



Adobe

Adobe MAX 2011
Multi-Device Best Practices

BYOD Lab

Written & Developed by Andrew Trice

Contents

Introduction3
The Files3
Preparation3
Getting Started – Create Your Mobile Project.....4
Device Orientation & Form Factor.....7
Network Status Detection..... 17
CSS Media Queries 21
Multi-DPI Images..... 24
Conclusion..... 25

Introduction

Welcome to Adobe MAX 2011! This BYOD Lab “Multi-Device Best Practices” was developed to highlight development techniques crucial to building successful multi-device applications using Adobe Flex and Adobe AIR. By the end of this lab you will have the knowledge and skills necessary to build multi-platform and multi-form factor applications using Adobe’s mobile tools.

The Files

The source files for this project contain several images that will be used in one of the code examples. Once you reach that example within the lab workbook, you will need to copy these files into your development project.

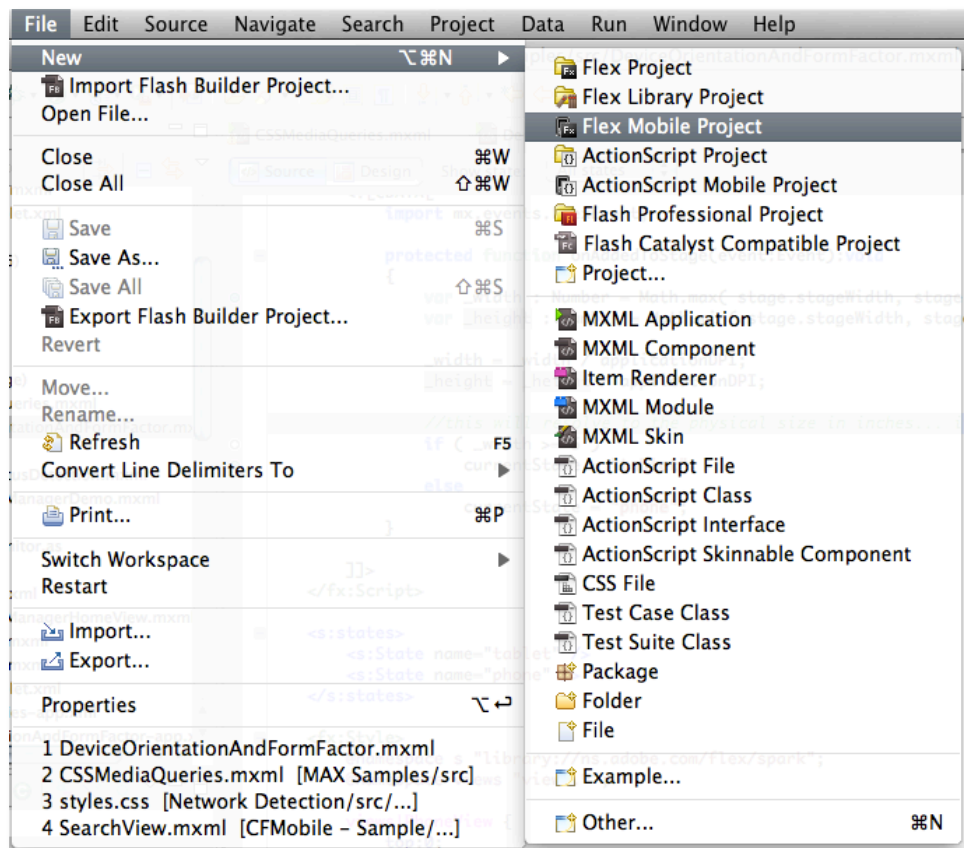
Preparation

To get the most out of this lab you should have a familiarity with Flex and ActionScript. If you wish to deploy your application to your own devices, you should have all necessary provisioning files already accessible, and you should already have Flash Builder installed and configured accordingly.

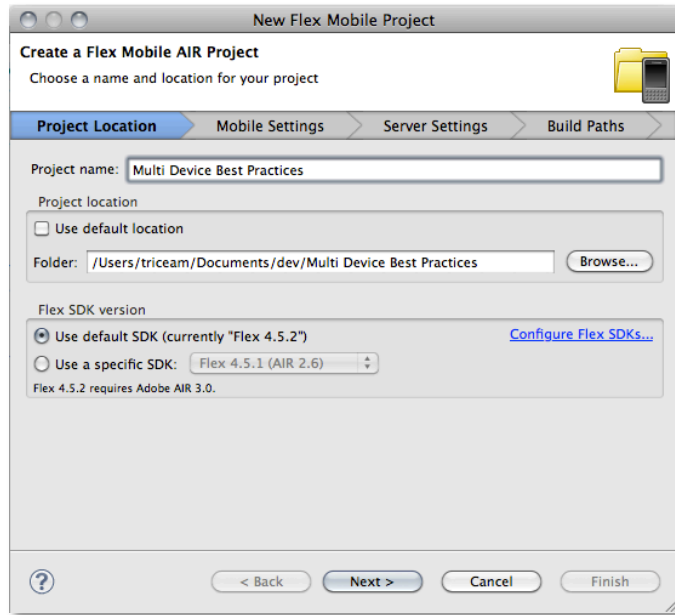
Getting Started – Create Your Mobile Project

The first thing that you need to do is create a new Flex mobile project.

