

AppSimple1 – simple application, does nothing

This application does nothing. It has the bluish/gray gradient background the all Flex applications default to. It is the standard default application Flex Builder generates for you when you create a new project, or when you create a new MXML application within a project.

In this case this is a new MXML application created within the SimpleApplications1 Flex project. Notice the SimpleApplications1.mxml file in the project. It is identical to AppSimple1.mxml except for the comments.

Notice also the only two properties set in this application's <mx:Application> tag, for the Flex namespace 'mx' and to set the layout for this application to 'absolute'.

In the next few applications, we will examine setting properties and style properties in MXML and in ActionScript, and will see the differences between properties and style properties in Flex.

AppSimple2 – two labels and two buttons

This application uses some simple Flex controls, specifically two labels and two buttons. Clicking one of the buttons increases the font size of the two labels. Clicking the other button decreases the label font size.

Launch the solution application (use the following method so you do not see the solution code)

- Right-click on AppSimple2Solution.mxml and select Run Application from the context menu.

Save the application file

- Immediately save AppSimple2.mxml as AppSimple2MyApp.mxml

Inside the opening <mx:Application> tag – set backgroundColor and backgroundGradientColors

- Set the backgroundColor style property to #00c6ff
- Set the backgroundGradientColors style property to [0x00c6ff, 0x00c6ff]

Within the opening and closing <mx:Application> tags – add a Label in MXML

- Add a <mx:Label> control and set properties and style properties:
 - id property to "srcLabel"
 - text property to "SOURCE"
 - Launch the application (Ctrl + F11)
 - left and top style properties to 10 and 10
 - fontSize style property to 20

Within the opening and closing <mx:Application> tags – add a Label in MXML

- Add a <mx:Label> control and set properties and style properties:
 - id property to "tgtLabel"
 - text property to "TARGET"
 - Refresh the application with F5.
 - left and top style properties to 200 and 10
 - fontSize style property to 20
 - Refresh the application with F5 and see the two labels.

Within the opening and closing <mx:Application> tags – add a Button in MXML

- Add a <mx:Button> control and set properties and style properties:
 - text property to "Decrease Font Size"
 - left and top style properties to 10 and 65
 - fontSize style property to 14
 - Refresh the application with F5.
 - click event property to inline ActionScript event listener code:

```
srcLabel.setStyle('fontSize', srcLabel.getStyle('fontSize')-5);tgtLabel.setStyle('fontSize', tgtLabel.getStyle('fontSize')-5);
```

Within the opening and closing <mx:Application> tags – add a Button in MXML

- Add a <mx:Button> control and set properties and style properties:
 - text property to "Increase Font Size"
 - left and top style properties to 10 and 100
 - fontSize style property to 14
 - Refresh the application with F5.
 - click event property to inline ActionScript event listener code:

```
srcLabel.setStyle('fontSize', srcLabel.getStyle('fontSize')+5);tgtLabel.setStyle('fontSize', tgtLabel.getStyle('fontSize')+5);
```
- Refresh the application with F5 and click the buttons a few times. The font size of the two labels should increase and decrease.

Inside the opening <mx:Label> tags – set up data binding

- Set up data binding within the tgtLabel <mx:Label> tag:
 - Change the text property of Label tgtLabel to {srcLabel.text} (don't forget the curly braces).
 - Refresh the application with F5 and notice the both labels have SOURCE as their text.

AppSimple3 – AppSimple2 created in ActionScript

This application recreates the AppSimple2 application in ActionScript.

Launch the solution application (use the following method so you do not see the solution code)

- Right-click on AppSimple3Solution.mxml and select Run Application from the context menu.

Save the application file

- Immediately save AppSimple3.mxml as AppSimple3MyApp.mxml.

Inside the opening <mx:Application> tag – set creationComplete to init()

- Set the creationComplete event property to the function init().
- You may get errors because the init() function is not yet defined.

Within the opening and closing <mx:Application> tags – add a Script tag

- Add a <mx:Script> tag.
- If Flex Builder does not add the <![CDATA[]]> block in your <mx:Script> tag, copy the following to your application file. All ActionScript outside MXML tags must be within the <![CDATA[]]> block.

```
<mx:Script>
<![CDATA[

]]>
</mx:Script>
```

Inside the <mx:Script> tag CDATA block – define the init() function to set style properties

- Create a private function init() taking no parameters with void return type, and add two statements:
 - Use the setStyle() method to set the backgroundColor style property to #00c6ff.
setStyle("backgroundColor", "#00c6ff");
 - Use the setStyle() method to set the backgroundGradientColors style property to [0x00c6ff, 0x00c6ff].
setStyle("backgroundGradientColors", [0x00c6ff, 0x00c6ff]);

Notice the default Flex background appears first, so better to set the backgroundColor and backgroundGradientColors within the opening <mx:Application> tag.

Inside the <mx:Script> tag CDATA block – create two Labels in ActionScript and add to display list

- Create two private variables named "srcLabel" and "tgtLabel" outside the init() method so they will be global, with their data type set to Label, then set properties and style properties.
 - srcLabel Label:
 - text property to "SOURCE"
 - left and top style properties to 10 and 10
 - fontSize style property to 20
 - tgtLabel Label:
 - text property to "TARGET"
 - left and top style properties to 200 and 10
 - fontSize style property to 20
 - Launch the application (Ctrl + F11), and notice the labels do not display.
 - Add the labels to the display list using addChild().
this.addChild(srcLabel);
this.addChild(tgtLabel);
 - Refresh the application with F5, and notice the labels do display.

Inside the <mx:Script> tag CDATA block – create two Buttons in ActionScript and add to display list

- Create two private variables named "decBtn" and "incBtn" outside the init() method, with their data type set to Button, then set properties and style properties.
 - decBtn Button:
 - label property to "Decrease Font Size"
 - left and top style properties to 10 and 100
 - fontSize style property to 14
 - Use addEventListener() to set the MouseEvent.CLICK event property to function clickFunc.
decBtn.addEventListener(MouseEvent.CLICK, clickFunc);
 - incBtn Button:
 - label property to " Increase Font Size"
 - left and top style properties to 65 and 100
 - fontSize style property to 14
 - Use addEventListener() to set the MouseEvent.CLICK event property to function clickFunc.
incBtn.addEventListener(MouseEvent.CLICK, clickFunc);
 - Add the buttons to the display list using addChild().
this.addChild(decBtn);
this.addChild(incBtn);
 - The application may have errors because the event handlers are not defined yet.

