OSCON

HANDS ON LAB

BUILDING MULTI-SCREEN MOBILE APPLICATIONS WITH FLEX AND SPRING

This hands on tutorial will teach attendees how to build their first mobile application using Adobe's Open Source Flex SDK (http://opensource.adobe.com/wiki/display/site/Home) and compiling it for the Android and BlackBerry operating systems. Using an open source tool chain, including the Flex SDK for the UI, Spring, and Hibernate, developers can build apps that work on a variety of devices including PCs, Android phones / tablets, the BlackBerry PlayBook tablet, iPhones, and iPads. This session will walk developers through the steps for creating these cross-device apps with Flex and Java.

If you have any trouble with the setup, please send an email to me at ksutton at adobe-dot-com. I also plan to be in the lab room 20 minutes before the session to answer any questions or help out.

See you there! This is going to be the best workshop I have ever had the pleasure of teaching! An expanded version of this course will be taught at this year's Adobe MAX in Los Angeles.

Keith Sutton @keithsutton

Contents

Preparation Guide and Software Required ... 3
Mandatory ... 3
Free IDE Options ... 3
Preparation Options ... 3
Lab 1 – New Project ... 4
Lab 2 – Getting System Details ... 7
Lab 3 – Adding a Back Button to the ActionBar ... 8
Lab 4 – Accelerometer ... 10
Lab 5 – Zoom Gesture ... 12
Lab 6 – Rotate Gesture ... 14
Lab 7 – Pan Gesture ... 15
Lab 8 – SMS ... 16
Lab 9 – Email ... 18
Lab 10 – Camera ... 19
Lab 11 – Microphone Access ... 20
Lab 12 – Service View ... 22
Lab 13 – FilesView ... 24
Lab 14 – Flex 4 and Spring ... 25

Preparation Guide and Software Required

All courseware will be distributed at the venue itself. In order to take this course, you will require the following software and assets.

Mandatory

1. Flash Builder version 4.5.1 with Android, Blackberry Playbook and iOS support (60 day trail)– http://www.adobe.com/go/try_flashbuilder
2. Adobe Photoshop version CS4 or greater –http://adobe.com/PhotoshopFamily
3. Bring your own laptop. Operating System: Windows or Mac OS X.
4. Bring a power cord.
5. A cable to tether it to your laptop. It is not required that you bring you're a mobile device (Apple, Android or Blackberry Playbook) but it is highly recommended.
6. Download project assets here: http://www.spinnaret.com/OSCON/OSCON_Building_Multi_Screen_Mobile_Applications_with_Flex_and_Spring.zip.

Free IDE Options

There are free IDE options for developing Flash/Flex applications using the open source Flex SDK, but there, in some cases, significant manual setup and project configuration is required (the cost of free). This setup is not factored into the lab time and those wishing to explore this option should do so before or after the lab. The most popular free options for Flex 4:

- FlashDevelop. Current version 4 is in Beta, download this IDE here (with patch) be sure to follow setup instructions and documentation. There are guides to building and exporting (Part 1 and 2).
- Eclipse IDE. Sean Smith provides a great tutorial on setting up Eclipse (either Classic or PHP).

Preparation Options

Here are some tutorials for those interested in exploring more before or after:

- Mobile Development using Adobe Flex 4.5 SDK and Flash Builder 4.5, Narcisco Jaramillo
- Creating your first AIR application for Android with Flex SDK
- Building iOS Applications using Flex and Flash Builder 4.5, Serge Jespers
- Flex for the BlackBerry PlayBook in 90 Minutes, Chripstophe Coenraets

