Adobe Flex 4 Tutorial 2010

A Tutorial from Technical Evangelism @ Adobe

Version Date: Jan 1, 2010

Synopsis:

This course is to introduce developers to the Adobe Flex 4 SDK and Flash Builder 4 IDE in conjunction with Flash Player and AIR. Attendees will learn all about the development environment and provide them with an overview of the unique features of AIR 1.5 and Flex 4. This document is the teachers guide There are several other documents that go with this course including the Student handouts that may be used as a self paced tutorial.

License:

Table of Contents

Forward:

Welcome to the 2010 Adobe Flex and AIR Code Camp. This course has been put together in hopes to provide developers a boot camp to learn all the basics of Adobe Flex and AIR development in as short as time as possible. Although we tried hard to cover everything, it was simply not possible in such a short time. Our overall goal is to provide you with an introduction to Flex 4 so you can make your own decision if you want to pursue this exciting new application development technology in the future. If you do, we will be providing additional references where you can continue learning and become part of the larger community after this course is over.

IMPORTANT: There is a high probability we will not get through the entire course during the time allotted. This is by design. The course reflects a best case scenario whereby everyone covers the materials quickly. We felt it better to have extra rather than not enough content. If we do not get through the entire course, you can take the remaining labs by yourself as this instructional handout has sufficient notes to complete everything.

We hope you enjoy this course as much as we enjoyed putting it together. Remember – we are here for you. Don't hesitate to ask any questions during the event and afterwards.

Audience Assumptions

- This is a Boot Camp to introduce people to the Adobe Flash Platform who have never used it before or want to go past Hello World. The projects are designed to maximize learning of concepts in as short as time as possible.
- Attendees have Adobe's AIR 1.5.2 runtime and Flash Builder 4 or later installed.
- Attendees are roughly familiar with XML syntax rules

Preparation

There is a separate preparation guide which you will need to complete this course. It is available at http://www.web2open.org/ABC2009/ABC2009/

Other resources are available there as well.

Agenda

An overview of Flex/AIR/Flash

Architecture

Runtime Architecture

Development environment

Deployment architecture (App descriptor file, how to export AIR apps)

Coding (90%)

Project 1: An Advanced Hello World

Build project, compile and run it. This will be done from scratch so there is no starter code in the project.

This file is under the ~/completed projects folder. It takes approximately 15 minutes to explain all the concepts and have all attendees build this projects from start to finish, compiling it several times to see progress.

The project will demonstrate the following concepts:

1. setting up a new project
2. differences between design and source view
3. controlling a components property from another component
4. working with Visual Controller Components (Slider, Button, etc)
5. writing an app with both MXML and AS3
6. writing a function
7. Data binding – curly braces and the [Bindable] keyword
8. Events – calling the function based on an event

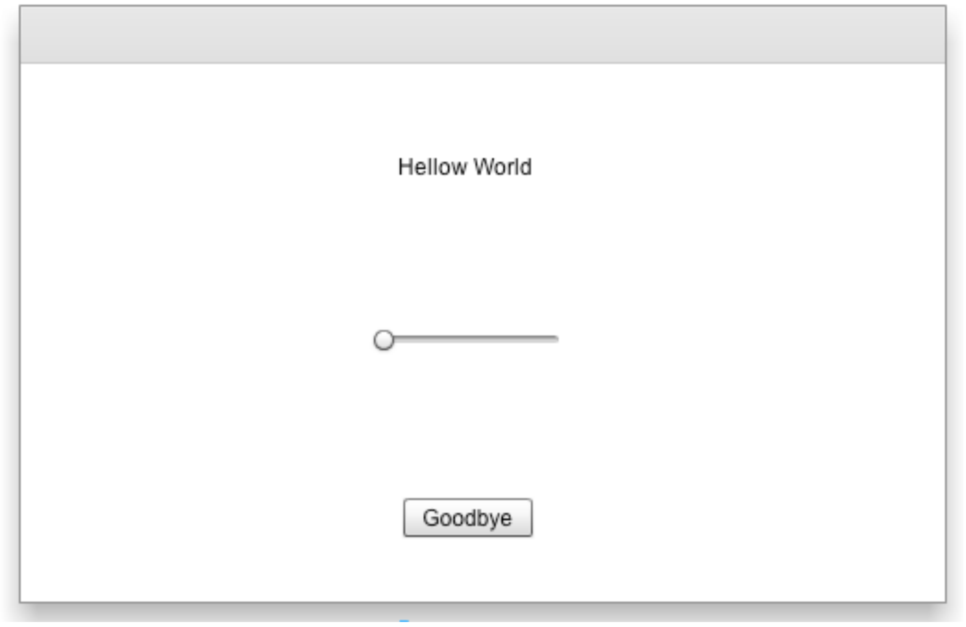
Instructions:

1. Start a new AIR project and name it "ABC1-HelloWorld"

2. In Design View, Drag a Panel into your project and resize it to around 500W * 400H

3. Add a label saying "Hello World".

4. Add a button and label it "Goodbye" and a horizontal slider as shown below.



5. Switch to code view and give the following components the identifiers (id's) as shown below:

```
    <s:Panel id="myPanel" x="99" y="14" width="502" height="324">
   <s:HSlider id="mySlider" x="191" y="144" />
    <mx:Label id="myLabel" x="202" y="47" text="Hellow World"/>
    <s:Button x="207" y="235" label="Goodbye" />
```

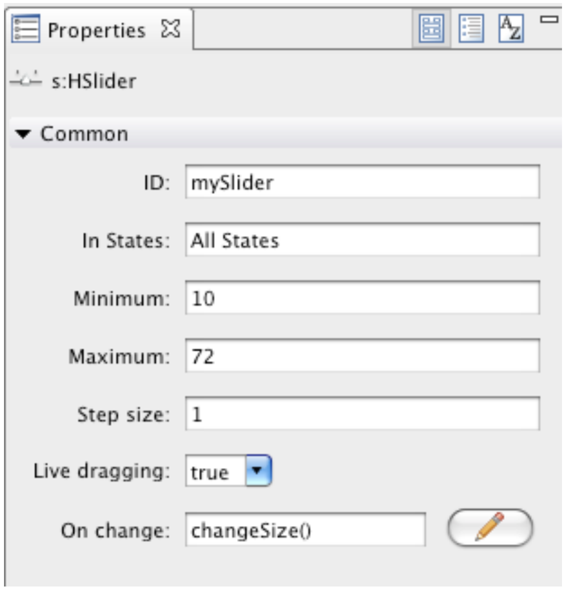
6. Make the button work by adding the following code into the `<s:Button ..>` element then run the application and hit the button.

```
    click="myPanel.visible=(false)"/>
```

7. Make a <fx: Script> block as shown below. BY CONVENTON: most times developers will only use one <fx:Script> block inside an MXML application or component and place it near the top. If you require additional code, it is best to separate concerns and adopt a more Object Oriented approach to developing AIR and Flex applications by setting up separate AS3 classes.

```
<fx:Script>
     <![CDATA[
            [Bindable]
            private var textSize:Number = 24;

            private function changeSize():void
            {
                    textSize = mySlider.value;
            }
        ]]>
        </fx:Script>
```

7. In Design mode, give the horizontal slider the following properties. This is done by first highlighting it in the design view, then using the properties inspector.