Inside the <mx:Script> tag CDATA block – define Button click listener function

- Create a private function clickFunc taking a parameter named evt of type MouseEvent returning void, and in the function do the following:
 - Create a local var named btn of type Button, casting using evt.currentTarget as Button.
var btn:Button = evt.currentTarget as Button;
 - Check the label of btn in an if statement, and if the label is “Decrease Font Size” do this:
srcLabel.setStyle('fontSize', srcLabel.getStyle('fontSize')-5);tgtLabel.setStyle('fontSize',
tgtLabel.getStyle('fontSize')-5);
 - Else do this:
srcLabel.setStyle('fontSize', srcLabel.getStyle('fontSize')+5);tgtLabel.setStyle('fontSize',
tgtLabel.getStyle('fontSize')+5);
- Refresh the application with F5 and click the buttons a few times.

Within the opening and closing <mx:Application> tags (just after the closing </mx:Script> tag)

- Set up data binding in ActionScript, making label srcLabel bindable:
 - Add this MXML tag after the <mx:Script> tag:
<mx:Binding source="srcLabel.text" destination="tgtLabel.text"/>
 - Refresh the application with F5 and notice the both labels have SOURCE as their text.

NOTICE THERE IS LOTS MORE CODE IN ACTIONSCRIPT - USE MXML FOR UI WHENEVER POSSIBLE

AppSimple4

This app examines a few more Flex controls, and using data in Flex.

Launch the solution application (use the following method so you do not see the solution code)

- Right-click on AppSimple4Solution.mxml and select Run Application from the context menu.

Save the application file

- Immediately save AppSimple4.mxml as AppSimple4MyApp.mxml

Inside the opening <mx:Application> tag – set creationComplete to init()

- Set the creationComplete event property to the function init().
- You may get errors because the init() function is not yet defined.

Within the opening and closing <mx:Application> tags – add a Script tag

- Add a <mx:Script> tag.
- If Flex Builder does not add the <![CDATA[]]> block in your <mx:Script> tag, copy the following to your application file. All ActionScript outside MXML tags must be within the <![CDATA[]]> block.

```
<mx:Script>
<![CDATA[

]]>
</mx:Script>
```

Inside the <mx:Script> tag CDATA block – include the data file

- Add this include statement to “include” the data file:
include data/actionScriptData/simpleData1.as;
- Examine that data file and it’s ArrayCollection and XML.

Within the opening and closing <mx:Application> tags – add a Label in MXML

- Add a <mx:Label> control and set properties and style properties:
 - text property to "US State Population Density"
 - color style property to 0x0000FF
 - top style property to 10
 - horizontalCenter style property to 0
 - fontFamily style property to "Arial"
 - fontSize style property to 20
 - fontWeight style property to bold

Within the opening and closing <mx:Application> tags – add a Label in MXML

- Add another <mx:Label> control and set properties and style properties:
 - text property to "Filter by density:"
 - top style property to 100
 - horizontalCenter style property to -90
 - fontFamily style property to "Arial"
 - Refresh the application with F5.

Within the opening and closing <mx:Application> tags – add a ComboBox in MXML

- Add a <mx:ComboBox> control and set properties and style properties:
 - id property to "filterCbx"
 - dataProvider property to "{filterAC}"
 - change property to "filterByDensity(event)"
 - rowCount property to "{filterAC.length}"
 - horizontalCenter style property to 15
 - top style property to 100
 - You may get errors because the filterByDensity change handler function is not defined yet.

Inside data file data/actionScriptData/simpleData1.as – make filterAC bindable

- Open data file **data/actionScriptData/simpleData1.as** and make filterAC bindable by adding [Bindable] at the start of the line that begins with "**private var filterAC**". Also remove any commas from the density field for New Jersey and Rhode Island, as commas in the number cause filtering to malfunction.
- The warnings "Data binding will not be able to detect assignments to filterAC" should go away.

Within the opening and closing <mx:Application> tags – add a DataGrid in MXML

- Add a <mx>DataGrid> control and set properties and style properties:
 - id property to "statesDG"
 - dataProvider property to "{dgXLC}"
 - rowCount property to "{dgXLC.length>15?15:dgXLC.length}" What does it do?
 - top style property to 130
 - horizontalCenter style property to 0

Within the opening and closing <mx>DataGrid> tags – add columns in the DataGrid in MXML

- Add a <mx>DataGridColumn> tag and set properties and style properties.
 - headerText style property to "State"
 - dataField property to "name"
 - width property to 120
- Add a <mx>DataGridColumn> tag and set properties and style properties.
 - headerText style property to "Rank"
 - dataField property to "rank"
 - width property to 50
- Add a <mx>DataGridColumn> tag and set properties and style properties.
 - headerText style property to "Density Sq/Mi"
 - dataField property to "density"
 - width property to 110
 - You may get errors as we have not yet defined the DataGrid dataProvider.

Inside the <mx:Script> tag CDATA block – create an XMLListCollection object as DataGrid dataProvider

- After the include statement for the data file, define a private bindable variable dgXLC of type XMLListCollection as the dataProvider for the datagrid, using e4x syntax when instantiating the XMLListCollection to bring in all “state” elements from the data file “statesInfo1” XML object.
[Bindable] private var dgXLC:XMLListCollection = new XMLListCollection(statesInfo1..state);
- Refresh the application with F5, and observe the DataGrid displays and has data.

Inside the <mx:Script> tag CDATA block – create ComboBox change event handler function

- Define the filterByDensity(event) change handler function.
 - The function takes a single ListEvent parameter.
 - The function return type is void.
 - Within the function body, create a local variable.
 - Variable name will be “cbx”.
 - Variable data type will be ComboBox.
 - Set the variable value to the event object currentTarget cast using the “as” keyword:
var cbx:ComboBox = evt.currentTarget as ComboBox;
 - Within the function body, create a local variable.
 - Variable name will be “filterVal”
 - Variable data type will be Number.
 - Set the variable value to cbx.selectedItem.data using an explicit cast to a Number:
var filterVal:Number = Number(cbx.selectedItem.data);
 - Reset the datagrid dataProvider dgXLC to filter data based on the ComboBox selected item data:
dgXLC = new XMLListCollection(statesInfo1.state.(density > filterVal));
 - Refresh the application with F5, change the value in the ComboBox, and observe the filtering.

AppSimple5

This app examines a constraint based auto-resizing and several more Flex 3 controls.

Launch the solution application (use the following method so you do not see the solution code)

- Right-click on AppSimple5Solution.mxml and select Run Application from the context menu.