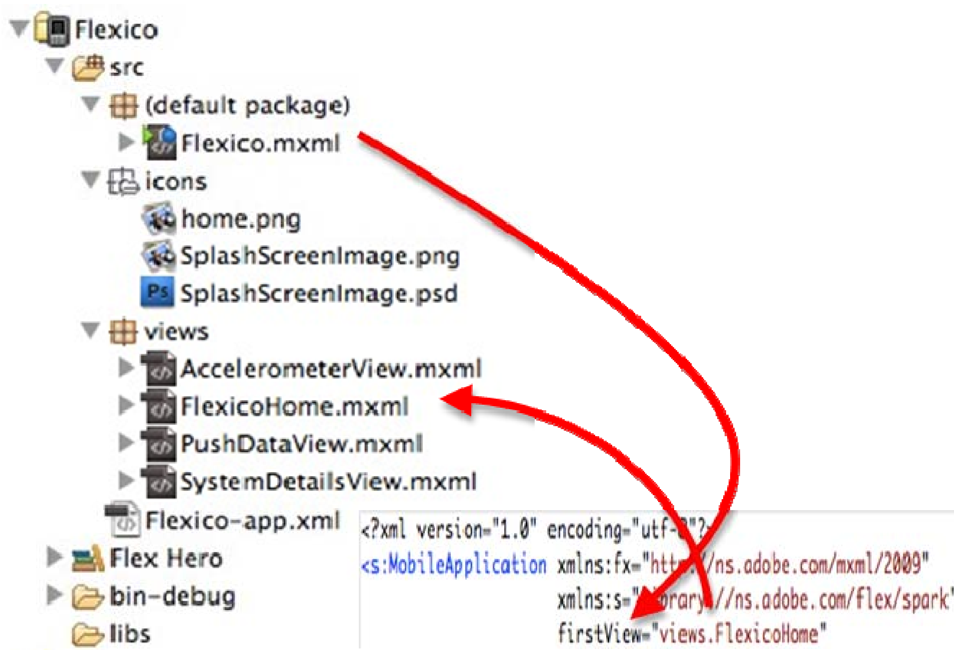
Lab 1 – New Project

During this lab, you will set up a new project, learn how to use a list and to pass data between views. This section of the course should take around 20 minutes to complete.

1.1 Open up Flash Builder 4.5 (Burrito) and select **New -> Flex Mobile Project.**

1.2 Give project a name then hit **Next**. Explain the defaults selected and use the View Navigator based project settings. Click **Finish**.

Explain: How the project is structured. Demonstrate the view navigator and the fact there is a main entry point into the app but it also has a first view.



1.3 Make sure you open the firstView file (the name may vary) and add an <fx:Script> block.

1.4 Within the script block, create a new [Bindable] Array called **demos** as shown below and add the following values to the Array.

```
[Bindable]
Private var demos:Array = new Array(" Push Data", "Accelerometer", "Zoom
Gesture", "Getting System Details", "SMS", "Email", "Service View");
```

1.5 Now create and instantiate a new [Bindable] **ArrayCollection** object called **demoList.**

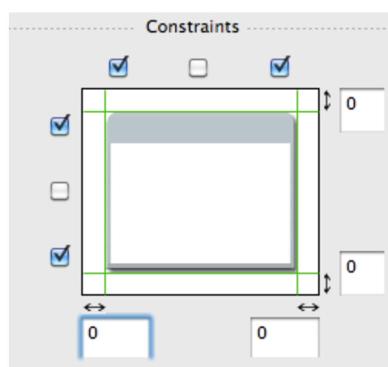
```
[Bindable] private var demolist:ArrayCollection = new
                ArrayCollection();
```

1.6 Add a new function called **init()**, modify the root element of the application to call it based on the **viewActivate** event. **Explain:** what the event is and when it is fired. Also explain the destruction policy.

1.7 Within the **init()** function, add the following lines of code to bind values from the **Array** to the **ArrayCollection**. **Explain:** What the differences are between the Array and ArrayCollection classes.

```
private function init():void
      {
            for (var i:int = 0; i < demos.length; i++)
            {
                  demolist.addItem(demos[i] as String);
            }
      }
```

1.8 Switch to Design View and add an **<s:List>** to the main area of the application. With the List selected, **explain** how to bind it to the **top, bottom, left** and **right** and why this is important for mobile apps that auto-orientate.



1.9 Switch back to the source code view and add a **dataProvider** attribute to the **List** and bind it to the {demoList} object. **Explain** the curly brackets (binding) expression and [Bindable] keyword if there are newbies in the room.

1.10 With the cursor in the **List** component, hit the space bar and start typing the word "**change**". Use the auto complete to detect the change event and then click

enter again to allow it to auto generate the event handler. **Explain** this powerful new feature of FlashBuilder.

```
<s:List change="|" id="viewsList" left="0"
                  Generate Change Handler
```

1.11 Place your cursor over the generated function name and select "**Refactor**". Rename it to "**SelectDemo**".

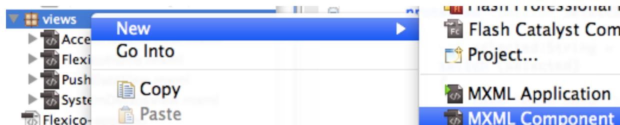
1.12 Within the **SelectDemo** function, declare a member variable named **selected** with a type of **String**

```
var selected:String = viewslist.selectedItem as String;
```

1.13 Within the **SelectDemo** function, add a **Switch-Case** structure as shown below to link navigation to the appropriate selected item.

```
 switch (selected)
{
        case "Push Data" :
          navigator.pushView(PushDataView, selected);
          break;
 }
```

1.14 Right click on the Add a **New > MXML Component** to the **Views** package named **PushDataView**.



1.15 Within the **PushDataView**, add an **<s:Button>** and center it vertically and horizontally. Explain why we do this (horizontal and vertical orientation).

