

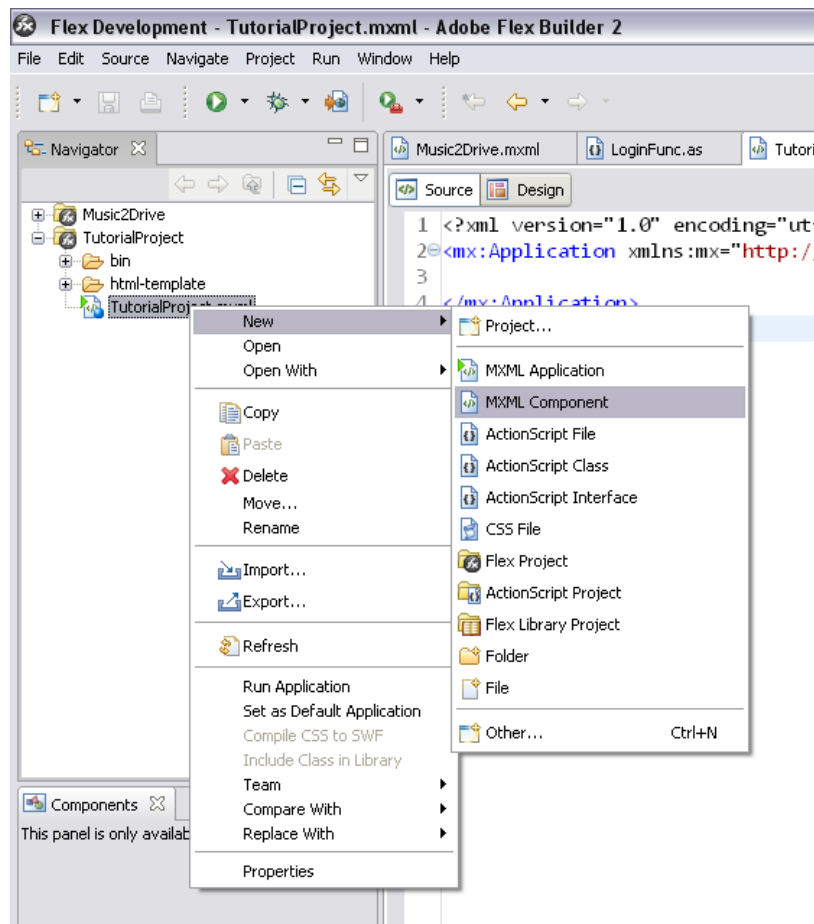
Flex 2 - Custom Components and Item Renderers

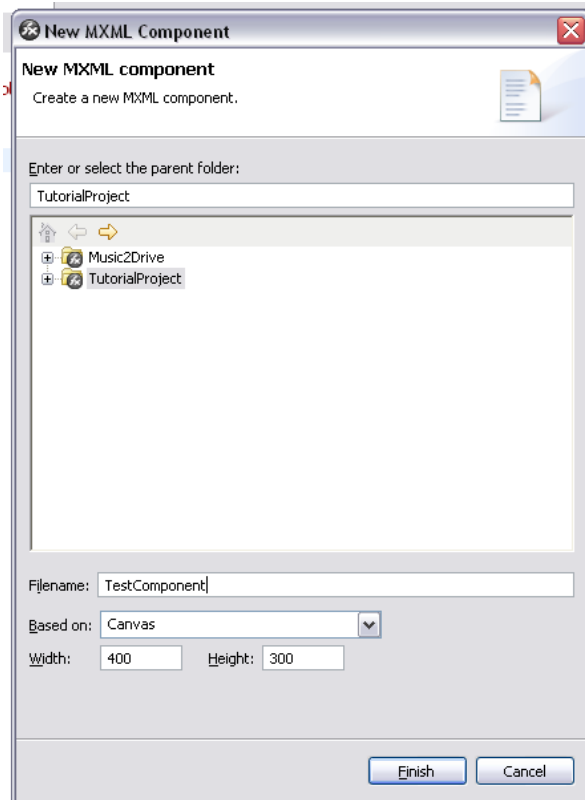
Part 1 - Custom Components

When coding in Flex there are many times where the GUI elements that they provide are just not enough. Adobe realized this and came up with a simple way to create custom components. These components can be used like Buttons and Datagrids, or as display objects, i.e. Item Renderers, within any element that is based off the List class. Yet, these concepts are among the most confusing to implement correctly for new users. Hopefully by reading this two part series on how to create Custom Components and Item Renderers new users to Flex will feel more at ease implementing these concepts in their own code.