Save the application file

- Immediately save AppSimple5.mxml as AppSimple5MyApp.mxml

Within the opening and closing <mx:Application> tags – add an ApplicationControlBar in MXML

- Add a <mx:ApplicationControlBar> to hold some controls and set properties and style properties:
 - dock property to true
 - left style property to 10
 - right style property to 10
 - horizontalAlign style property to “center”

Within the opening and closing <mx:ApplicationControlBar> tags – add a Canvas in MXML

- Add a <mx:Canvas> tag and set the width property to 100%. The Canvas will allow us to achieve constraint based layout, as the ApplicationControlBar is auto-layout by default.

Within the opening and closing <mx:Canvas> tags – add an Image in MXML

- Add a <mx:Image> control and set properties and style properties:
 - id property to “incFont”
 - source property to: “@Embed(source='assets/images/incFont.PNG')”
 - left style properties to 50
 - verticalCenter style property to 0
 - click property to “changeFontSize(event)”
 - toolTip style property to “Increase Font Size”

Within the opening and closing <mx:Canvas> tags – add an Image in MXML

- Add a <mx:Image> control and set properties and style properties:
 - id property to "decFont"
 - source property to: "@Embed(source='assets/images/decFont.PNG')"
 - left style properties to 100
 - verticalCenter style property to 0
 - click property to "changeFontSize(event)"
 - toolTip style property to "Decrease Font Size"

Within the opening and closing <mx:Canvas> tags – add a Label in MXML

- Add a <mx:Label> control and set properties and style properties:
 - text property to "Word Master Text Editor"
 - fontFamily style property to "Arial"
 - fontSize style property to 20
 - fontWeight style property to bold
 - horizontalCenter style property to 0
 - verticalCenter style property to 0
 - color style property to "#0000FF"
 - Refresh the application with F5 and examine the application.

Within the opening and closing <mx:Application> tags – add a TextArea in MXML

- Add a <mx:TextArea> control and set properties and style properties:
 - id property to "txtArea"
 - left style property to 10
 - right style property to 10
 - top style property to 10
 - bottom style property to 40
 - Refresh the application with F5 and notice the TextArea.

Within the opening and closing <mx:Application> tags – add a Button in MXML

- Add a <mx:Button> control and set properties and style properties:
 - label property to "Save"
 - bottom style property to 10
 - horizontalCenter style property to -50
 - fontSize style property to 12
 - click property to "saveText();"

Within the opening and closing <mx:Application> tags – add a Button in MXML

- Add a <mx:Button> control and set properties and style properties:
 - label property to "Clear Text"
 - bottom style property to 10
 - horizontalCenter style property to 50
 - fontSize style property to 12
 - click property to "clearText();"
 - You may get errors because the button click handler functions have not yet been defined.

Within the opening and closing <mx:Application> tags – add a Script tag

- Add a <mx:Script> tag.
- If Flex Builder does not add the <![CDATA[]]> block in your <mx:Script> tag, copy the following to your application file. All ActionScript outside MXML tags must be within the <![CDATA[]]> block.

```
<mx:Script>
<![CDATA[

]]>
</mx:Script>
```

Inside the <mx:Script> tag CDATA block – embed and image for an Alert

- Add this code to embed an image:
[Embed(source="assets/images/exclaimIcon.png")]
[Bindable] public var exclaimIconClass:Class;

Inside the <mx:Script> tag CDATA block – create Button click event handler function

- After the opening <![CDATA[code within in the <mx:Script> tag, add this code to implement the changeFontSize() method:

```
private function changeFontSize(evt:MouseEvent):void{
    if(evt.currentTarget.id == "incFont"){
        txtArea.setStyle("fontSize", txtArea.getStyle("fontSize")+1);
    } else{
        txtArea.setStyle("fontSize", txtArea.getStyle("fontSize")-1);
    }
}
```

Inside the <mx:Script> tag CDATA block – create Button click event handler function

- After the opening <![CDATA[code within in the <mx:Script> tag, add this code to implement the saveText() method:

```
private function saveText():void{
    mx.controls.Alert.show("Would you like to save this document?", "Save Document",
        Alert.YES | Alert.NO, this, closeSaveHandler, null, Alert.YES);
}
```

Inside the <mx:Script> tag CDATA block – create Alert close event handler function

- After the opening <![CDATA[code within in the <mx:Script> tag, add this code to implement the closeSaveHandler() method:

```
private function closeSaveHandler(evt:CloseEvent):void{
    if(evt.detail == Alert.YES){
        mx.controls.Alert.show("Implement functionality to save to web (can't save to local machine).");
    }
}
```

Inside the <mx:Script> tag CDATA block – create Button click event handler function

- After the opening <![CDATA[code within in the <mx:Script> tag, add this code to implement the clearText() method:

```
private function clearText():void{
    mx.controls.Alert.show("Are you sure you would like to clear this document?", "Clear Text Warning",
        Alert.YES | Alert.NO, this, clearTextHandler, exclaimIconClass);
}
```

Inside the <mx:Script> tag CDATA block – create Alert close event handler function

- After the opening <![CDATA[code within in the <mx:Script> tag, add this code to implement the clearTextHandler() method:

```
private function clearTextHandler(evt:CloseEvent):void{
    if(evt.detail == Alert.YES){
        txtArea.text = "";
    }
}
```
- Refresh the application with F5. Type some text in the TextArea, click the buttons to increase and decrease the font size, and click the buttons to save and clear the text (saving the text is not implemented).

AppFlexText

This app requires no changes on your part. It simply presents information on some Flex text-based controls.

Launch the application and examine the text controls

- Launch the application (Ctrl + F11).
- Examine the text controls.

| Control | Multi-line | Allows user to change text |
|----------------|-------------------|-----------------------------------|
| Label | No | No |
| TextInput | No | Yes |
| Text | Yes | No |
| TextArea | Yes | Yes |
| RichTextEditor | Yes | Yes |

AppHtmlText

This app requires no changes on your part. It presents information on using HTML tags in Flex text controls.

Launch the application and examine the use of HTML in the text controls

- Launch the application (Ctrl + F11).
- Examine the HTML in the text controls.

The Flex text controls allow you to include HTML tags in their text using the `htmlText` property. You use the `htmlText` property to set or get an HTML-formatted text string. You can also use one tag that is not part of standard HTML, the `textFormat` tag. The following link provides details on supported tags and attributes:

[Using tags in HTML text](#)

You can also specify text formatting by using Flex styles. You can set a base style, such as the font characteristics or the text weight, by using a style, and override the base style in sections of your text by using tags, such as the `` tag. In the following example, the `<mx:Text>` tag styles specify blue, italic, 14 point text, and the `<mx:htmlText>` tag includes HTML tags that override the color and point size.