1.16 Add a click handler to the button and add the following code

```
<s:Button label="Back" horizontalCenter="0" verticalCenter="0"
          click="navigator.popView()"/>
```

1.17 Add an **<s:Label>** to the application and bind it to the **left, right** and **top** by 50, 50 and 25 pixels respectively. Add the **text** property and bind it to **{data}. EXPLAIN:** the key variable "data". The PushDataView class now should appear

as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="PushDataView">
      <s:Label left="50" right="50" top="25" text="{data}" textAlign="center"
              verticalAlign="middle"/>
      <s:Button label="Back" horizontalCenter="0" verticalCenter="0"
               click="navigator.popView()"/>
</s:View>
```

1.18 Run the demo and vary the orientation on the simulator. Note the passing of the data from one view to another.

Lab 2 – Getting System Details

In this lab you will learn how to access System details and to lay out more complex views.

1. **Close** the **PushDataView.mxml** is not already done.

2. Create a new view (**New > MXML Component**) called **SystemDetailsView**.

3. Add the code below to your component. **EXPLAIN:** the complex layout and how it works in both vertical and horizontal modes.

```
<s:Label y="35" width="200" horizontalCenter="-135" text="Screen Type:"
         textAlign="right"/>
<s:Label y="35" width="225" horizontalCenter="85" text="{touchScreenType}"
         textAlign="left"/>

<s:Label y="65" width="200" horizontalCenter="-135" text="Device Name:"
         textAlign="right"/>
<s:Label y="65" width="225" horizontalCenter="85" text="{thisDeviceName}"
         textAlign="left"/>

<s:Label y="95" width="200" horizontalCenter="-135" text="Aspect Ratio:"
         textAlign="right"/>
<s:Label y="95" width="225" horizontalCenter="85"
         text="{thisDevicePixelAspectRatio}" textAlign="left"/>
```

4. Create a Script block and add the 4 variables mentioned above and type them as follows. Make them Bindable. You may have to `import`

flash.system.Capabilities;

```
[Bindable] private var touchScreenType:String =
flash.system.Capabilities.touchscreenType; //"finger" || "stylus"
[Bindable] private var thisDeviceName:String =
flash.system.Capabilities.manufacturer; //format "Adobe OSName". The value for
OSName could be "Windows", "Macintosh", "Linux"
[Bindable] private var thisDevicePixelAspectRatio:Number =
flash.system.Capabilities.pixelAspectRatio; //returns number
```

5. Open up the first view and add a **case** clause to the **switch** statement to allow the navigator to push the **SystemDetailsView**.

```
    case "Getting System Details" :
          navigator.pushView(SystemDetailsView);
          break;
```

6. Run the application and navigate to the view. **EXPLAIN:** Why this might be important.

Lab 3 – Adding a Back Button to the ActionBar

During this lab you will learn how to make a back button that is accessible from all views. When pushed, it simply calls the **popView()** (and later **popToFirstView()**) methods of the view navigator.

1. Close all files except the main entry point into your application (or open it if all are closed).
2. Add the following code to your project

```
<s:navigationContent>
      <s:Button label="Back" click="navigator.popView()" />
</s:navigationContent>
```

3. Run the project.

4. Stop the project running and now add a new directory called "icons" under the /src/ folder for your project.

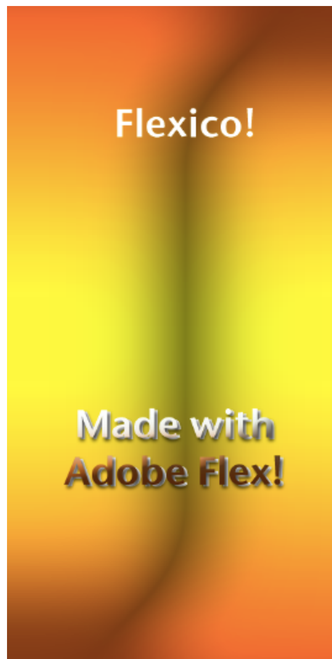
5. Copy the image from the URL http://www.22ndcenturyofficial.com/images/home.png into the /icons

folder.

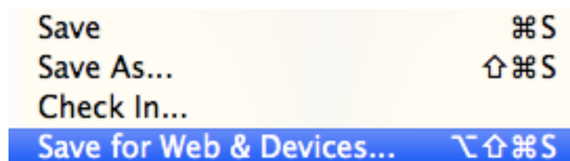
6. Replace the code you added before with the code below.