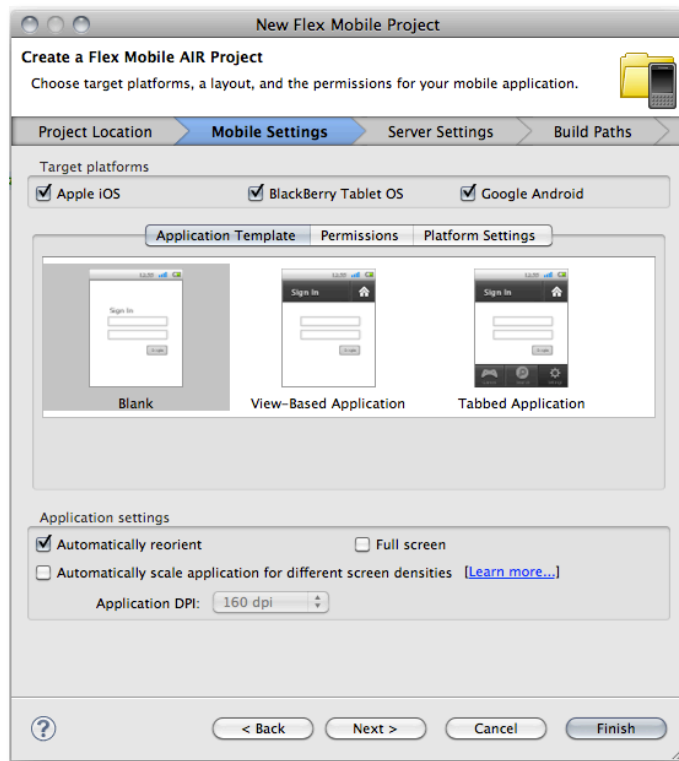
1. Launch Flash Builder.
2. Create a new Flex Mobile project by going to the "File -> New -> Flex Mobile Project" menu.



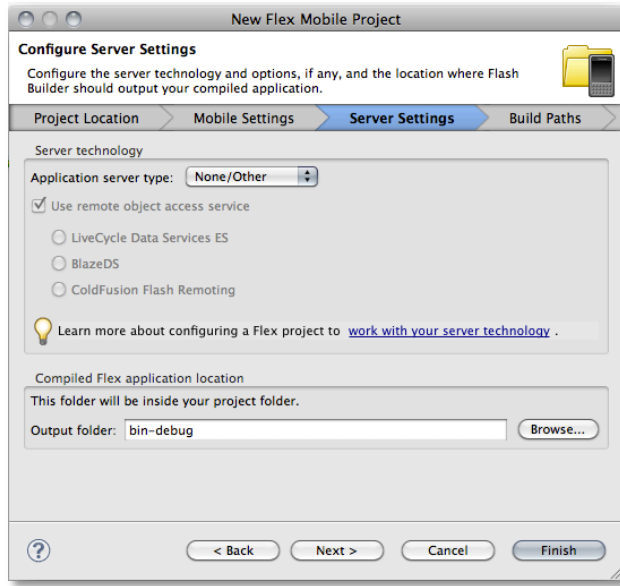
3. The new project wizard will be displayed. Enter the project name "Multi Device Best Practices" then click on the "Next" button.



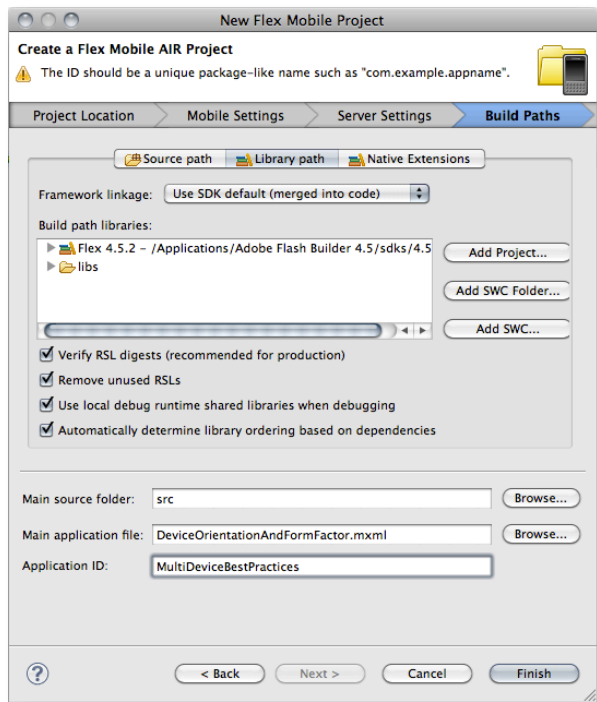
4. Select the application type "Blank" then click on the "Next" button.



5. On the "Server Settings" screen, click next without changing any options.



6. On the "Build Paths" screen, enter the main application file name "DeviceOrientationAndFormFactor.mxml", and enter the application id value "MultiDeviceBestPractices".



Device Orientation & Form Factor

Now that your project is setup, let's examine techniques to have your application respond to device form factor and orientation. We will build an application that has two distinct views between phone and tablet form factors, and each of those views will respond to landscape and portrait orientation.

1. The first thing that we will do is to create two view states: 1 for the tablet form factor and one for the phone form factor. We will use this to demonstrate a global application that has separate UI layouts for phone and tablet devices.

```
<s:states>
    <s:State name="tablet" />
    <s:State name="phone" />
</s:states>
```