These HTML tags are supported in Flex 3:

| | | |
|---|--|--|
| Anchor tag (<code><a></code>) | Bold tag (<code></code>) | Break tag (<code> </code>) |
| Font tag (<code></code>) | Image tag (<code></code>) | Italic tag (<code><i></code>) |
| List item tag (<code></code>) | Paragraph tag (<code><p></code>) | Underline tag (<code><u></code>) |
| Text format tag (<code><textformat></code>) | | |
| <code>blockindent</code> | <code>leftmargin</code> | <code>rightmargin</code> |
| <code>indent</code> | <code>leading</code> | <code>tabstops</code> |

AppImagesInFlex

This app requires no changes on your part. It presents information on Flex controls for loading graphics.

Launch the application and examine the use of images

- Launch the application (Ctrl + F11).
- Examine the images displayed in the application.
- Click the button to swap the blue circle and red square in the `<mx:SWFLoader>`. The images are SWF files created in Adobe Flash CS4.
- Click the buttons to hide and show the small seaside landscape image. They illustrate the fact that when hiding a control by setting the "visible" property to false, also set the "includeInLayout" property to false, otherwise Flex will continue to reserve space in the layout for the control, even if it is not visible.

The `<mx:Image>` control can be used for importing GIF, JPEG, PNG, SWF, at runtime and compile time, and GIF, JPEG, PNG, SWF, SVG files at compile time.

The `<mx:SWFLoader>` control can be used for loading Flex applications or SWF within Flex applications.

Image extends SWFLoader, so you can use the `load()` method with images to load them at runtime.

You can hide images by setting both the "visible" AND "includeInLayout" properties to false.

You typically use `<mx:Image>` for static graphics/SWF, and `<mx:SWFLoader>` for loading Flex apps.

Embedded images load immediately, as they are compiled into the app SWF, but they add to the size of the application SWF, and slow down app initialization.

Non-embedded images load at run time from the local file system where SWF file runs, or via remote images over the network. The referenced images add no overhead to an app's loading time, but may result in a delay when compared to embedded images, as they have to be loaded.

SWF files can only access either local or network resources, cannot access both.

Compiler option `-use-network` set to false access local filesystem, but not network. If set to true access network, but not local filesystem.

Can reference images using a URL, but default security settings only permit Flex applications to access resources on same domain as the application.

To access images on other servers, use a `crossdomain.xml` file.

AppAutoLayout1

This app simply displays several images in a horizontal row using auto-layout in the container `<mx:Application>`. You will modify a property so the images display vertical rather than horizontal.

Save the application file

- Immediately save `AppAutoLayout1.mxml` as `AppAutoLayout1MyApp.mxml`

Initially launch the application and examine the app

- Launch the application (Ctrl + F11), notice the layout of the images is horizontal.

Inside the opening `<mx:Application>` tag – set a property

- `layout` property to vertical
- Refresh the application with F5 and notice the layout of the images is now vertical.

AppAutoLayout2

This app displays several images in rows and columns. The app is designed to illustrate the use of style properties controlling the gaps between controls, and the padding around the edges of containers.

Save the application file

- Immediately save AppAutoLayout2.mxml as AppAutoLayout2MyApp.mxml

Initially launch the application and examine the app

- Launch the application (Ctrl + F11) and notice how the application displays, particularly the space between and around the rectangles.

Inside <mx:Application> tag – set layout property to horizontal and refresh app to see changes.

Inside <mx:Application> tag – set layout property to vertical and refresh app to see changes.

Inside <mx:Application> tag – set horizontalGap style property to 30 and refresh app to see changes.

Inside <mx:Application> tag – set verticalGap style property to 30 and refresh app to see changes.

Inside <mx:Application> tag – set horizontalGap and verticalGap to 0 and refresh app to see changes.

Inside ALL the opening <mx:HBox> tags – add and change a style property

- Add the horizontalGap style property, set it to 30, and refresh with F5 to see changes.
- Change the horizontalGap style properties to 0, and refresh with F5 to see changes.

Inside the opening <mx:Application> tag – add some style properties

- paddingBottom set to 0
- paddingLeft set to 0
- paddingRight set to 0
- paddingTop set to 0
- Refresh the application with F5 to see changes.

AppAutoLayout3

This application illustrates using the Form container. It also shows one type of basic validation, and it shows using the CheckBox control.

Launch the solution application (use the following method so you do not see the solution code)

- Right-click on AppAutoLayout3Solution.mxml and select Run Application from the context menu.

Save the application file

- Immediately save AppAutoLayout3.mxml as AppAutoLayout3MyApp.mxml

Inside the opening <mx:Application> tag – set layout to vertical

- set the layout property to vertical.

Within the opening and closing <mx:Application> tags – add a Label in MXML

- Add a <mx:Label> control and set properties and style properties:
 - text property to "ProWeb Internet Services"
 - fontSize style property to 20
 - color style property to 0xAA00AA
 - fontWeight style property to bold

Within the opening and closing <mx:Application> tags – add a Label in MXML

- Add a <mx:Label> control and set properties and style properties:
 - text property to "New Customer Registration"
 - fontSize style property to 14
 - color style property to 0x0000FF
 - fontWeight style property to bold

Within the opening and closing <mx:Application> tags – add a Form in MXML

- Add a <mx:Form> control.

Within the opening and closing <mx:Form> tags – add a FormHeading in MXML

- Add a <mx:FormHeading> control and set this property:
 - label property to "Customer Information"