```
<s:navigationContent>
	<s:Button icon="@Embed('icons/home.png')"
			click="navigator.popToFirstView()"/>
</s:navigationContent>
```

7. Open up Adobe PhotoShop and create a new image that has the following parameters:
   Name = SplashScreenImage
   Width = 400
   Height = 800
   Resolution = 72 DPI



8. Make a splash screen similar to the one below and then select **File > Save for Web & Devices**. **EXPLAIN:** How this is different from normal "Save".



9. Save the file to the src/icons folder. Then add the splashScreen code to the application entry point class as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<s:MobileApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                     xmlns:s="library://ns.adobe.com/flex/spark"
                     firstView="views.FlexicoHome"
                     splashScreenImage="@Embed('icons/SplashScreenImage.png')">

      <s:navigationContent>
             <s:Button icon="@Embed('icons/home.png')"
                            click="navigator.popToFirstView()"/>
       </s:navigationContent>

</s:MobileApplication>
```

10. Run the Application.

Lab 4 – Accelerometer

During this lab you will learn how to access the accelerometer and display the results. ***Note – This will not work on a device that has no accelerometer.***

4.1 Open up the applications **firstView** in the default package and add a case clause to the switch statement to allow the navigator to push the view.

```
       case "Accelerometer" :
             navigator.pushView(AccelerometerView);
             break;
```

4.2 Add the **AcceleratorView** to your project (right click on Views, **New > MXML Component**)

4.3 Within the AcceleratorView, add an **<s:Label>** and bind it to the **top, right** and **left** by values of 25 pixels. Give it an **id** property with the value of "l".

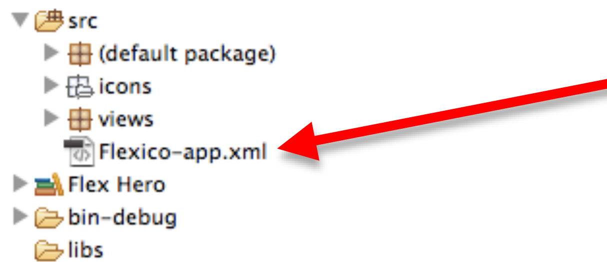
4.4 Add an **<fx:Script>** block and insert the following import statements and functions.

```
<fx:Script>
      <![CDATA[
      import flash.sensors.Accelerometer;

      public var a:Accelerometer = new Accelerometer();

      private function init():void
      {
            if(Accelerometer.isSupported)
            {
```

```
                    l.text = "Acquiring data...";
                    a.addEventListener(AccelerometerEvent.UPDATE, readA);
            }
            else
            {
                    l.text = "Accelerometer is not supported";
            }
    }

    private function readA(e:AccelerometerEvent):void
    {
            l.text = "accelerationX = " + e.accelerationX +
                     "\naccelerationY = " + e.accelerationY +
                     "\naccelerationZ = " + e.accelerationZ +
                     "\ntimestamp = " + e.timestamp;
    }
    ]]>
</fx:Script>
```

4.5 Call the **init()** function based on the **viewActivate** event.

4.6 Android works on a system of ***intents, filters*** and ***actions***. In order for an application to request access to a feature, the developer must declare the application's intent to use a permission. This is done in the manifest, a separate file that makes up part of the *.apk package. This allows the application user to understand what the application is requesting access to. In your main project folder, **open** up the **application descriptor file**. This file will have the same name as the application concatenated with "-app.xml".



4.7 Around line number 190, if not already present, add the declaration necessary to access the LOCATION and FINE_LOCATION. These appear as follows:

```
<!-need to verify if all these are required→
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

4.8 **Save** and **close** the **application descriptor file** containing the manifest then **run** the program on a device that has an accelerometer.

EXPLAIN: at this point explain how to run on a USB tethered device and change the run configurations for the project.

Lab 5 – Zoom Gesture

During this lab you will learn how to work with the transform gestures for zoom

5.1 Open up the application's **firstView** and add a case clause to the switch statement to allow the navigator to push the view.