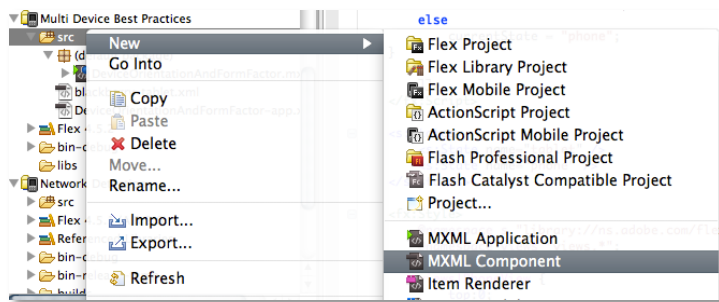
2. Next, let's add an "addedToStage" event handler on the root application. In this event handler we will evaluate the stage pixel size, divided by the applicationDPI. This will give us a general estimate of the physical dimensions of the device. In this example, I assume that if the physical dimension is greater than 5 (5 inches), then the view state "tablet" will be used. If the physical dimension is less than 5, the "phone" view state will be used.

```
protected function onAddedToStage(event:Event):void
{
    var _width : Number =
        Math.max( stage.stageWidth, stage.stageHeight );
    var _height : Number =
        Math.min( stage.stageWidth, stage.stageHeight );

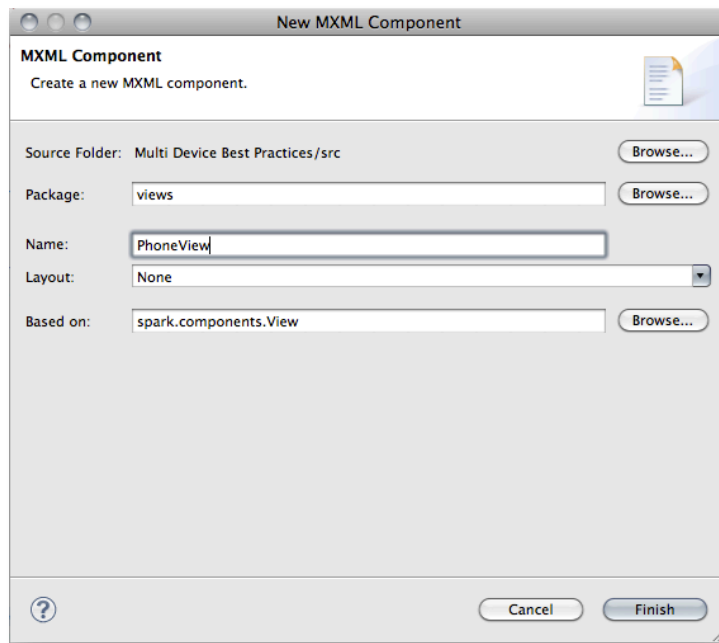
    _width = _width / applicationDPI;
    _height = _height / applicationDPI;

    // this will resolve to the physical size in inches
    // if greater than 5 inches, assume its a tablet
    if ( _width >= 5 )
        currentState = "tablet";
    else
        currentState = "phone";
}
}
```

3. Next, right click on the "src" directory in the package explorer, and go to "New -> MXML Component".



4. Create a new MXML component named "PhoneView" in the "views" package, which is based on spark.components.View (the default value).



5. After you have created the PhoneView, follow the exact same steps and create a new MXML component called "TabletView", also in the "views" package.
6. Go back to the file "DeviceOrientationAndFormFactor.xml" and add the PhoneView and TabletView components. On the PhoneView, ensure that it is only included in the "phone" view state. On the TabletView, ensure that it is only included in the "tablet" view state.

```
<views:PhoneView includeIn="phone" />
```

```
<views:TableView includeIn="tablet" />
```

7. Next, add CSS styles to ensure that both the PhoneView and TableView take up the full amount of available space within the application.

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    @namespace views "views.*";

    views|PhoneView {
        top:0;
        left:0;
        bottom:0;
        right:0;
    }

    views|TableView {
        top:0;
        left:0;
        bottom:0;
        right:0;
    }
</fx:Style>
```

8. Open the PhoneView file for editing.
9. Next, we will create a layout that adapts to device orientation. First create "portrait" and "landscape" view states.

```
<s:states>
    <s:State name="portrait" />
    <s:State name="landscape" />
</s:states>
```

10. Next, add a script element and include an updateOrientation function that will be used to update the current view state depending upon the current screen dimensions.

```
protected function updateOrientation(
    event : StageOrientationEvent ) : void
{
    this.currentState =
```

```
        (width > height ? "landscape" : "portrait");  
    }
```

11. Add an "addedToStageHandler()" function that will be invoked when the view is added to the stage. In it, call the updateOrientation function, and add an event listener for stage orientation change events.

```
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"  
        xmlns:s="library://ns.adobe.com/flex/spark"  
        title="PhoneView"  
        addedToStage="addedToStageHandler(event)" >  
    <fx:Script>  
    <![CDATA[  
  
        protected function addedToStageHandler(event:Event):void  
        {  
            updateOrientation(null);  
            stage.addEventListener(  
                StageOrientationEvent.ORIENTATION_CHANGE,  
                updateOrientation );  
        }  
    ]]>  
    </fx:Script>  
</s:View>
```

12. Next, add a s:Label instance that will only be included in the "landscape" view state and add text to it (this can be any text).

```
<s:Label top="10" left="10" width="190"  
        includeIn="landscape">  
    <s:text>Lorem ipsum dolor sit amet, consectetur  
adipiscing elit, sed do eiusmod tempor incididunt ut labore et  
dolore magna aliqua. </s:text>  
</s:Label>
```

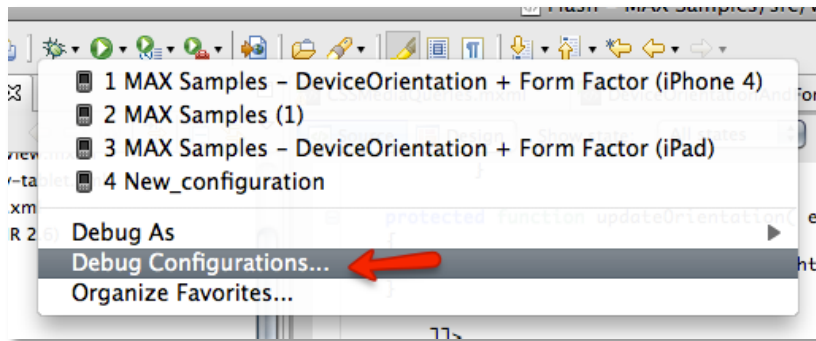
13. Add a list instance. The top, left, and bottom should all be set to zero. The left value should be set to 200 if it is in landscape orientation, and zero if in portrait orientation.