Within the opening and closing <mx:Form> tags – add FormItem tags in MXML

- While adding the following controls you may get errors as the event handler functions are not yet defined.
- Add a <mx:FormItem> control and set this property:
 - label property to "Name:"
 - Add a <mx:TextInput> control within the <mx:FormItem> opening and closing tags, and set properties.
 - id property to "custName"
 - change property to "validateInfo()"
- Add a <mx:FormItem> control and set this property:
 - label property to "Address1:"
 - Add a <mx:TextInput> control within the <mx:FormItem> opening and closing tags, and set properties.
 - id property to "addr1"
 - change property to "validateInfo()"
- Add a <mx:FormItem> control and set this property:
 - label property to "Address2:"
 - Add a <mx:TextInput> control within the <mx:FormItem> opening and closing tags, and set properties.
 - id property to "addr2"
 - change property to "validateInfo()"
- Add a <mx:FormItem> control and set this property:
 - label property to "City:"
 - Add a <mx:TextInput> control within the <mx:FormItem> opening and closing tags, and set properties.
 - id property to "city"
 - change property to "validateInfo()"
- Add a <mx:FormItem> control and set this property:
 - label property to "State:"
 - Add a <mx:TextInput> control within the <mx:FormItem> opening and closing tags, and set properties.
 - id property to "state"
 - change property to "validateInfo()"
- Add a <mx:FormItem> control and set this property:
 - label property to "Zip:"
 - Add a <mx:TextInput> control within the <mx:FormItem> opening and closing tags, and set properties.
 - id property to "zip"
 - change property to "validateInfo()"
- Add a <mx:FormItem> control and set this property:
 - label property to "Phone:"
 - Add a <mx:TextInput> control within the <mx:FormItem> opening and closing tags, and set properties.
 - id property to "phone"
 - change property to "validateInfo()"
- Add a <mx:FormItem> control and set this property:
 - label property to "Email:"
 - Add a <mx:TextInput> control within the <mx:FormItem> opening and closing tags, and set properties.
 - id property to "email"
 - change property to "validateInfo()"
- Add a <mx:FormItem> control and add a CheckBox within the FormItem:
 - Add a <mx:CheckBox> control within the <mx:FormItem> opening and closing tags, and set properties.
 - id property to "promotions"
 - change property to "promotionsHandler()"
 - label property to "Send information on special promotions."

- Add a <mx:FormItem> control and add a CheckBox within the FormItem:
 - Add a <mx:CheckBox> control within the <mx:FormItem> opening and closing tags, and set properties.
 - id property to "newsletter"
 - change property to "newsletterHandler()"
 - label property to "Sign me up for the ProWeb monthly newsletter."
- Add a <mx:FormItem> control and add a Button within the FormItem:
 - Add a <mx:Button> control within the <mx:FormItem> opening and closing tags, and set properties.
 - label property to "Register"
 - enabled property to "{allDataFilled}"
 - disabledColor style property to 0xCCCCCC
 - click property to "registerUser()"

Within the opening and closing <mx:Application> tags – add a Script tag

- Add a <mx:Script> tag.
- If Flex Builder does not add the <![CDATA[]]> block in your <mx:Script> tag, copy the following to your application file. All ActionScript outside MXML tags must be within the <![CDATA[]]> block.

```
<mx:Script>
<![CDATA[

]]>
</mx:Script>
```

Inside the <mx:Script> tag CDATA block – create a Boolean variable to track if all data fields are filled

- Create a private variable named "allDataFilled, with a data type of Boolean.

Inside the <mx:Script> tag CDATA block – implement TextInput change event handler function

- Add this code to implement the validateInfo() method:

```
private function validateInfo():void{
    if(custName.text!="" &&
        addr1.text!="" &&
        addr2.text!=""
        && city.text!=""
        && state.text!=""
        && zip.text!=""
        && phone.text!=""
        && email.text!=""){
        allDataFilled = true;
    }
}
```

Inside the <mx:Script> tag CDATA block – implement CheckBox change event handler function

- Add this code to implement the promotionsHandler() method:

```
private function promotionsHandler(event:Event):void{
    if(event.currentTarget.selected){
        mx.controls.Alert.show("Promotions change weekly.");
    }else{
        mx.controls.Alert.show("You can signup for promotions anytime.");
    }
}
```

Inside the <mx:Script> tag CDATA block – implement CheckBox change event handler function

- Add this code to implement the newsletterHandler() method:

```
private function newsletterHandler(event:Event):void{
    if(event.currentTarget.selected){
        mx.controls.Alert.show("Our newsletter has great tips.");
    }else{
        mx.controls.Alert.show("You can signup for our newsletter anytime.");
    }
}
```

Inside the <mx:Script> tag CDATA block – implement Button click event handler function

- Add this code to implement the registerUser() method:

```
private function registerUser():void{
    var optionsString:String = "";
    if(promotions.selected){
        optionsString += "You signed up for promotions.\n";
    }
    if(newsletter.selected){
        optionsString += "You signed up for our newsletter.";
    }
    mx.controls.Alert.show("You are now a registered user !!!\n' + optionsString);
}
```
- Refresh the application with F5. Enter some data in some and all the TextInput fields and notice that the "Register" button does not become enabled until all the TextInput fields have data. Check one or both of the CheckBox and click the Register button.

AppAutoLayout4

In this application you create the same layout side-by-side, one implemented using auto-layout and the other implemented using absolute layout.

This is the first application where you are not provided with all the properties and style properties required to control layout, and you may need to add some properties and/or style properties to achieve the desired layouts.

You will be creating two Panels as outlined below, positioned side-by-side, one using auto-layout, one using absolute layout, as seen in this screenshot `assets\images\screenshots\AppAutoLayout4Solution.PNG`

Launch the solution application (use the following method so you do not see the solution code)

- Right-click on AppAutoLayout4Solution.mxml and select Run Application from the context menu.

Save the application file

- Immediately save AppAutoLayout4.mxml as AppAutoLayout4MyApp.mxml

Inside the opening <mx:Application> tag – set creationComplete to init()

- Set the creationComplete event property to the function init().
- Use inline ActionScript in the creationComplete="init()" to popup an Alert control:
 - Alert title: "SeaSide Galleries"
 - Alert text: "Welcome to SeaSide Galleries"

Within the opening and closing <mx:Application> tags – add a Panel in MXML

- Add a <mx:Panel> control and set properties:
 - title property to "SeaSide Galleries - Auto-Layout"
 - layout property to vertical
 - width property to 395
 - height property to 250
 - Refresh the application with F5 and examine the Panel. An Alert should popup when app is launched

Within the opening and closing <mx:Panel> tags – add an HBox in MXML

- Add an <mx:HBox> control.

Within the opening and closing <mx:HBox> tags – add an Image in MXML

- Add an <mx:Image> control and set properties.
 - source property to “assets/images/BigSur1.png”
 - width property to 200
 - height property to 150
 - Refresh the application with F5 and examine the Panel. The image should display in the Panel.

Within the opening and closing <mx:Panel> tags – add a VBox in MXML

- Add a <mx:VBox> control.

Within the opening and closing <mx:VBox> tags – add a Text in MXML

- Add a <mx:Text> control and set properties.
 - text property to "This photo beautifully captures the wonder of the big sur coastline."