```
case "ZoomGesture" :
      navigator.pushView(ZoomView);
      break;
```

5.2 Add a new view named **ZoomView** to your project

5.3 Add the code shown below to the ZoomView:

```
<fx:Script>
<![CDATA[
      private function init():void
      {
      myBC.addEventListener(TransformGestureEvent.GESTURE_ZOOM, onZoom);
      }

      private function onZoom(event:TransformGestureEvent):void
      {
            var mySprite:Sprite = myBC as Sprite;
            mySprite.scaleX *= event.scaleX;
            mySprite.scaleY *= event.scaleY;
      }
]]>
</fx:Script>


      <fx:Declarations>
            <fx:Array id="colorArray">
                  <s:GradientEntry color="0x22FF55"/>
                  <s:GradientEntry color="0xFF0000"/>
                  <s:GradientEntry color="0x00FF00"/>
                  <s:GradientEntry color="0x0000FF"/>
            </fx:Array>
      </fx:Declarations>

      <s:BorderContainer id="myBC" width="100%" height="100%"
backgroundColor="#C2C414">
            <s:Ellipse height="300" width="220" x="10" y="10" >
                  <s:fill>
                        <s:LinearGradient entries="{colorArray}" />
```

```
                </s:fill>
            </s:Ellipse>
        </s:BorderContainer>
```

5.4 Run the project. Purposely scale the graphic very small and take your fingers off the screen then discuss the best practices of not allowing scaling below a certain size.

5.5 Stop the project and return to the Source Code view.

5.6 Add the following code (highlighted in shading).

```
<fx:Script>
<![CDATA[
        private function init():void
        {
        myBC.addEventListener(TransformGestureEvent.GESTURE_ZOOM, onZoom);
        }

        private function onZoom(event:TransformGestureEvent):void
        {
                var mySprite:Sprite = myBC as Sprite;
                if(mySprite.scaleY.valueOf() >= 1)
                {
                        mySprite.scaleX *= event.scaleX;
                        mySprite.scaleY *= event.scaleY;
                }
                else
                {
                        mySprite.scaleX = 1;
                        mySprite.scaleY = 1;
                }
        }
]]>
</fx:Script>
```

5.7 Run the project again and note that it cannot scale below a factor of 1.

Lab 6 – Rotate Gesture

During this lab you will learn how to work with the transform gestures that control rotatation.

6.1 Add a new view to the project called **RotateView**.

6.2 Open up the application's first view and add a case clause to the switch statement to allow the navigator to push the RotateView.

6.3 Add an **<s:Image>** to the **RotateView**. Use the **SplashScreenImage.png** you created in the previous labs. In the Source Code view, set the **horizontalCenter="0" verticalCenter="0"** and **scaleMode="letterbox".**

6.4 Add an id property to the image and provide the value as "I".

6.5 Add an **init()** function within a Script block and call it based on **viewComplete**.

6.6 Within the **init()** function, add an event listener to the I (image) object and make a call to a new function named "**rotate**".

6.7 Add the **rotate** function wioth a private access modifier and ensure the method signature accepts the `TransformGestureEvent.GESTURE_ROTATE` event.

6.8 Within the **rotate** function, add the first line of code to create a member variable of type **Sprite** named **mySprite** and cast the events' currentTarget property as a Sprite.

6.9 Add a second line of code to declare that the rotation of the Sprite is equal to the accumulative rotation of the event.

6.10 **Run** the project and explore the two fingered rotation gesture.

6.11 **Close** the project and encourage students to explore with other variations such as changing **rotation** to **rotationX, Y and Z**. Change the operator from **+=** to **=+** and note the results.

6.12 The code should appear as below.

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="RotateView"
        viewActivate="init()">
<fx:Script><![CDATA[
      private function init():void
      {
         i.addEventListener(TransformGestureEvent.GESTURE_ROTATE, rotate);
      }

      private function rotate(e:TransformGestureEvent):void
      {
            var mySprite:Sprite = e.currentTarget as Sprite;
            mySprite.rotation += e.rotation;
      }
]]>
```

```
</fx:Script>
<s:Image horizontalCenter="0" verticalCenter="0" scaleMode="letterbox"
         id="i" source="icons/SplashScreenImage.png"/>
</s:View>
```

Lab 7 – Pan Gesture

During this lab you will learn how to work with pan Gesures. Pan gestures allow users to swipe content on an X or Y plane using two fingers. Note: You will need to run this on a device that supports the Pan Gesture to have it work.