```
<s:List top="0" right="0" bottom="0"  
        left.landscape="200" left.portrait="0">  
    <s:dataProvider>  
        <s:ArrayList>  
            <fx:String>Item 1</fx:String>  
            <fx:String>Item 2</fx:String>  
        </s:ArrayList>  
    </s:dataProvider>  
</s:List>
```

```
<fx:String>Item 3</fx:String>
<fx:String>Item 4</fx:String>
<fx:String>Item 5</fx:String>
<fx:String>Item 6</fx:String>
<fx:String>Item 7</fx:String>
<fx:String>Item 8</fx:String>
<fx:String>Item 9</fx:String>
<fx:String>Item 10</fx:String>
</s:ArrayList>
</s:dataProvider>
</s>List>
```

14. Save your work.

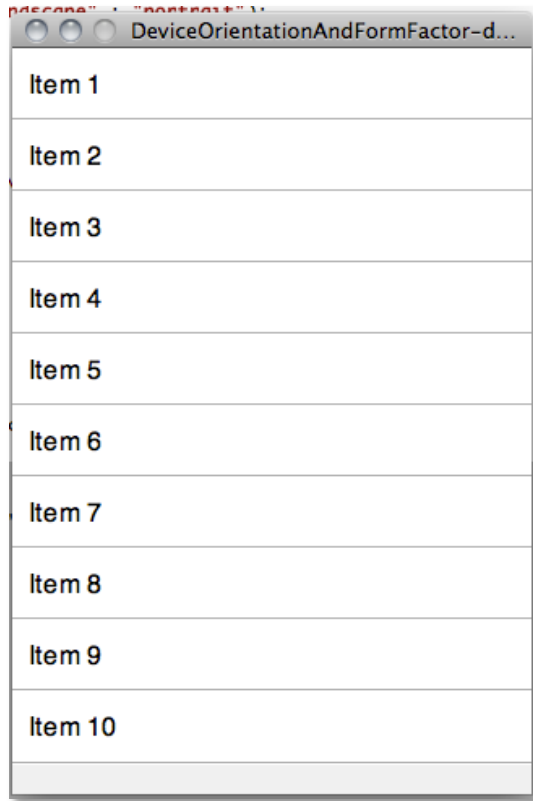
15. Click on the debug menu and select "Device Configurations".



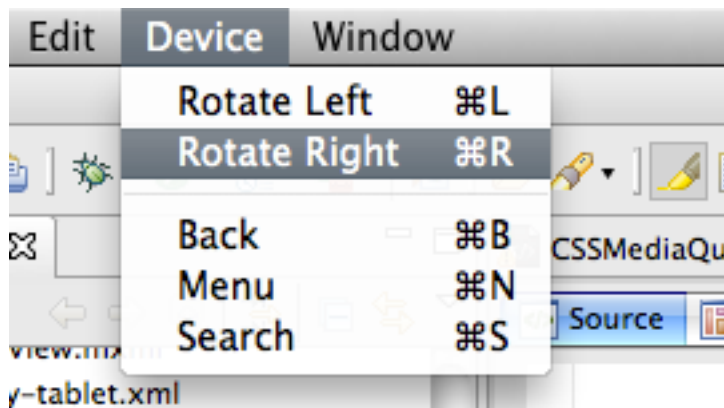
16. Create a new debug target, and select the target platform "Apple iOS", and launch method "on desktop". Then, select the device to simulate as "Apple iPhone 3GS".

17. Click on the "Debug" button to launch the simulator (you could also debug on a device here, but it will take longer, depending on the platform).

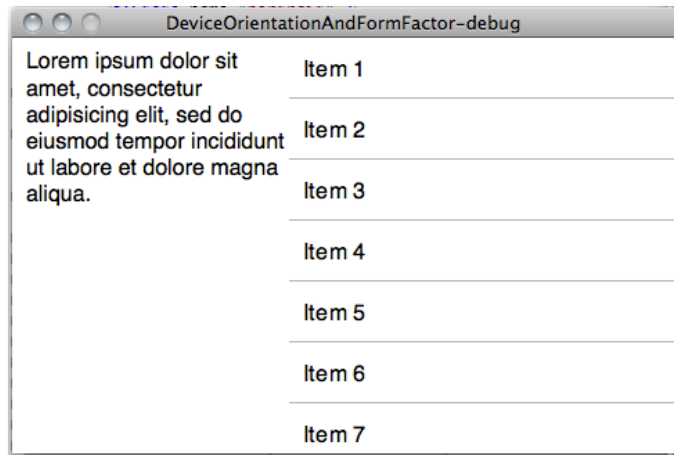
18. The application will launch in landscape orientation. You will see that the list is visible.



19. With the simulator active, go to the "Device -> Rotate Right" menu option.



20. This will change the simulator orientation to landscape, and you will see the layout update to reflect the landscape view state.



21. Next, go back to your code and open the file `TableView.mxml`.
22. Add "portrait" and "landscape" view states, as you did for the `PhoneView` component:
23. Again we will add an `updateOrientation` function and `addedToStageHandler` function, as we did with the `PhoneView` component. However, in this case, also add bindable `Number` variables `halfwidth` and `halfheight`, as shown below:

```
<s:states>
  <s:State name="portrait" />
  <s:State name="landscape" />
</s:states>
```

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

    [Bindable]
    private var halfWidth : Number = 0;

    [Bindable]
    private var halfHeight : Number = 0;

    protected function
      addToStageHandler(event:Event):void
    {
      updateOrientation(null);
      stage.addEventListener(
        StageOrientationEvent.ORIENTATION_CHANGE,
```

```
        updateOrientation );
    }

    protected function
    updateOrientation(
        event : StageOrientationEvent ) : void
    {
        this.currentState =
            (width > height ? "landscape" :
             "portrait");
        halfWidth = width/2;
        halfHeight = height/2;
    }
}]]>
</fx:Script>
```