ID: mySlider
In States: All States
Minimum: 10
Maximum: 72
Step size: 1
Live dragging: true
On change: changeSize()

8. Switch to Code view and add the following attribute within the Label that says Hello World. The curly brace syntax is for data binding. It registers an event listener in the event the textSize variable changes state.

```
fontSize="{textSize}"
```

9. Within the horizontal slider element, add the attribute to call the function:

```
change=”changeSize()”
```

10. Run the application.

Project 1: Solution Code:

NOTE: The code below is formatted to fit inside the textbox. Your code looks nicer properly indented ;-)

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                          xmlns:s="library://ns.adobe.com/flex/spark"
                          xmlns:mx="library://ns.adobe.com/flex/halo">
<fx:Script>
      <![CDATA[
            [Bindable]
            private var textSize:Number = 24;

            private function changeSize():void
            {
                  textSize = mySlider.value;
            }
      ]]>
</fx:Script>
<s:Panel id="myPanel" x="99" y="14" width="502" height="324">
<s:HSlider  change="changeSize()" id="mySlider" x="191" y="144"
            minimum="10" maximum="72" stepSize="1" liveDragging="true"/>
<mx:Label fontSize="{textSize}" id="myLabel" x="202" y="47"
            text="Hellow World"/>
<s:Button x="207" y="235" label="Goodbye" click="myPanel.visible=false"/>