7.1 Follow the steps in previous exercises to create a new **Array** item named "**Pan Gesture**", add a clause to the **Switch-Case** statement in the home view and then add a new view named **PanView** to the project.

7.2 On the applications home view, add the **PanView** to the **array** of **demos** as a choice and also a **case** clause in the **switch** statement as per previous exercises.

7.3 Open the **PanView**, cut and paste the spark **Array** and **BorderContainer** code from the **ZoomView** exercise 5.3 above.

7.4 Add an event to listen for the **TransformGestureEvent.GESTURE_PAN** and pass the event to a new function called **pan()**.

7.5 Cast the event into a **Sprite** variable called **mySprite** and use the events **offsetX** and **offsetY** to assign accumulative values to the Sprite. The scripting should look as follows.

```
<fx:Script>
      <![CDATA[
            private function init():void
            {
                  myBC.addEventListener(TransformGestureEvent.GESTURE_PAN, 
pan);
            }

            private function pan(e:TransformGestureEvent):void
            {
                  var mySprite:Sprite = e.currentTarget as Sprite;
                  mySprite.x += e.offsetX;
                  mySprite.y += e.offsetY;
            }
      ]]>
</fx:Script>
```

7.6 Run the application and navigate to the Pan Gesture demo. After verification, close the application and experiment with changing '+=' to '=+' and also enhancing the offset velocity by adding math code such as shown below.

```
mySprite.x += (e.offsetX * 1.7);
mySprite.y += (e.offsetY * 1.7);
```

Lab 8 – SMS

During this lab you will learn how to invoke an SMS message to be sent.

8.1 Open up the first view and add a case statement to allow the navigator to push the view.

```
case "SMS" :
      navigator.pushView(SMSView);
      break;
```

8.2 **Add** the **SMSView** to your project

8.3 **Insert** an **<s:Button>** with the label "SMS" and **generate an event handler** for the **click** event. Refactor the function name to ***sms*** and remove the event from the signature and from being passed.

8.4 **Anchor** the button to the **top** by a value of **180** pixels and set the **width** to **100%.**

8.4 Add a **<s:TextInput>** and **<s:Label>** to the view. Set the attributes for all three components as shown below:

```
<s:Label top="35" horizontalCenter="0" text="Enter a mobile phone number"/>
<s:TextInput id="t" top="90" horizontalCenter="0" text="+16047263329"
             textAlign="center"/>
<s:Button    top="180" width="100" label="SMS" click="sms()"
             horizontalCenter="0"/>
```

8.5 In the **sms function**, add the following code

```
	protected function sms():void
	{
		var n:String = t.text;
		navigateToURL(new URLRequest("sms:" + n));
	}
```

8.6 **Run** your application on the mobile device and test it.

8.7 Close your application and add a new **<s:Button>** anchored at 290 pixels from the top and **label** it ***"phone".***

8.8 **Generate** a **click event handler** for the new button, **refactor** the name to ***"phone"*** and remove the **event**.

8.9 Add the same code as above in section 8.5 to the newly created event handler function and **change "sms"** to **"tel"** then run your application on a mobile device again.

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
		xmlns:s="library://ns.adobe.com/flex/spark" title="SMSView">
<fx:Script>
<![CDATA[
	protected function sms():void
	{
		var n:String = t.text;
		navigateToURL(new URLRequest("sms:" + n));
	}

	protected function phone():void
	{
		var n:String = t.text;
		navigateToURL(new URLRequest("tel:" + n));
	}
]]>
</fx:Script>
<s:Label top="35" horizontalCenter="0" text="Enter a mobile phone number"/>
<s:TextInput id="t" top="90" horizontalCenter="0" text="+16047263329"
			 textAlign="center"/>
<s:Button top="180" width="100" label="SMS" click="sms()"
			 horizontalCenter="0"/>
<s:Button x="177" top="290" label="Phone" click="phone()" />
</s:View>
```

Lab 9 – Email

During this lab you will learn how send an email with the native email client.

9.1 Open up the first view and add a case clause to the switch statement to allow the navigator to push the view.

