Developing Apps for BlackBerry PlayBook using Adobe AIR

Lab # 2

Learning to Use PlayBook Libraries and Deployment Tools

The objective of this lab is to guide you in using PlayBook specific libraries in your Flash Builder Project. It will continue where Lab #1 left off, by adding a native (to the Playbook device) slide-down menu that can be accessed by sliding your finger from the top downwards towards the screen. To give this menu a purpose, we will add some functionality to the program, that will allow the user to add or remove a ball from the stage.
A small self-test is included at the end of this lab to test your understanding of its contents

This guide continues where AS3 – Lab 1 left off. You should have a working application that looks similar to Figure 1.

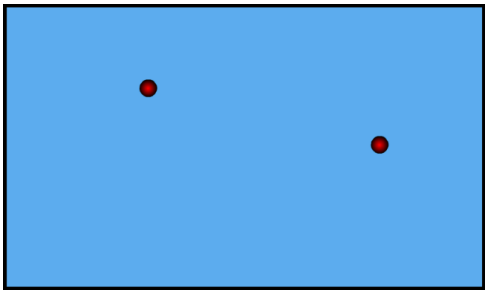
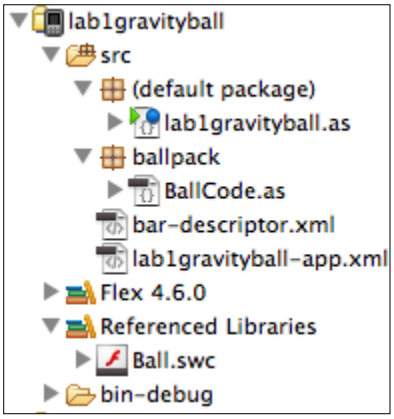


Figure 1: Balls that bounce

And the file structure of your app should be similar to Figure 2.



We will be adding a native BlackBerry PlayBook swipe down App Menu, the ability to add balls (to a max of 10), and the ability to remove balls (counter cannot go below 0) through buttons on the menu, and an onscreen textbox that will display the amount of balls on screen.

Start by opening your Lab1 project file in Flash Builder.

New Variables and Libraries

The addition of our onscreen textbox is going to require the addition of some libraries, so the compiler knows what functions we are calling, and variables. Since they are related to the balls, it's a good idea to contain the textbox in the same class the balls are located.

Open the *BallCode* ActionScript class located inside your *ballpack* package.

Find your *import* space. It should be inside the package, and outside of the class. Add the following libraries to your package:

```
import flash.text.TextField;
```

```
import flash.text.TextFormat;
```

We want our text field variables to be accessible by every function within our *BallCode* class. Recalling from Lab1, that means our variables should be declared within the class declaration, and outside all functions. You can place them right underneath our other variables. I've declared them like so:

```
private var txt:TextField = new TextField;
private var txtFormat:TextFormat = new TextFormat();
```

txt represents the variable that will track our TextField.
txtFormat will contain the format we will declare for our TextField.

The Constructor Function

Recall from Lab 1, that the constructor function is a function that automatically runs when it's parent class is instantiated. In the first lab, we didn't want anything to happen when we instantiated our BallCode class, however, this lab is different. We want to add our on screen ball counter (text field) to the stage when the class is instantiated. Find your BallCode constructer function, and add the following functionality to it.

```
public function BallCode()
{
	//Assign a size to our format
	txtFormat.size = 26;
	//Assign text to the field
	txt.text = String(numofBalls) + " balls";
	//Disable interaction with our field
	txt.mouseEnabled = false;
	//Assign our created format to the field
	txt.defaultTextFormat = txtFormat;
	//Set the location of the field
	txt.y = 555;
	txt.x = 900;
	//add field to the display list
	addChild(txt);
}
```

addBall() function

Notice that we already have an addBall function. We used it in the first lab, to originally add the two balls to stage.

Modularity, (according to Wikipedia) is the degree to which a system's components may be separated and recombined. One example of modularity is function reuse, and that is a major component to create effective and clean code using Object Oriented Programming(OOP). We'll implement the idea of function reuse in this lab.