</s:Panel>

</s:WindowedApplication>
```

Project 1: Test Questions

In order to gauge your knowledge please complete the following test questions.

1. **Q: What keyword must be used with a variable in order that is may be referenced and updates in state detected from the MXML portion of your project?**

   A:

2. **Q: Why must ActionScript be bound in an <fx:Script> element and what must be nested immediately within that element?**

   A:

3. **Q: What must a visual component have set before it can be referenced by other parts of the application?**

   A:

4. **Q: What syntax must be used when binding a variable to an MXML component?**

   A:

5. **Q: What four parts should be present in each declaration for a function?**

   A:

6. **Q: Describe the differences between the root element "WindowedApplication" and "Application"?**

   A:

Project 2: Chrome-less applications:

The lab will contain the project outline and basic assets but attendees will need to go through the exercise from start. The project will change at runtime from an Apple shaped application to a Pear shaped application.

The following components and processes will be discussed

Importing an *.fxp project into Flash Builder 4.

stage.nativeWindow .startMove()

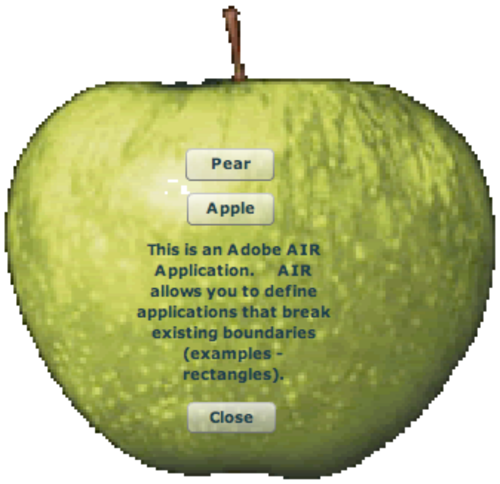
stage.nativeWindow.close()

mainPanel.addEventListener(MouseEvent.MOUSE_DOWN, startMove)

Discussion points:

Why do developers have to feed the “event” object to the event handler and why this is done.

What values might people want to read form the event object?

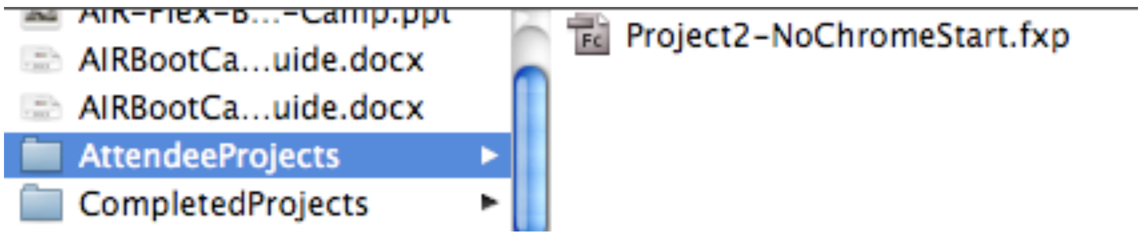

Pear
Apple