```
        case "Email" :
              navigator.pushView(EmailView);
              break;
```

9.2 **Add** the **EmailView** to your project.

9.3 **Cut** and **paste** the code from the SMS solution above into the EmailView and **change** the default value of the **<s:TextInput>** to your email address.

9.4 Change the ***"sms"*** in the click handler to ***"mailto"***, change the labels and run the code.

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="EmailVIew">
<fx:Script>
<![CDATA[
      protected function email():void
      {
            var n:String = t.text;
            navigateToURL(new URLRequest("mailto:" + n));
      }
]]>
</fx:Script>
<s:Label top="35" horizontalCenter="0" text="Enter an Email Address:"/>
<s:TextInput id="t" top="90" horizontalCenter="0" text="duane@nickull.net"
            textAlign="center"/>
<s:Button top="180" width="175" label="Email" click="email()"
            horizontalCenter="0"/>
</s:View>
```

Lab 10 – Camera

During this lab you will learn how to work with Camera

10.1 Open up the first view and add a case statement to allow the navigator to push the view.

```
        case "Camera" :
```

```
            navigator.pushView(CameraView);
            break;
```

10.2 Add the CameraView to your project

10.3 Add the code below in the Wardian style (named after James Ward). Note the contrast with coding styles in previous labs. This style has a more compact syntax.

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="CameraView"
        xmlns:mx="library://ns.adobe.com/flex/mx">

    <s:creationComplete>

        var v:Video = new Video(width, height);
        uic.addChild(v);
        var c:Camera = Camera.getCamera();
        c.setMode(width, height, 10);
        v.attachCamera(c);
    </s:creationComplete>
    <mx:UIComponent id="uic" width="100%" height="100%"/>
</s:View>
```

Lab 11 – Microphone Access

In this lab you will learn how to access the microphone. To make this lab work, you will need to use request a permission in the manifest. The permission is:

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
```

1. Add **Microphone** to the array in the **home view** and also add the case clause in the switch statement as with other labs.

2. Create a new view called **MicrophoneView**

3. Add an **<s:BorderContainer>** and anchor it to the **top, bottom, right** and **left** by 0 pixels. Set the **backgroundColor** property to **#FFFFFF**.

4. Add an **<s:Ellipse>** object that has a **width** and **height** of **3 pixels.** Set both the **vertical** and **horizontal** center to **0**.

5. Within the **Ellipse**, add a fill element and then a nested **<s:SolidColor>** element with a **color** property of **#ff0000**.

6. Add a label anchored 25 pixels from the top and 50 pixels from the right and left. Set the label to “Gain Control” and the text alignment to center.

7. Add a **<s:HorizontalSlider>** with an **id** of **gainControl** and anchored to center. The code should appear as follows:

```
<s:BorderContainer backgroundColor="#ffffff" top="0"
                   bottom="0" left="0" right="0" >
      <s:HSlider scaleX="2.5" scaleY="2.5" id="gainControl"
                 left="60" right="60" top="80" height="15"
                 change="init()" maximum="100" minimum="1" value="65"/>

      <s:Ellipse width="3" height="3" id="e" verticalCenter="0"
                 horizontalCenter="0">
                <s:fill>
                   <s:SolidColor color="#ff0000"/>
                </s:fill>
      </s:Ellipse>

      <s:Label left="50" right="50" top="25" height="34" text="Gain Control"
               textAlign="center"/>

</s:BorderContainer>
```

8. Add a **<fx:Script>** block and create an **init()** function. Call the function from the viewActivate event.

9. Within the init() function, add the following lines of code to register an event listener and reference a handler function.

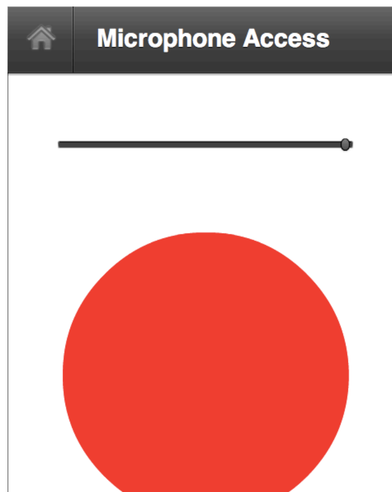
```
      private function init():void
      {
        // recommended best practice is use "-1" to get default system mic.
        var microphone:Microphone = Microphone.getMicrophone(-1);
        microphone.addEventListener(SampleDataEvent.SAMPLE_DATA, UseData);
        microphone.gain = gainControl.value;
      }
```

10. Create the handler function as follows

```
      private function UseData (event:SampleDataEvent):void
      {
             while(event.data.bytesAvailable)
```

```
            {
                var sample:Number = event.data.readFloat();
                e.width = e.height = sample * 1024;
            }
    }
```

11. Run the application and test the Microphone. To work with audio further, please see the Microphone VoiceBox demo and open source code on Technoracle (http://technoracle.blogspot.com)



Lab 12 - Service View

Wines.xml File is required (This file is at http://www.nickull.net/xml/Wines.xml), An internet service is required unless you reference this file locally to test.