Within the opening and closing <mx:VBox> tags – add a Label in MXML

- Add a <mx:Label> control and set properties.
 - text property to " Price: \$49.00"

Within the opening and closing <mx:Panel> tags – add a ControlBar in MXML

- Add a <mx:ControlBar> control:

Within the opening and closing <mx:ControlBar> tags – add a Label in MXML

- Add a <mx:Label> control and set properties.
 - text property to “Quantity”

Within the opening and closing <mx:ControlBar> tags – add a NumericStepper in MXML

- Add a <mx:NumericStepper> control and set properties.
 - id property to “photoHS1”

Within the opening and closing <mx:ControlBar> tags – add a Spacer in MXML

- Add a <mx:Spacer> control and set properties.
 - width property to “100%” (spacer will expand, fill the empty space, pushing button to right)

Within the opening and closing <mx:ControlBar> tags – add a Button in MXML

- Add a <mx:Button> control and set properties.
 - label property to “Add to Cart”
 - click property to “addToCart1()”
 - enabled property to “{Boolean(photoHS1.value)}”
 - You may have errors because the button click event handler function is not yet implemented.

Within the opening and closing <mx:Panel> tags – add a Script tag

- Add a <mx:Script> tag.
- If Flex Builder does not add the <![CDATA[]]> block in your <mx:Script> tag, copy the following to your application file. All ActionScript outside MXML tags must be within the <![CDATA[]]> block.

```
<mx:Script>
<![CDATA[

]]>
</mx:Script>
```

Inside the <mx:Script> tag CDATA block – implement Button click event handler function

- Add this code to implement the addToCart1() method:

```
private function addToCart1():void {
    var cartStr:String = "Added ";
    if(photoHS1.value > 1){
        cartStr += photoHS1.value + " photos";
    }else{
        cartStr += photoHS1.value + " photo";
    }
    cartStr += " to the cart.";
    mx.controls.Alert.show(cartStr);
}
```
- Refresh the application with F5 and examine the Panel. All controls should be fully functional.

Within the opening and closing <mx:Application> tags – add another Panel in MXML

- Add a <mx:Panel> control and set properties:
 - title property to "SeaSide Galleries – Absolute Layout"
 - layout property to absolute
 - width property to 395
 - height property to 250
 - Refresh the application with F5 and examine the Panel.

Within the opening and closing <mx:Panel> tags – add an Image in MXML

- Add an <mx:Image> control, and set properties:
 - id property to "pic"
 - source property to "assets/images/BigSur1.png"
 - width property to 200
 - height property to 150
 - Set the x and y properties to values that will position the image as seen in the screenshot:
assets\images\screenshots\AppAutoLayout4Solution.PNG

Within the opening and closing <mx:Panel> tags – add a Text in MXML

- Add a <mx:Text> control, and set properties:
 - id property to "desc"
 - text property to "This photo beautifully captures the wonder of the big sur coastline."
 - verticalCenter style property to a value that will position the text as seen in the screenshot:
assets\images\screenshots\AppAutoLayout4Solution.PNG
 - x property calculated using the "pic" image "width" property

Within the opening and closing <mx:Panel> tags – add a Label in MXML

- Add a <mx:Label> control, and set properties:
 - text property to "Price: \$49.00"
 - x property calculated using the "pic" image "width" property
 - y property calculated using the "desc" text "y" and "height" properties

Within the opening and closing <mx:Panel> tags – add a ControlBar in MXML

- Add a <mx:ControlBar>.

Within the opening and closing <mx:ControlBar> tags – add a Label in MXML

- Add a <mx:Label> control, and set properties:
 - text property to "Quantity"

Within the opening and closing <mx:ControlBar> tags – add a NumericStepper in MXML

- Add a <mx:NumericStepper>, and set properties:
 - id property to "photoHS2"

Within the opening and closing <mx:ControlBar> tags – add a Button in MXML

- Add a <mx:Button>, and set properties:
 - label property to "Add to Cart"
 - click property to "addToCart2()"
 - Set the enabled property to an expression similar to the one for the auto-layout panel created earlier.
 - You may have errors because the button click event handler function is not yet implemented.

Within the opening and closing <mx:Panel> tags – add a Script tag

- Add a <mx:Script> tag.
- If Flex Builder does not add the <![CDATA[]]> block in your <mx:Script> tag, copy the following to your application file. All ActionScript outside MXML tags must be within the <![CDATA[]]> block.

```
<mx:Script>
<![CDATA[

]]>
</mx:Script>
```

Inside the <mx:Script> tag CDATA block – implement Button click event handler function

- Add this code to implement the addToCart2() method:

```
private function addToCart2():void {
    var cartStr:String = "Added ";
    if(photoHS2.value > 1){
        cartStr += photoHS2.value + " photos";
    }else{
        cartStr += photoHS2.value + " photo";
    }
    cartStr += " to the cart.";
    mx.controls.Alert.show(cartStr);
}
```
- Refresh the application with F5 and examine the Panel. All controls in both Panels should be fully functional, and the launched application should look very close to the solution application.

AppExcessiveAutoLayout1

This application shows using too many auto-layout containers. Even though there are many containers here, resize the browser and performance is not affected that badly, though this is not a real world scenario.

AppExcessiveAutoLayout2

This application shows using too many auto-layout containers. Even though there are many containers here, resize the browser and performance is not affected that badly, though this is not a real world scenario.

Exercise1

This exercise application is designed to build upon and extend the skills in Flex 3 attained thus far. Some concepts required to build this application have been presented briefly, others have not been touched upon at all. With this in mind, consider this exercise a stretch application where you are confronted with problems you must solve by being "nimble" as you seek out the solutions, a good quality for application developers.

There are usually several ways to implement any one feature, and each is valid. Try your best to implement the exercise application to reproduce as closely as possible the provided screenshot, and the expected behavior as seen in the solution application. Even if you reproduce the application visually, you might implement code that is less efficient, or perhaps more efficient, than the code in the solution code.

It may be good if you collaborate with other training participants to overcome challenges in completing this exercise, or you may wish to work alone.

A number of resources have been provided to guide and help you:

- description of the theoretical purpose of the application
- description of how the app should function as the user manipulates it
- links to relevant Adobe Flex 3 LiveDoc API pages
- instructions on what to do, particularly on features to be covered in more detail later

Purpose of the Application

The application is for a fictional art gallery "Western States Galleries" that is developing a Flex 3 application that will allow users to browse photography available for purchase. Users can add photos to their cart. The cart is NOT fully functional, and does not keep a running total if the user adds several images to the cart.