Since we will be using the function addBall to add more balls via a button on our soon-to-be programed menu, we'll need to review it to ensure it is sufficient to do what we want. The function is perfect for adding balls, however, if it is called to much – say 100 times – we could end up with 100 balls on screen which may cause a lot of lag and crash our application. We'll need to implement some defensive programming in order to prevent this from happening. Although there are many ways to do this, the easiest way to do this, is to wrap the entire block of code in and if statement. Finally, when a ball is added, we also want to update our label so it displays the correct number of balls on screen. You can do it like this:

```
public function addBall():void
```

```
{
        if (numofBalls < 10)
        {
        ball = new Ball();
        ball.x = Math.random() * (1024 - pD) + pR;
        ball.y = Math.random() * (600 - pD) + pR;
        ball.speed = 12;
        ball.angle = Math.random() * 360;
        updateballVelocities(ball);
        addChild(ball);
        balls[numofBalls] = ball;
        numofBalls++;
        //Update text field
        txt.text = String(numofBalls) + " balls";
        }
        else
        { //Do nothing
        }
}
```

This means the function will run as long as there are less than 11 balls on screen (remember, 0 counts as a ball, so 0 to 9 is equal to 10 balls). This function can now be used to add two balls to the screen on app launch, as well as when the user wants to add balls on the screen.

removeBall() function

Unfortunately, creating this function is not as simple as reusing existing functions, as we do not have a function that removes balls from the screen. So, let's create a removeBall() function inside `public class BallCode extends MovieClip { //in here }`

ActionScript functions work by declaring them like so:
<Access modifier> function **<name>** (**<parameters>**):**<returned var type>** { **//code block** }

This means our removeBall() function should look something like:
`public function removeBall():void { //code block }`

Now, all we need to do is fill the code block. Refer to the following code, reading the comments for explanation.

```
public function removeBall():void
{
        //defensive if
        if (numofBalls > 0)
        {
        //decrease ball count
        numofBalls--;
        /*assign the ball to be removed
        to a variable so it can be accessed*/
        ball = balls[numofBalls];
        //remove ball from display list
        removeChild(ball);
        /*"pop" last element, our ball
        off the balls array. We no longer
        need it, and this recovers our
        used memory*/
        balls.pop();
        //update text field
        txt.text = String(numofBalls) + " balls";
        }
        else
```

```
                { //Do nothing
                }
        }
```

Now that we're done adding all our functionality to our BallCode class, we can move on and start creating our menu. But first, you've done a lot of work. And you probably want to test what you've done so far. So, why not? Open the simulator and run your code. You should see something like Figure 3.

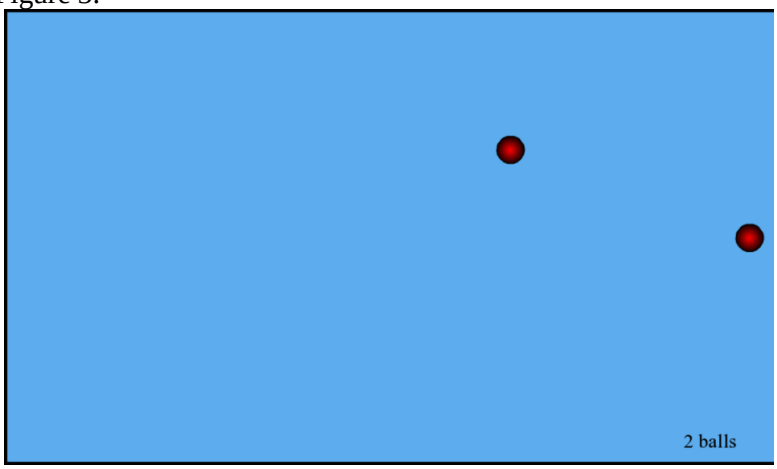


Figure 3: Ball Counter

Notice the ball counter in the bottom right? Cool.

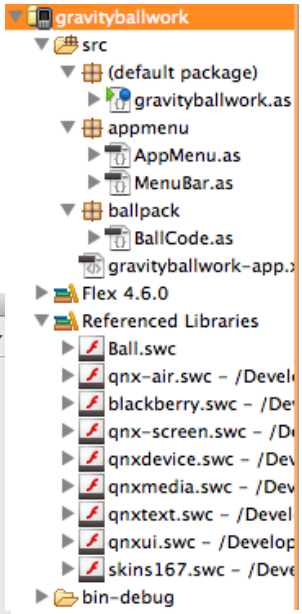
But our addBall and removeBall functions still have no method of being called. Let's build that menu we were talking about.

The PlayBook slide down App Menu

The PlayBook features a native menu, that users can access by sliding there finger from the top frame of the screen down.

