CMER
Centre for Mobile Education and Research

Developing Apps for the BlackBerry PlayBook with Adobe AIR

Lab # 3

Fundamentals of Intuitive GUI Design using Flash Builder

Introduction

The objective of this lab is to introduce the basic fundamentals of GUI design. This will be done, by developing an application for the BlackBerry Playbook using the Flex 4 platform (Adobe Air). In this lab, we will build a functional BMI calculator, with a polished, intuative, and interactive graphical user interface (GUI).

A small self-test is included at the end of this lab to test your uderstanding of it's contents.

Software Requirements

-Flash Builder 4.6 (http://www.adobe.com/products/flash-builder.html)
-Adobe Flex SDK (Included with Flash Builder)
-Playbook SDK for Adobe Air 2.0
(https://bdsc.webapps.blackberry.com/air/download/sdk/)
    -Playbook SDK must be integrated into Flash Builder.
-Playbook Simulation Enviroment

Skill Requirements

-This lab assumes basic knowledge of AS3
-This lab assumes basic understanding of the Flash Builder IDE
-This lab assumes you have completed parts 1 and 2 of the *gravityball* lab.

Set-Up

Included in this lab's folder, is a file called "*CalcBMI.fxp". Open this file, by starting up Flash Builder, and selecting *File > Import Flash Builder Project*. Browse for *CalcBMI.fxp* and import it.

This project is a Flex Mobile Project, targeted for the Playbook 2.0. It is a View-Based Application template, and includes images that I've created for use in this lab, as well as two ActionScript classes that will be used to skin components in our App.

The View-Based Application works slightly different than our previous *gravityball* app. It creates a display list based on a display array. The items of this array are screens, and the last item in the array is the screen visible. Popping off the last item in the array, will make the second last item in the array the new last item, and therefore it would be displayed. *Pushing* (more on this later), adds a screen to the array (at the end), and is then the new screen displayed to the user.

This project has a main file called *CalcBMI.mxml*, in the default package. This is the file that launches when the app starts. It sets up the previously mentioned display list, refered to as the *navigator*. Refering to the code, you'll notice the firstview is *views.CalcBMIHomeView*. This means, that once the display list array is created, *CalcBMIHomeView* is added to the array, and since it is the last element (currently, it's the only element so it must be last) it is the *actual* first screen the user will see. So we will put our code in this file (views.CalcBMIHomeView).

<u>Backgrounds</u>

Everyone likes backgrounds, they came about when the first new generation GUI's were being invented (late 70's early 80's) and have been around ever since. They are effective at making things look polished, and we definitely want that polished look for our app.

I've included a background image in the *images* package of our project file.

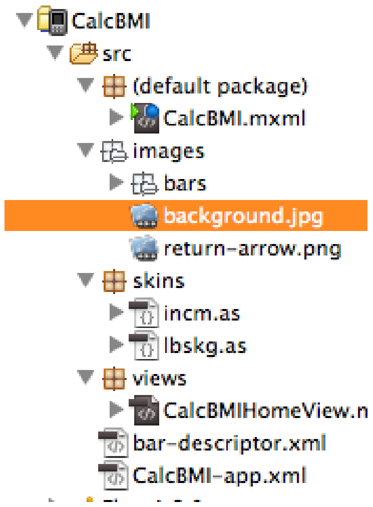

Figure 1: Background Image

We will need to create a *view/screen* skin, that includes our background image, and then apply that skin to our view. So, let's start this lab.

Right click the *skins* package, and select *New > MXML Skin.*
Name it *BackgroundSkin.*
Use the host component *spark.components.Application.*
Create as a copy of *spark.skins.spark.ApplicationSkin*.
Remove all Actionscript styling. (We'll be using XML to code this skin. It's much easier to read and edit.)

This is an exact copy of the default skin. It allows you to see what properties can be modified in the view skin. Read through this file, to get a basic understanding of what properties can be edited in the view skin.

Since we already load the default skin with the [Host Component], there is no need to load it all. So, once you've read through the file, delete its entire contents, and replace it with the code found in *backgroundSkin.txt* (included in this lab's folder). You'll notice that with this code, we overwrite the fill property with a bitmapimage function, and load our image into it. We then create a group, (content group), on top of this image. We will place our views content into this group, so it is ontop of the image. (Not behind it, where it can't be seen.) Save your skin.

Now, to apply your skin to our view, open up *views.CalcBMIHomeView.*

Find where your ViewNavigator is defined. (This call defines the view Array for this view. The original view array is 2 dimensional) It should look like this:

```
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark" firstView="views.CalcBMIHomeView"
applicationDPI="160">
```

Add the property *skinClass*, and give it the value "*skins.backgroundSkin*". The modified code should look like this:

```
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark" firstView="views.CalcBMIHomeView"
applicationDPI="160" skinClass="skins.BackgroundSkin">
```

Run it. If all went well, it should look something like Figure 2.