Application Behavior

The application should behave as follows upon user interaction:

- when user clicks a photo in the photo strip at the bottom of the application, that photo should be displayed in the large image frame and also in the small panel displaying price information.
- when user clicks a photo in the photo strip at the bottom of the application, the title of that photo should be displayed under the photo in the large image frame and also in the title of the small panel displaying price information.
- when user clicks a photo in the photo strip at the bottom of the application, the description and price for that photo should be displayed in the small panel allow user to add photos to the cart.
- the "Add to Cart" button should be disabled until the quantity has been increased to 1 or more.
- when the user clicks the "Add to Cart" button, an Alert should display with their total purchase price, calculated as the quantity of photos times the price. NOTE: the application does not need to keep track of what photos have already been clicked and added to the cart, and does not need to display summary information of all purchases.
- if the user has increased the quantity for a picture, when the user changes to a different picture the quantity should be reset to zero.

Links to Relevant Adobe Flex 3 LiveDoc API Pages

<http://livedocs.adobe.com/flex/3/langref/mx/controls/HorizontalList.html>
<http://livedocs.adobe.com/flex/3/langref/mx/controls/Image.html>
<http://livedocs.adobe.com/flex/3/langref/mx/controls/NumericStepper.html>
<http://livedocs.adobe.com/flex/3/langref/mx/containers/Panel.html>
<http://livedocs.adobe.com/flex/3/langref/mx/rpc/http/mxml/HTTPService.html>
<http://livedocs.adobe.com/flex/3/langref/mx/formatters/NumberFormatter.html>
<http://livedocs.adobe.com/flex/3/langref/mx/collections/XMLListCollection.html>
<http://livedocs.adobe.com/flex/3/langref/mx/containers/ControlBar.html>
<http://livedocs.adobe.com/flex/3/langref/mx/controls/Text.html>

You can also search the Flex Builder help system for information on Flex controls, MXML and ActionScript language help, etc.

Instructions on What to Do

Launch the solution application (use the following method so you do not see the solution code)

- Right-click on Exercise1Solution.mxml and select Run Application from the context menu.

Save the application file

- Immediately save Exercise1.mxml as Exercise1MyApp.mxml

Tweak UI Later

- As you proceed in creating the application, you might want to wait until later when the functionality has been implemented to tweak properties and style properties related to how the app UI displays, so the code stays smaller and easier to read.

The following will be the basic container structure of the application:

```
<mx:Application>
  <mx:HBox>
    <mx:VBox>
      VBox CONTENTS
    </mx:VBox>
    <mx:Panel>
      <mx:HBox>
        OTHER HBOX CONTENTS
        <mx:VBox>
          VBox CONTENTS
        </mx:VBox>
      </mx:HBox>
      <mx:ControlBar>
        CONTROLBAR CONTENTS
      </mx:ControlBar>
    </mx:Panel>
  </mx:HBox>
  OTHER APPLICATION CONTENTS
</mx:Application>
```

Within the opening and closing <mx:Application> tags – add an HBox in MXML

- Add a <mx:HBox> control and set properties:
 - id property to "mainHB".

Within the opening and closing <mx:HBox> tags – add a VBox in MXML

- Add a <mx:VBox> control and set properties:
 - id property to "currImageVB".

Within the opening and closing <mx:VBox> tags – add a Label in MXML

- Add a <mx:Label> control and set properties
 - text property to "Western States Galleries"

Within the opening and closing <mx:VBox> tags – add a Image in MXML

- Add an <mx:Image> and set properties
 - Id property to "currImage"
 - source property to "assets/images/BigSur1.png"

Within the opening and closing <mx:VBox> tags – add a Label in MXML

- Add a <mx:Label> control and set properties
 - id property to "imgCaption"
 - text property to "Big Sur Coastline 1"

Within the opening and closing "mainHB" <mx:HBox> tags – add a Panel in MXML

- Add the <mx:Panel> implementing auto-layout from AppAutoLayout4 within mainHB and set properties in the Panel and it's child controls:
 - Panel title property to "Big Sur Coastline 1"
 - Image source property to "assets/images/BigSur1.png"
 - Text text property to "This photo beautifully captures the wonder of the Big Sur coastline."
 - Label text property to "Price: 49.95"
 - Move the <mx:Script> tag block outside of the Panel and place it at the application level, just below the opening <mx:Application> tag.

Within the opening and closing <mx:Application> tags – add a HorizontalList in MXML

- Add a <mx:HorizontalList> just above the closing </mx:Application> tag and outside any other containers and set properties.
 - id property to "viewer"
 -

Launch the application and confirm the following

- You should have no errors or warnings.
- The UI should look similar to that in Exercise1Screenshot3.PNG.
- Ensure the Panel functionality works (NumericStepper, Button enable and click functionality).

Your application may differ at this point

- Yours application may look different depending on what properties and style properties you have set.
- You will set many properties and style properties later in this exercise as you tweak the app to display according to the specifications.

Inside the <mx:Script> tag – create an XMLListCollection object as HorizontalList dataProvider

- Define a private bindable variable photoData of type XMLListCollection as the dataProvider for the HorizontalList, using e4x syntax when instantiating the XMLListCollection to bring in all "photo" elements from the data file **data\xml\westernGalleries.xml**.

Within the opening and closing <mx:Application> tags (just after the closing </mx:Script> tag)

- Immediately after the closing </mx:Script> tag, add the following:

```
<mx:HTTPService id="photoRequest" resultFormat="e4x" useProxy="false"
    result="photoInfoHandler(event)" url="data/xml/westernGalleries.xml"/>
```

We will cover HTTPService in detail in a future session, but for now we want to bring in data from an XML file. The resultFormat property is set to "e4x" so we can use e4x syntax to access the data. If you save and compile now you may get an error that photoInfoHandler() does not exist, we'll create that soon.

Inside the <mx:Script> tag – create the photoInfoHandler() HTTPService result event function

- Add this code to implement the photoInfoHandler() method:

```
private function photoInfoHandler(evt:ResultEvent):void{
    var xmllist:XMLList = evt.result.photo;
    photoData = new XMLListCollection(xmllist);
}
```

If you get this error: Type was not found or was not a compile-time constant: ResultEvent

You need to add the following to your <mx:Script> tag:

```
import mx.rpc.events.ResultEvent;
```

BUT HERE IS A TRICK I USE ALL THE TIME:

Because I copy/paste code in my apps all the time, I don't try to necessarily memorize the paths to all these classes. So do this:

- in the line with photoInfoHandler(evt:ResultEvent):void
- select and delete the text ":ResultEvent" after evt
- type : and then Res
- press the UpArrow key on your keyboard to highlight **ResultEvent** in the context list Flex Builder displays
- press the Enter key
- import mx.rpc.events.ResultEvent; should be added for you

If you get an error that type XMLListCollection cannot be found, add an import statement for XMLListCollection using the steps just outlined for ResultEvent.