This is an Adobe AIR Application. AIR allows you to define applications that break existing boundaries (examples - rectangles).
Close

Attendees will start this by importing a project with two graphics – an apple and a pear with transparent backgrounds. Demo will first build the project, then take attendees through all the steps to making it chrome-less. These include:

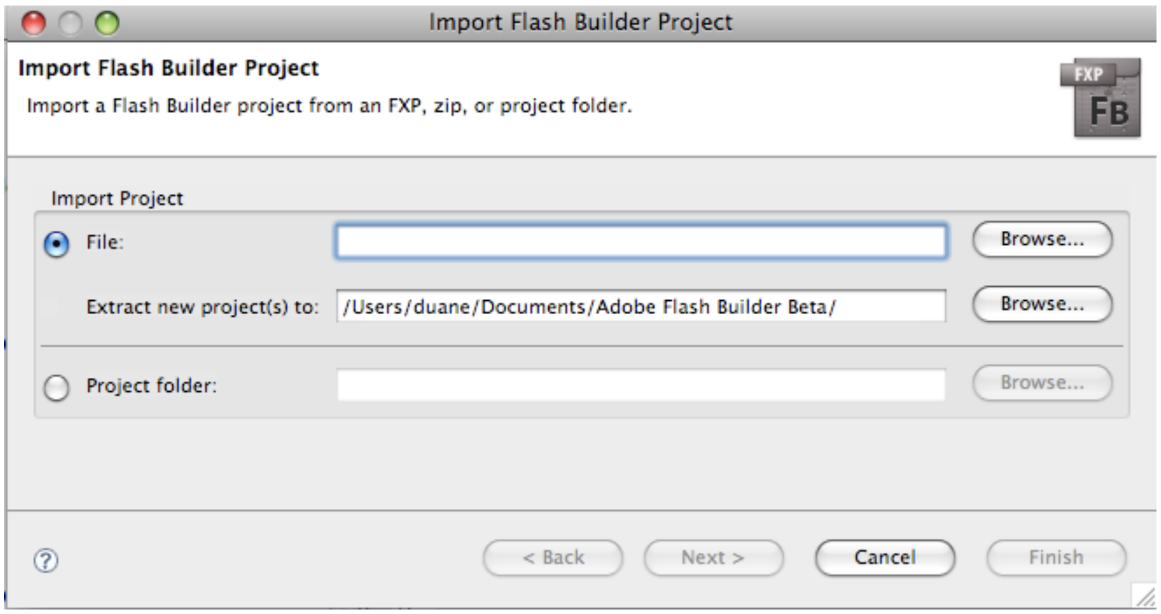
1. Modify the App Descriptor file (transparency-true, chrome=none) (describe what this does, what the file is for)
2. Adding “showFlexChrome=”false”
3. Adding the CSS style block shown below
4. How to add event handlers so you can move your app
5. How to replace the system chrome “close()” function

Instruction:

1. Import the project to Flash Builder 4 from your <root_folder >/AttendeeProjects. The project name is Project2-NoChromeStart.fxp.



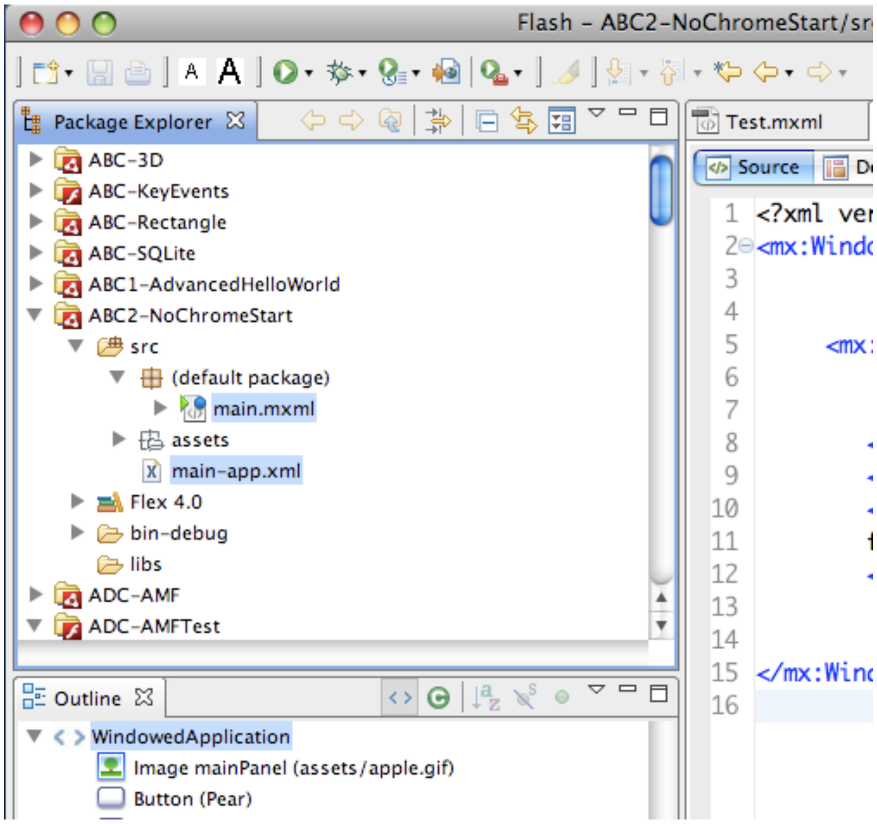
2. To import a project into Flash Builder 4, click on “File” -> “Import” -> “Flash Builder Project”. This will open up a dialog window as shown below.


Import Flash Builder Project
Import Flash Builder Project
Import a Flash Builder project from an FXP, zip, or project folder.
FXP
FB
Import Project
File:
Browse...
Extract new project(s) to:
Browse...
Project folder:
Browse...
< Back
Next >
Cancel
Finish

/Users/duane/Documents/Adobe Flash Builder Beta/

3. Click "Browse" and navigate to the project and select it, then select "Finish".

4. In Flash Builder 4, find your newly imported project under the Package Explorer and open both the main.mxml and main-app.mxml files. They are highlighted in Blue below.

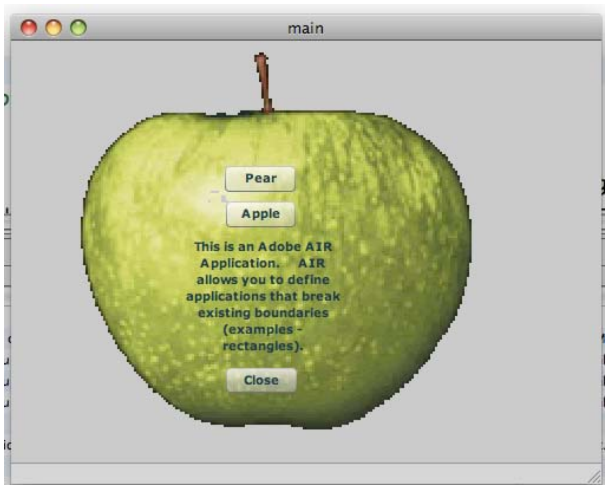