1. Add a new view named **ServiceView** to your project and reference it the same as you did in earlier labs.

2. Add an **<s:List>** to the **ServiceView** in the design view and anchor it to the **top, bottom, left** and **right** by **0 pixels**.

3. Find the **Data Services** tab and choose **Data > Connect to HTTP…**

4. **Change** the existing name "**Operation1**" to **getWines**

5. Change the URL of the operation to http://www.nickull.net/xml/Wines.xml

6. In the **Service Name** field, enter **getWines** and click **Finish**.

7. Switch to **Design View** and **drag** the **getWines**() service onto list. This brings up the dialog.

8. Click **Configure Return Type.**

9. In the dialog box ensure the second radio button (**Enter a complete URL**...) is selected and enter the URL in the **URL to get** field. Hit **Next**.

10. Select **vintage** for the root, ensure the **labelField** is set to name and click **Finish**.

11. Run the project and you should get the wines list. Close the project and return to the sources view.

12. Add **change** event handler for list, and select **autogenerate function.**

13. **Refactor** the generated function name to **onSelect()**

14. **Create** new view called **WineDetails**

15. In the **onSelect() function**, write the following code:

```
protected function onSelect(event:IndexChangeEvent):void
{
	var wine:Object = list.dataProvider.getItemAt(event.newIndex);
	navigator.pushView(WineDetails, wine);
}
```

16. In the **WineDetails** view, change the code as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
         xmlns:s="library://ns.adobe.com/flex/spark" title="{name1}"
         viewActivate="init()">

       <fx:Script>
       <![CDATA[
         import valueObjects.Vintage_type;

         [Bindable] private var name1:String;
         [Bindable] private var vintner:String;
         [Bindable] private var price:String;
         [Bindable] private var parker:String;
         [Bindable] private var vintage:String;

         private function init():void
         {
              var thisWine:Vintage_type = data as Vintage_type;
              name1 = thisWine.Name;
              vintner = thisWine.Vintner;
              price = thisWine.Price;
              parker = thisWine.ParkerNotation;
              vintage = thisWine.Vintage;
         }

       ]]>
       </fx:Script>
       <s:Label x="79" y="59" width="256" height="37" text="{vintner}"/>
       <s:Label x="79" y="120" width="256" height="37" text="{price}"/>
       <s:Label x="79" y="186" width="256" height="37" text="{parker}"/>
       <s:Label x="79" y="244" width="256" height="37"
                text="{vintage}"/>

</s:View>
```

17. **Run** the project and navigate to the wines then pick one and navigate to the **WineDetails** view.

18. Ask people if they see any architectural issues?

19. **Stop** the application, turn on **network monitor** and repeat, point out auto loading of wines list.

20. Add the **destructionPolicy** attribute to **none** and re-run.

Lab 13 - FilesView

In this lab, we will explore how to interact with the native file system.

1. Add a new view named FileSystemView and add the switch-case statement as per previous labs to allow it to be navigated to.
2.

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="FileInitialView"  creationComplete="init()">
<fx:Script>
<![CDATA[
import flash.filesystem.File;
import mx.collections.ArrayCollection;

[Bindable]
private var initialDir:File = File.getRootDirectories()[0];
[Bindable]
private var fileList:ArrayCollection = new ArrayCollection();
[Bindable]
private var fileNameList:ArrayCollection = new ArrayCollection();
[Bindable]
private var filePathList:ArrayCollection = new ArrayCollection();

private function init():void
{
	fileList.source = initialDir.getDirectoryListing();
	for (var i:int = 0; i < fileList.length; i++)
	{
		var theFile:File = fileList[i];
		if (!theFile.isHidden)
		{
			fileNameList.addItem(theFile.name);
			filePathList.addItem(theFile.nativePath);
		}
	}
	l.dataProvider = fileNameList;
}
]]>
</fx:Script>
<s:List 	left="0" right="0" top="0" bottom="0" id="l" />

</s:View>
```

Lab 14 – Flex 4 and Spring

This tutorial builds on the mobile project by adding access to data services built using BlazeDS Spring Integration. This tutorial will be based on the latest DZone Refcardz:

http://refcardz.dzone.com/refcardz/flex-4-and-spring-3