Inside the opening <mx:Application> tag – call the HTTPService send() method upon creationComplete

- Set the creationComplete event property to "photoRequest.send();" This will execute the HTTPService request when the application launches.

Inside the opening <mx:HorizontalList> tag – set the dataProvider property

- dataProvider property to "{photoData}" (don't forget the binding curly braces)
- Launch the app and you should see raw XML in the HorizontalList.

Within the opening and closing <mx:HorizontalList> tags – add an inline itemRenderer

- Add the following to define an inline itemRenderer (we'll discuss itemRenderers in greater detail later):

```
<mx:itemRenderer>
    <mx:Component>
        <mx:Image width="230" height="140" source="{data.path}"/>
    </mx:Component>
</mx:itemRenderer>
```

The itemRender will create an Image control for each item in the dataProvider, in this case <photo> XML elements, and the **data.path** above will extract the <path> element from each <photo> element.

- Launch the application and you should see pictures displaying in the HorizontalList, similar to Exercise1Screenshot4.PNG. You will tweak the properties and style properties later so the display is more like the completed app screenshot Exercise1Screenshot1.PNG.

Open file "src\data\xml\westernGalleries.xml" and examine the data.

- The first entry looks like this:

```
<photo>
    <path>assets/images/BigSur1.png</path>
    <title>Big Sur Coastline 1</title>
    <description>This photo beautifully captures the wonder of the Big Sur coastline.</description>
    <price>49.95</price>
</photo>
```

- The HTTPService result handler function contained this text:
evt.result..photo

That is e4x syntax that assigns the "photo" elements from our XML data to an XMLList, which is used to create our XMLListCollection.

- The HorizontalList itemRenderer Image component has this property: `source="{data.path}"` which also uses e4x syntax to access the "path" XML element from each "photo" XML element. Similar uses of e4x are used later for the photo title, description, and price.

Inside the opening <mx:Image> tag with id "currImage" – set the source property

- Change the "source" property to `"{viewer.selectedItem.path}"`

Inside the opening <mx:Image> tag within the Panel – set the source property

- Change the "source" property to `"{viewer.selectedItem.path}"`

Inside the opening <mx:HorizontalList> tag – set the creationComplete property

- creationComplete property to `"viewer.selectedIndex=0;"`

As you click each image in the HorizontalList, that image is displayed in the main big image control and in the Panel image control. But when the app first loads, no image displays, so you add the above code for the HorizontalList creationComplete event. It ensures the first image is selected upon startup, and thus is used to initially populate the two Image controls.

Launch the app and click on images

Launch the app and click on images in the HorizontalList and you will probably see a flicker as the newly selected image loads. To prevent this you must set the size of the Image tags. This can require some experimentation because as seen in other session one applications you built, if you set the width for an image that differs from its actual size in the .png file, the image height tag may reserve empty space for the full height. If you get the values just right, the flickering will be minimal and each selected image will display in both Image tags nicely. Refer to the completed app screenshot, but if you don't get it just right don't worry too much, you can tweak it further later, or get the right dimensions from the solution file later.

Inside the opening <mx:Panel> tag – set the title property

- Change the "title" property to `"{viewer.selectedItem.title}"`

Inside the opening <mx:Label> tag id "imgCaption" within the Panel – set the text property

- Change the "text" property to `"{viewer.selectedItem.title}"`

Inside the opening <mx:Label> tag for the price within the Panel – set the text property

- Change the "text" property to `"Price: ${viewer.selectedItem.price}"`

These changes ensure the selected photo "title", "description", and "price" data are used to populate corresponding controls in the main big photo UI and in the Panel. As you click each image in the HorizontalList, the title for that image is displayed in the caption for the main big image area and in title of the Panel, and the price displays in the Panel.

Launch the app and click on images

Launch the application and select a few photos in the HorizontalList and verify the image and caption in the main big photo area, and the image, title, description, and price in the Panel update correctly.

Inside the opening <mx:HorizontalList> tag – set the change event property

- change property to `"photoHS1.value=0;"`

Notice that if you increase the quantity for an image, and then select a different image, the quantity remains at its previously increased value. We are not creating a fully functional shopping cart in this app, but the quantity should reset to zero when the selected image is changed. The code entered above ensures that when the selectedItem of the HorizontalList "changes", the NumericStepper value will be reset to zero.

Inside the opening <mx:HorizontalList> tag – change a property to ensure 5 photos display

- Next you need to figure out how to ensure that exactly five photos are displayed at all times in the HorizontalList. Examine the API LiveDocs page and determine what code should be added.
<http://livedocs.adobe.com/flex/3/langref/mx/controls/HorizontalList.html>

Inside the <mx:Script> tag – change the addToCart1() method to display desired informaton

- Now you need to change the addToCart1() method that currently displays an Alert with the number of photos "added to the cart". We want it to also display a message informing the user of their total cost. This is an example of the text that should be displayed:

"Added 2 photos to the cart.\nTotal cost \$99.90."
(the \n forces a line wrap so sentences are on separate lines)

Modify the addToCart1() method so it looks like this:

```
private function addToCart1():void {
    var nf:NumberFormatter = new NumberFormatter();
    nf.precision = 2;
    var cartStr:String = "Added ";
    if(photoHS1.value > 1){
        cartStr += photoHS1.value + " photos";
    }else{
        cartStr += photoHS1.value + " photo";
    }
    cartStr += " to the cart.\nTotal cost $" +
        nf.format((photoHS1.value * Number(viewer.selectedItem.price))) + ".";
    mx.controls.Alert.show(cartStr);
}
```

Notice these key points:

- We create a NumberFormatter object and set its precision to 2, as we are dealing with monetary values.
- We use the NumberFormatter to format the price information as a number with 2 decimal precision. If we did not do this the prices would display as \$99.9 (missing 0)

Tweak the UI

You are almost done !!! All you need to do now is modify properties and style properties throughout the application to change the width, height, positioning, color, padding, gap, font size and style, alpha, etc. so the application displays as closely as possible to the completed application screenshot Exercise1Screenshot1.PNG.

Getting Help

Try to collaborate with the other training participants if necessary to help each other solve problems you run into, reference the LiveDoc links provided earlier in this document, search Google, and if you are stumped and need help, email me at glafrance@chikaradev.com and I will try to respond as soon as possible.