24. Next add three rectangles, each of different color. Be sure to enter the top, left, right, width, and height values as shown below. There will be very different layouts in portrait, so pay close attention. Also, you can add a label to reflect the current state (useful if debugging).

```
<s:Rect id="redbox"
    top="10" left="10"
    right.portrait="10"
    right.landscape="{ halfWidth+5 }"
    height.landscape="{ halfHeight-10 }"
    height.portrait="{ (height/3)-10 }">
    <s:fill>
        <s:SolidColor color="#FF0000" />
    </s:fill>
</s:Rect>

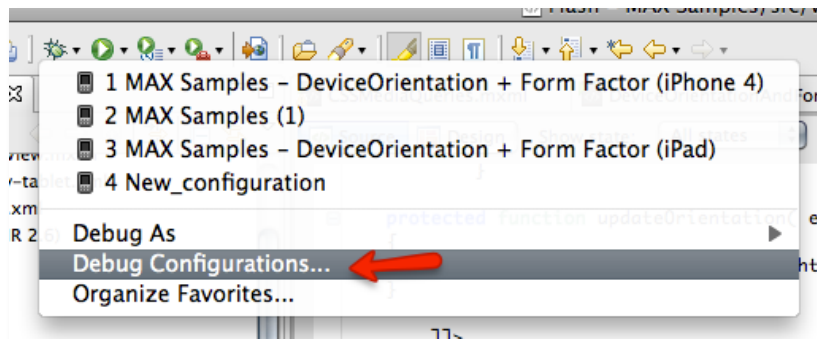
<s:Rect id="greenbox"
    top.landscape="{ halfHeight+10 }"
    top.portrait="{ (height/3)+10 }"
    left="10"
    right.portrait="10"
    right.landscape="{ halfWidth+5 }"
    height.landscape="{ halfHeight-20 }"
    height.portrait="{ (height/3)-10 }">
    <s:fill>
```

```
        <s:SolidColor color="#00FF00" />
    </s:fill>
</s:Rect>

<s:Rect id="bluebox"
    top.landscape="10"
    top.portrait="{ (2*height/3)+10 }"
    left.portrait="10"
    left.landscape="{ halfWidth+5 }"
    right="10"
    bottom="10">
    <s:fill>
        <s:SolidColor color="#0000FF" />
    </s:fill>
</s:Rect>

<s:Label text="{currentState}"
    top="15" left="15"/>
```

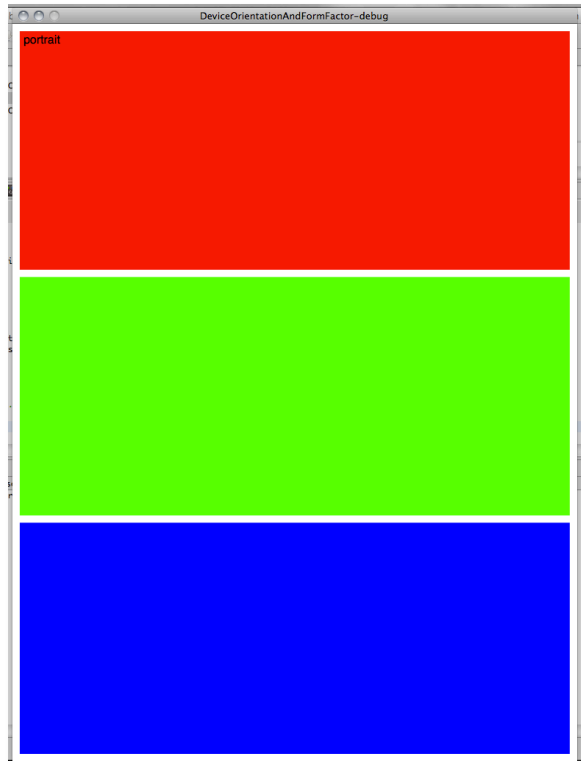
25. Click on the debug menu and select "Device Configurations".



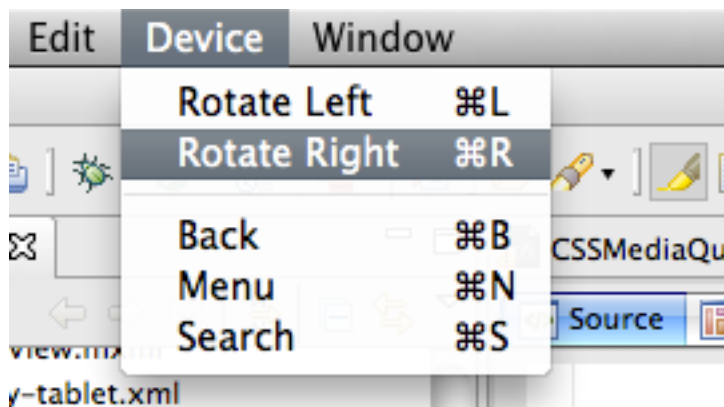
26. This time, we will test as a tablet device. Create a new debug target, and select the target platform "Apple iOS", and launch method "on desktop". Then, select the device to simulate as "Apple Pad".

