LoginEvent(result:String){
        super(result, true);
    }
}
```

#### **com/myCompany/classes/events/LoginSuccessEvent.as (custom event file)**

```
package com.myCompany.classes.events
{
    import flash.events.Event;

    // This custom event should be dispatched if the user
    // successfully logs into the application.

    public class LoginSuccessEvent extends LoginEvent{
        protected static const LOGIN_SUCCESS:String = "success";

        public function LoginSuccessEvent(){
            super(LoginSuccessEvent.LOGIN_SUCCESS);
        }
    }
}
```

#### **com/myCompany/classes/events/LoginFailureEvent.as (custom event file)**

```
package com.myCompany.classes.events
{
    import flash.events.Event;

    // This custom event should be dispatched if the user fails to
    // successfully log into the application.

    public class LoginFailureEvent extends LoginEvent{
        protected static const LOGIN_FAILURE:String = "failure";

        public function LoginFailureEvent(){
            super(LoginFailureEvent.LOGIN_FAILURE);
        }
    }
}
```

#### **components/Login.mxml (login component file)**

```
<?xml version="1.0" ?>
```

```

<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"
  title="Login" horizontalScrollPolicy="off"
  verticalScrollPolicy="off" creationComplete="init();"
  defaultButton="{loginBtn}">
  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      import com.myCompany.classes.events.*;
      import mx.collections.ArrayCollection;
      import mx.managers.PopUpManager;

      private var testUser:String = "user1";
      private var testPwd:String = "testme";

      private function init():void{
        userTxt.text = "";
        passwordTxt.text = "";
        PopUpManager.centerPopUp(this);
      }

      private function login(evt:Event):void{
        if(userTxt.text == testUser && passwordTxt.text == testPwd){
          this.dispatchEvent(new LoginSuccessEvent());
          PopUpManager.removePopUp(this);
        }else{
          this.dispatchEvent(new LoginFailureEvent());
          mx.controls.Alert.show("Login Failed!!!");
        }
      }
    ]]>
  </mx:Script>
  <mx:Form id="loginForm">
    <mx:FormItem label="Username:">
      <mx:TextInput id="userTxt"/>
    </mx:FormItem>
    <mx:FormItem label="Password:">
      <mx:TextInput id="passwordTxt"
        displayAsPassword="true"/>
    </mx:FormItem>
  </mx:Form>
  <mx:ControlBar horizontalAlign="center">
    <mx:Button label="Login" id="loginBtn" click="login(event);"/>
  </mx:ControlBar>
</mx:Panel>

```