First let's break down how Custom Components work. We will deal with Item Renderers later. When you are in **Design Mode** in your builder *right-click* on your project and go to:

New->MXML Component or go to File->New->MXML Component

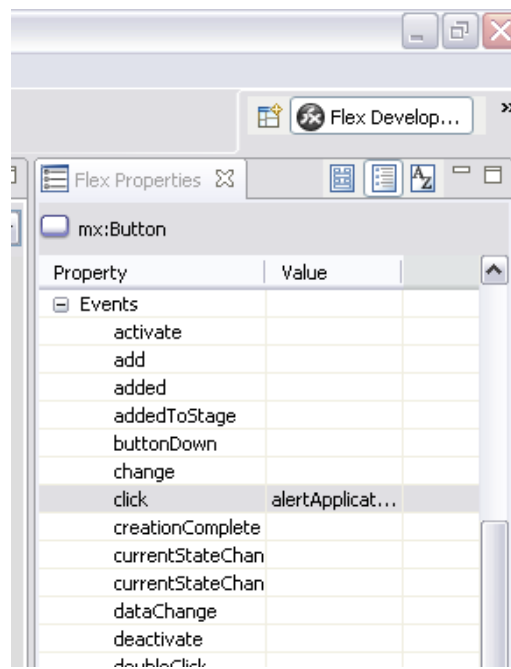




From here a window will appear this is where you will set the name of your new component class and what base class you are overriding, the default being a Canvas. Lets keep this first one simple and just call it **TestComponent** and leave the class being overridden as Canvas. When you click finish you will see that a new .mxml file has been added to your project source called **TestComponent.mxml**. You will also notice that under the components tab there will now be a new option in the Custom folder called TestComponent that you can add to your project like any other mxml GUI element.

Open the TestComponent.mxml file and take a look at the source code. You will see that there are two mxml tags, `<mx:Canvas ..>` and `</mx:Canvas>`. From here on all code that you add will go between these two tags. Go back to source view.

Now, lets add some GUI elements to the Canvas to help you understand how these things work. Drag two buttons onto the canvas. Make their labels Alert 1 and Alert 2. Set their click listeners to call the function `alertApplication(event)`; don't worry, we will be writing this next.



Just a quick check, if you switch to the source view of your component you should see something like this:

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

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">

<mx:Button x="10" y="10" label="Alert 1" click="alertApplication(event);"/>

<mx:Button x="10" y="40" label="Alert 2" click="alertApplication(event);"/>

</mx:Canvas>
```

Lets change the size of the canvas to a width of 100 and a height of 100 as well. Your top line should now read:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="100" height="100">
```

All good? Ok, so now that we have our buttons lets get that function in there. I have already written it for you below, please copy the *mx:Script* into your source code between the last *<mx:Button>* and *</mx:Canvas>* tags.

```
<mx:Script>

<![CDATA[

import mx.controls.Button

import mx.controls.Alert;

private function alertApplication(me:MouseEvent):void

{

//the button that called the Mouse Event CLICK

var tempButton:Button = new Button();

//Obtain the button that called CLICK from the Mouse Event

tempButton = me.target as Button;

//Show an Alert to confirm the click

Alert.show("Clicked on " + tempButton.label + " in component " + this.id, "Component Clicked", Alert.OK, this);

}

]]>
```

```
s</mx:Script>
```

Now your entire component should look like this in the source view:

```
<?xml version="1.0" encoding="utf-8"?>  
  
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">  
  
<mx:Button x="10" y="10" label="Alert 1" click="alertApplication(event);"/>  
  
<mx:Button x="10" y="40" label="Alert 2" click="alertApplication(event);"/>  
  
<mx:Script>  
  
<![CDATA[  
  
import mx.controls.Button;  
  
import mx.controls.Alert;  
  
private function alertApplication(me:MouseEvent):void  
  
{  
  
//the button that called the Mouse Event CLICK  
  
var tempButton:Button = new Button();  
  
//Obtain the button that called CLICK from the Mouse Event  
  
tempButton = me.target as Button;  
  
//Show an Alert to confirm the click  
  
Alert.show("Clicked on " + tempButton.label + " in component " + this.id, "Component Clicked", Alert.OK, this);  
  
}  
  
]]>  
  
</mx:Script>  
  
</mx:Canvas>
```

Now to actually use the component you have just created. Open the main mxml file of your project in design mode. In the components list should be a folder at the top called Custom containing a component called TestComponent. Click and drag two of these components onto your application. Give the first an ID of MyCom1 and the second an ID of MyCom2. Also note how in the application you cannot specifically select the two buttons contained within the component. Their properties can only be changed back in the TestComponent.mxml file. In the source view of your application you should see these two tags:

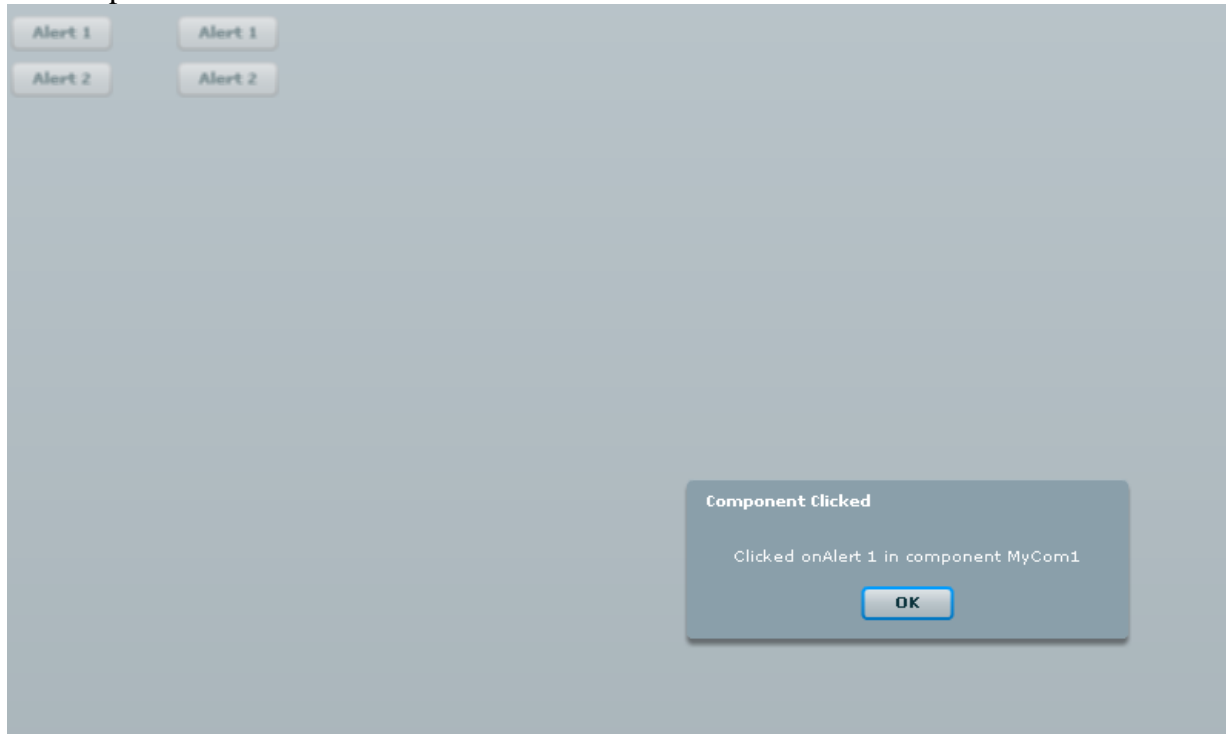
```
<ns1:TestComponent x="10" y="10" id="MyCom1">
```

```
</ns1:TestComponent>
```

```
<ns1:TestComponent x="10" y="118" id="MyCom2">
```

```
</ns1:TestComponent>
```

Give this code a run and click the buttons. If everything went right you should see Alert Windows popup every time you click a button telling you what button was clicked and the id of the component that the button is contained.



Some things you may want to keep in mind while using custom components.

1 – Once you have added a custom component there are two main ways to access code outside your component. The first is `parentDocument`. Calling `parentDocument` will give you access to the mxml file that contains the custom component. For example if you add a `customComponent1` to `customComponent2` and then add `customComponent2` to the main application, calling the `parentDocument` from `customComponent1` will only give you access to the public objects and data contained in `customComponent2`. Calling `parentDocument` from `customComponent2` will give you access to all that is public in the Application itself.

Calling `parentApplication` from any component at any level will always give you access to the main application file and all public objects and data that it contains.

In the code, the above calls would look something like this:

parentDocument.somePublicObject...

parentDocument.somePublicFunction();

parentApplication.somePublicObject...

parentApplication.somePublicFunction();

2 – Any component, custom or preexisting, can be used as an item renderer. The best way to think of how they will display in the code is that for every row contained in the List based code an item renderer will be created and displayed in that row containing all that row's information in the components data object. 3 – The tags for custom components in the mxml are not prefixed with a <mx, instead that are defaulted to <ns1. This ns1 stands for namespace1. If you were to look at the Application tag in your mxml tag after adding a custom component you would see that a new attribute has been set called xmlns:ns1="*".

So that is the skinny on custom components. I know that I am not a great technical writer, so please feel free to comment about any errors I may have stated, or maybe just helpful tips and tricks on creating these things. Next time I will go over Item Renderers, but I strongly urge you to become familiar with Custom Components first before diving into them since the manipulation of Custom Components is the core of dealing with Item Renderers.

QUICK NOTE ON EVENT SETTING: Adding event as a parameter to any event listener, E.G. myEventFunction(event), will automatically send the appropriately formatted event as a parameter to the function specified. If this event listener is declared in the MXML tags, <mx:Button ID="myButton" click="myEventFunction(event);"/>, of the program then it is not possible to remove the listener from the object at any point. However, if the listener is added in Actionscript, myButton.addEventListener(MouseEvent.CLICK, myEventFunction), then it can be removed later but the correctly formatted event object will automatically be sent as a parameter if done this way.