27. Click the "Debug" button to launch the application.

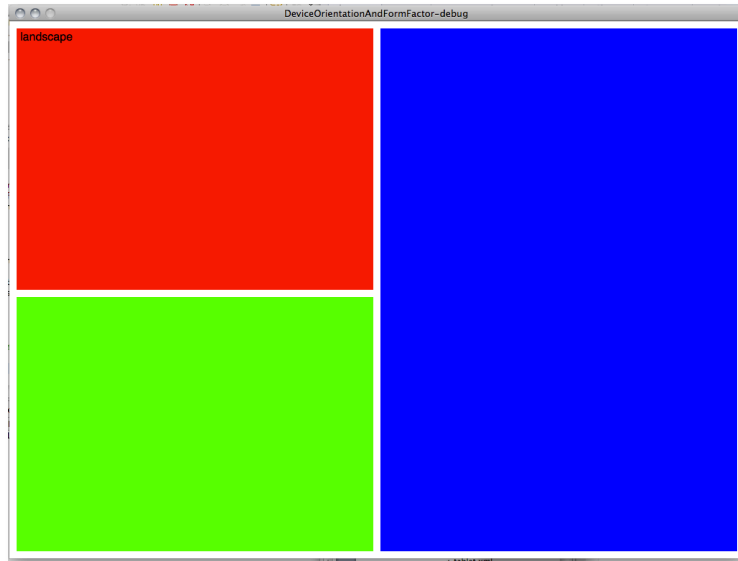
28. In portrait mode, you will see 3 vertically layered rectangles.



29. With the simulator active, go to the "Device -> Rotate Right" menu option.



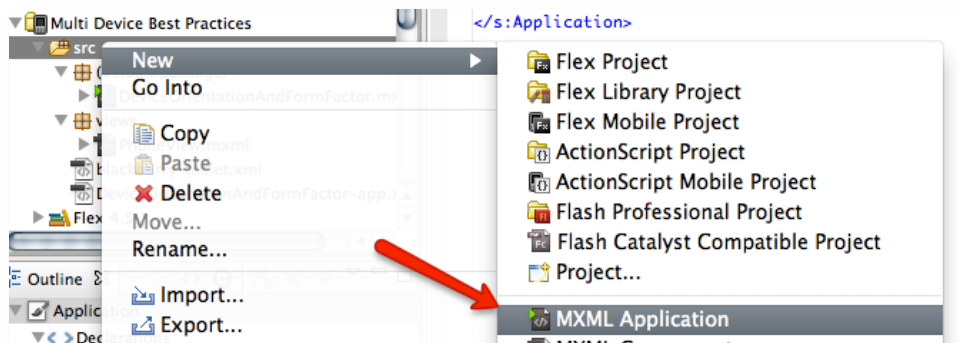
30. This will change the simulator orientation to landscape, and you will see the layout update to reflect the landscape view state. Notice that the layout of the rectangles has shifted significantly to better utilize the form factor offered by a tablet device.



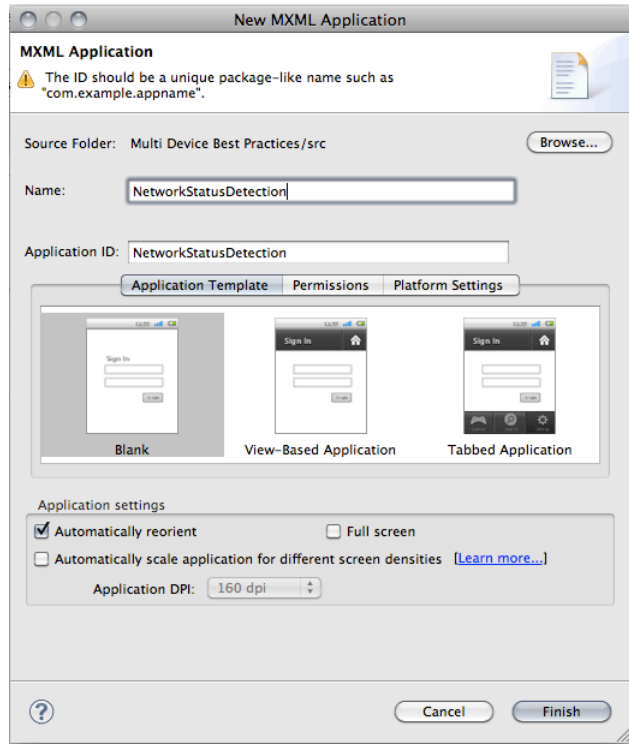
Network Status Detection

When working with multi-device scenarios, you also often run into cases where your applications lose network connectivity. This is especially the case with team members who are “out in the field” with mobile devices. In these cases, your application needs to respond accordingly to prevent data loss. This scenario will demonstrate how you can monitor network connectivity for use in mobile applications.

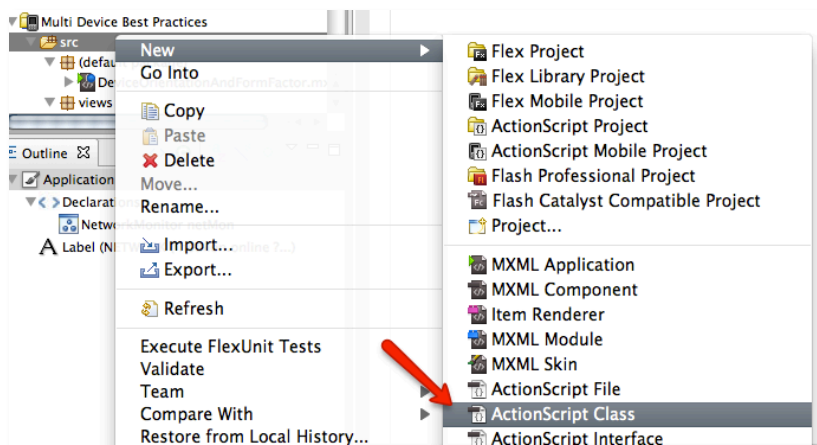
1. Right-click on the src folder in the package explorer, and go to “New -> MXML Application”.