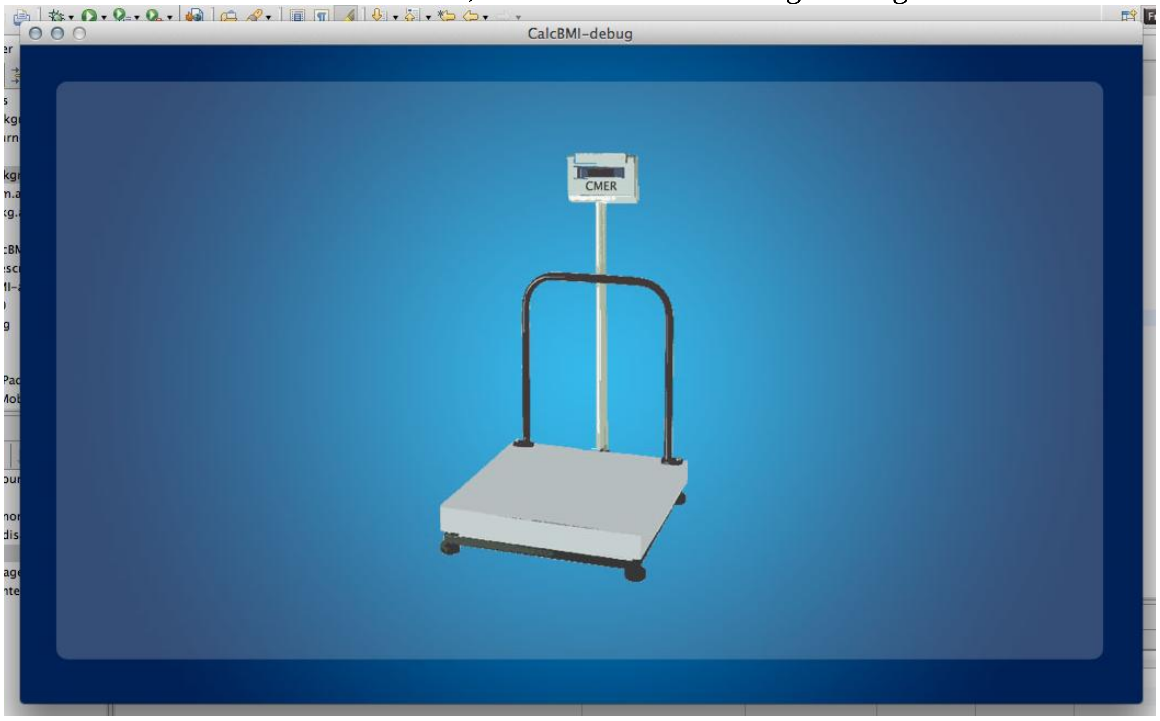

Figure 2: Background Applied

Calculator

Now that you have the base of your application set up, we need to create some functions that will handle the calculations for us. Create a new package, inside your *src* folder, called *functions*. Then, inside your functions package, create a new actionscript class called *calculate*. (Note: By convention, AS classes normally start with an uppercase. However, since I will be treating this class as a function, and not a graphical asset or object, I perfer to keep it lowercase so I can differeniate between what will act as tools, and what will act as manipulatable objects). Replace the entire code in this class, with that found in *calculate.txt*. Refer to the comments for explanations of the code.

Final Functionality and GUI Set Up

Now, return to *views.CalcBMIHomeView*, and paste the code found in *insideviewtags.txt*, inside the <*s:View*> tags. The code inside <*fx:script*> is ActionScript, and the rest is XML. XML allows use to make use of Flex's predefined components very quickly, and easily, which is why it is the perfered over ActionScript when assembling GUI's. To demonstrate this, I've presented to different animation techniques in the code. The *move* animation, is implemented with ActionScript, whereas the *Fade* animation, is implemented with XML. Take a look at both, to familarize yourself with the two. ActionScript allows for more customization, where XML allows you to teak predefined properties. Also notice that I've applied custom skins to the toggle switches, that allow me to change the on/off labels to in/cm and lbs/kg.

Refer to the comments in the code for further explanations about the code.

Why Animations?

Animations are important in GUI's. Having your graphical assets appear slowly, allow the user to notice where things are occuring, and gives them time to divert their attention. They are key to any dynamic GUI, and can be implementing in many ways, a few of which were introduced in this lab.

Test Yourself

1. Slow down the fade in, and movement animation by a full second
2. Add a new button to the GUI, using XML
3. Create another view
4. Add functionality to the created button, by popping the current view, and pushing your newly created view