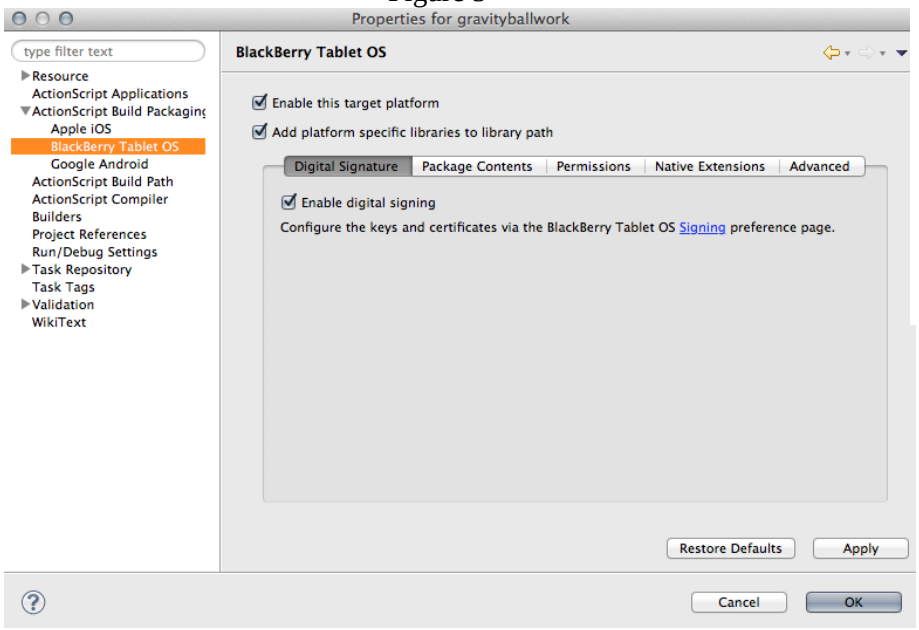
Figure 4: Swipe Down

We'll be implementing a menu like this one into our application.
First, we'll need to import the PlayBook Libraries into our project. (This guide assumes you've installed the PlayBook SDK, and have chosen to "Implement in Flash Builder" during the install)



To do this, Right Click your project folder in the *Package Explorer*. Select *Properties*. Under *ActionScript Build Packaging*, select *BlackBerry Tablet OS*. In the window, choose to *Add Platform Specific Libraries to Library Path*.
Refer to this figure 5 if you have trouble finding it.

Figure 5



This add's the BlackBerry Library .swc's to you project, and we can now import the PlayBook libraries into our classes. We'll need to do this to create our menu.

File Structure

Start by creating a new package called *appmenu*, inside your *src* folder. Then, create two classes inside your *appmenu* package, one called *AppMenu,* the other called *MenuBar*. When you're done, your Project File structure should look like figure 6.

Figure 6: File Structure

Coding The Menu

Place the code from *AppMenu.txt* into *AppMenu.as*.
Place the code from *MenuBar.txt* into *MenuBar.as*.

Refer to the comments in the code for explanation on how it works.

Now that we have set up our Menu Classes, we have to instantiate them, in our document class.

The Document Class

Open up your document class. Recall from Lab 1, this is your .as file inside your *default package*. Most likely, it is named *lab1gravityball.as*, (referring to the file structure image – mine is called *gravityballwork.as)*.

First, we need to import our AppMenu class into our document class so we can call it.

```
import appmenu.AppMenu;
```

Next, we need to declare a class wide variable, as well as instantiate our AppMenu.

```
private var appMenu:AppMenu = new AppMenu(ball);
```

Notice that we are passing it our variable that represents our BallCode class. This gives it access to the BallCode functions.

Finally, we want to add our AppMenu to the display list. We do so by adding the following to the constructor function:

```
addChild(appMenu);
```

(Your construction function is the *public function <classnamehere> ())*

Testing your Application

Unfortunately, we cannot test our application through the simulator/emulator, because our computers do not have touch screens. (Well, mine doesn't.) This means, that it cannot interrupt the "gesture swipe" event. In order to test our application, we will need to build it, sign it, and deploy it. Refer to the instructions in Lab 1, if you need help building, signing, or deploying.

If you've followed the lab correctly, you should have a working PlayBook App, with the ability to swipe down to add and remove balls.

Test Yourself

Complete the following to test your understanding of the code

1. Change the background color of your swipe-down menu
2. Change the color of the display text " x Balls"
3. Change the function "removeBall" to remove ALL balls on stage. Remember to maintain defensive coding (*numofballs* cannot be less than 0).