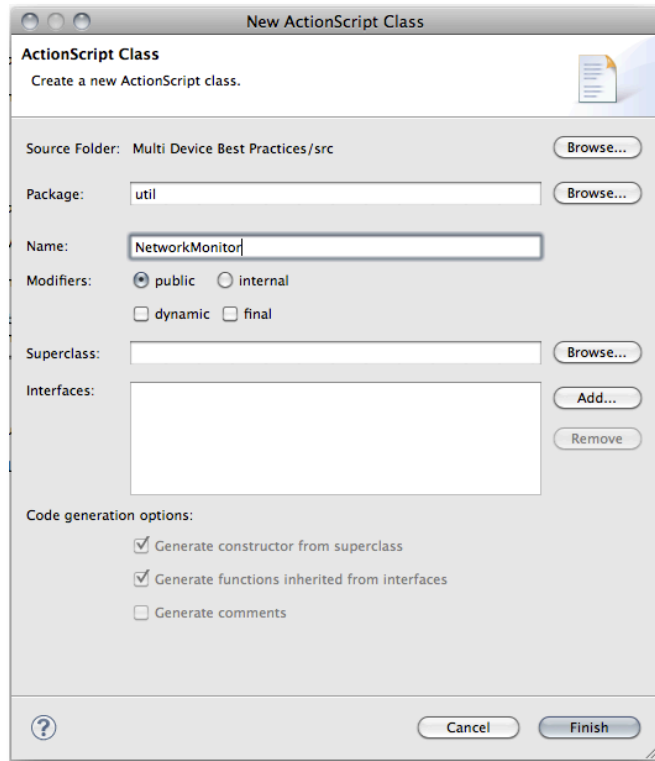
2. Enter the name “NetworkStatusDetection” and select the application template “Blank”, and click “Finish”.



3. This will create a new application for this scenario. Before we add anything to the UI let's create a utility class that we can use to monitor the network status. Right-click on the "src" folder in the package explorer and go to "New -> ActionScript Class".



4. Enter the package "util" and the Name "NetworkMonitor", and click Finish.



5. Inside of this class, add a public variable "online" that will be used to identify online/offline status, and also setup an AIR URLMonitor instance that can be used to monitor network connectivity changes. I used a polling interval of 10 seconds to monitor active network connections.

```
public class NetworkMonitor
{
    [Bindable]
    public var online : Boolean = false;

    protected var monitor:URLMonitor;

    public function NetworkMonitor()
    {
        monitor = new URLMonitor(
            new URLRequest('http://www.adobe.com'));
        monitor.addEventListener(
            StatusEvent.STATUS, onStatusUpdate );
        monitor.pollInterval = 10000;
        monitor.start();
    }
}
```

```
protected function onStatusUpdate(  
    event : Event ) : void  
{  
    online = monitor.available;  
}  
}
```

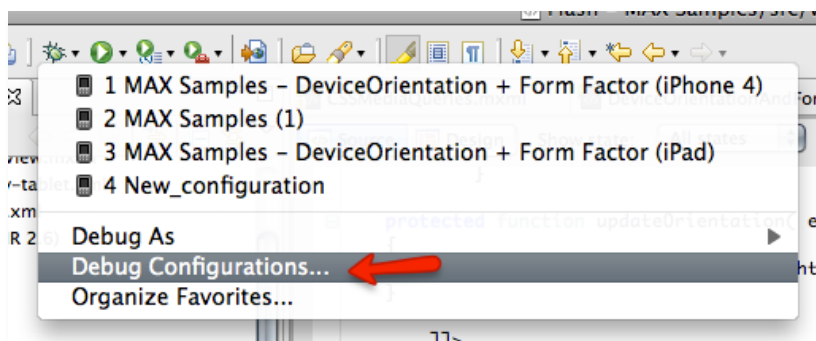
6. Go back to the "NetworkStatusDetection.mxml" file, and create an instance of the NetworkMonitor within the fx:Declarations tag.

```
<fx:Declarations>  
    <util:NetworkMonitor id="netMon" />  
</fx:Declarations>
```

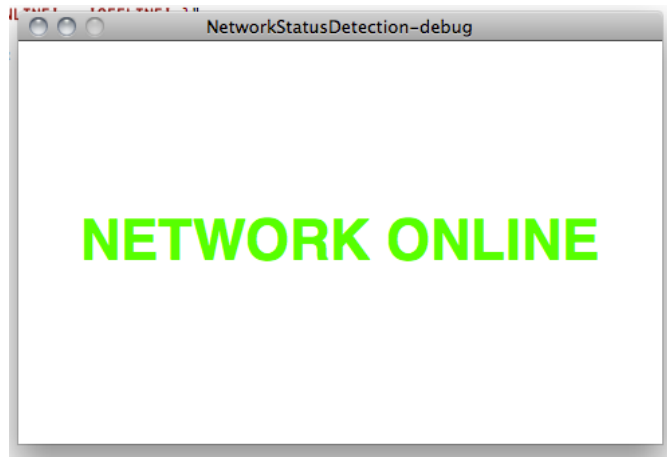
7. Now, add a label that will show the online/offline state of the application – This is just used to visibly show your application responding to network changes.

```
<s:Label text="NETWORK {netMon.online ? 'ONLINE':'OFFLINE'}"  
    fontSize="42" fontWeight="bold"  
    color="{ netMon.online ? 0x00FF00 : 0xFF0000 }"  
    horizontalCenter="0"  
    verticalCenter="0"  
    maxWidth="{ width }"  
    textAlign="center" />
```

31. Save your work and go to "Debug Configurations".



8. This time, select your application file "NetworkStatusDetection" and click the "Debug" button. This will launch your application for testing. If you have a device configured, you can launch this on your device as well.



CSS Media Queries

Another important factor for taking advantage of multiple devices, form factors, and platforms is that you can use CSS media queries to define screen or platform specific styles. This can be as simple as color changes per platform, button styles per platform (such as iOS button styles), font and layout changes per screen size, or even changing skin classes.

1. Right-click on the src folder in the package explorer, and go to "New -> MXML Application".
2. Enter the name "CSS Media Queries" and select the application template "Blank", and click "Finish".
3. Next, add a very simple UI: a button whose label displays the current applicationDPI value.

```
<s:Button label="Device DPI: { applicationDPI }" />
```

4. Next, add CSS styles to change the button text color based on application-dpi. Color changes per dpi may not always be useful, however this visual demonstrates the application of different styles based on different resolutions.

```
<fx:Style>  
    @namespace s "library://ns.adobe.com/flex/spark";  
  
    /* DPI SPECIFIC STYLES */  
    s|Button  
    {
```

```
        color: #000000;
        fontWeight: bold;
    }

    @media (application-dpi: 240)
    {
        s|Button
        {
            color: #FF0000;
        }
    }

    @media (application-dpi: 320)
    {
        s|Button
        {
            color: #0000FF;
        }
    }

</fx:Style>
```

5. Next, add platform specific styles. In this case, we will change the background color per platform.

```
/* PLATFORM SPECIFIC STYLES */
@media (os-platform:"IOS")
{
    s|Application
    {
        backgroundColor:#FFCCCC;
    }

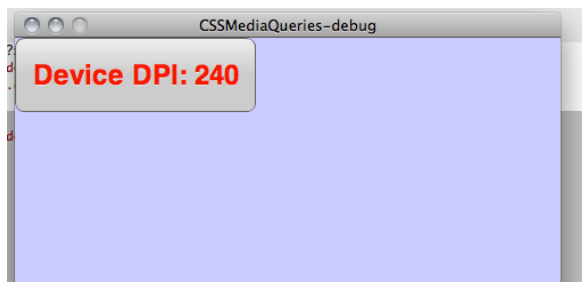
    s|ActionBar
    {
        defaultButtonAppearance: beveled;
    }
}

@media (os-platform:"Android")
{
    s|Application
```

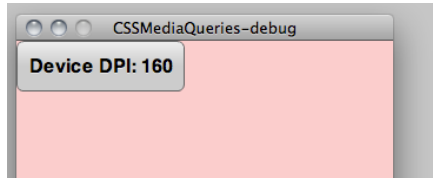
```
    {  
        backgroundColor:#CCCCFF;  
    }  
}
```

6. Save your work, and now its time to debug. If you have devices, test it on as many as you can to see the results. You can also create multiple debug configurations, targeting different platforms and sizes, and you can see how the UI responds. Below are a few:

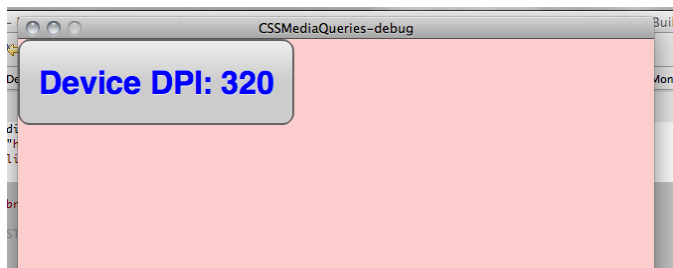
Simulated HTC desire:



Simulated iPhone 3Gs:



Simulated iPhone 4:



Multi-DPI Images

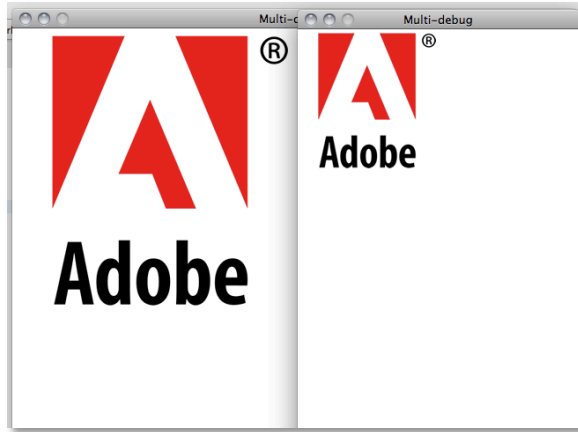
Again, when working with multiple devices, you deal with multiple sizes and multiple form factors. Here's a really basic and simple way to display images that actually work with the dimensions of your device.

1. Right-click on the src folder in the package explorer, and go to "New -> MXML Application".
2. Enter the name "MultiDPIImages" and select the application template "Blank", and click "Finish".
3. Copy the images folder from the source assets zip file into the your Flex Mobile project, making sure to maintain the "images" folder.
4. Next, create a simple s:Image instance. However, set the source to a s:MultiDPIBitmapSource object, and be sure to set the dpi-specific images.

```
<s:Image id="myImage">  
  <s:source>  
    <s:MultiDPIBitmapSource  
      source160dpi="images/160px-AdobeSystems.png"  
      source240dpi="images/240px-AdobeSystems.png"  
      source320dpi="images/320px-AdobeSystems.png"/>  
  </s:source>  
</s:Image>
```

5. Save your work, and create multiple device configurations to test with (or test on as many devices as you can). You will see that devices which have different DPI values will display different images, but will be more inline with the actual physical interface of the device.

An example of this is shown below. The logo on the left side of the screen is in a simulated iPhone 4 interface, where the image on the right is in a simulated iPhone 3Gs interface. The images are proportionate to the visual area of the device interface.



Conclusion

You should now be well on your way to understanding and building your own multi-device applications. These will help you with layout and environment, but you should also remember that generally mobile devices are not as powerful computing machines as desktop devices. You will always want to minimize computational overhead, and architect your application so that it doesn't have to do extra work or data processing if it really doesn't need to.