```
ABC-3D
ABC-KeyEvents
ABC-Rectangle
ABC-SQLite
ABC1-AdvancedHelloWorld
ABC2-NoChromeStart
  src
    (default package)
      main.mxml
    assets
    main-app.xml
  Flex 4.0
  bin-debug
  libs
ADC-AMF
ADC-AMFTest
```

```
<?xml ver
<mx:Windc

      <mx:

</mx:Winc
```

```
WindowedApplication
  Image mainPanel (assets/apple.gif)
  Button (Pear)
```

5. Run the project to see what happens. It should look like the image below. We will get rid of the chrome in the next steps.


main
Pear
Apple
This is an Adobe AIR Application. AIR allows you to define applications that break existing boundaries (examples - rectangles).
Close

6. In the main-app.mxml application descriptor file, find and modify these two lines of code by un-commenting them. Un commenting an XML element involved removing the <!-- before the element and the --> after it. They are usually around line 45-46 or so.

```
<!-- The type of system chrome to use (either "standard" or "none"). Optional.
Default standard. -->
        <systemChrome>none</systemChrome>

<!-- Whether the window is transparent. Only applicable when systemChrome is none.
Optional. Default false. -->
        <transparent>true</transparent>
```

7. Save and close the application descriptor file. Then try running the application. Note the ugly side effect.

8. Add the following attribute to the Root Element of WindowedApplication

```
showFlexChrome="false"
```

9. Run the project. It should look good but we still cannot move it and the close button does not work. Discussion: Why?

10. Time for some scripting to control these things. First we add the

functionality to close the application. Add this code in the <fx:Script> block:

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

        private function closeApp():void
        {
             stage.nativeWindow.close();
        }

        ]]>
        </fx:Script>
```

and add this line to the <mx:Button> to make the button call the function:

```
click="closeApp()"
```

Now ask the attendees to run it and test the close button. It should work. Next we have to deal with the moving of the application.

11. Create an init() function inside the Script element and inside the init() function, add the following lines of code:

```
        <fx:Script>
             <![CDATA[
        private function init():void
        {
             mainPanel.addEventListener(MouseEvent.MOUSE_DOWN, startMove);
        }

        private function startMove(event:MouseEvent):void
        {
             stage.nativeWindow.startMove();
        }

        private function closeApp():void
        {
             stage.nativeWindow.close();
        }

             ]]>
        </fx:Script>
```

12. then add the function call inside the <mx:WindowedApplication> element as highlighted in Yellow below:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                          xmlns:s="library://ns.adobe.com/flex/spark"
                          xmlns:mx="library://ns.adobe.com/flex/halo"
                          showFlexChrome="false"
                              layout="absolute"  creationComplete="init()">
```

13. Run your application. You should be able to move it by dragging it with the mouse!

Project 2: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                          xmlns:s="library://ns.adobe.com/flex/spark"
                          xmlns:mx="library://ns.adobe.com/flex/halo"
                          showFlexChrome="false"
                       layout="absolute" creationComplete="init()">

        <fx:Script>
              <![CDATA[
        private function init():void
        {
              mainPanel.addEventListener(MouseEvent.MOUSE_DOWN, startMove);
        }

        private function startMove(event:MouseEvent):void
        {
              stage.nativeWindow.startMove();
        }

        private function closeApp():void
        {
              stage.nativeWindow.close();
        }

              ]]>
        </fx:Script>
```

```
         <mx:Image id="mainPanel"   x="46" y="0" width="363"
                     height="343"  horizontalAlign="center"
                     source="assets/apple.gif" scaleContent="true"
                   autoLoad="true"/>

        <s:Button x="187" y="106" label="Pear" width="60"
                 click="mainPanel.source='assets/pear.gif'"/>
        <s:Button x="182" y="136" label="Apple"
                 click="mainPanel.source='assets/apple.gif'"/>
        <mx:Label text="This is Adobe AIR!" width="154" height="41"
                fontFamily="Verdana" fontWeight="bold" fontSize="14"  x="140"
                 y="189"/>
        <s:Button x="186"  y="276" label="Close" width="60"
                height="21" click="closeApp()"/>

</mx:WindowedApplication>
```

Project 2: Test

1. **Q: Are chromeless applications specific to AIR only or can they be made in Flex applications?**

   A:

2. **Q: What class is used to capture the mouse events when building chromeless applications?**

   A:
3.

4. **Q: Can you access the flash.display.stage class mentioned in question 2 directly? If not, what aspects may you control?**

   A:

5. **Q: When using addEventListener(), what two parameters must be specified?**

   A:

6. **Q: When writing a function called via an addEventListener() function, what must the function accept as an input parameter and why?**

   A:

7. **Q: How are assets referenced from within a project?**

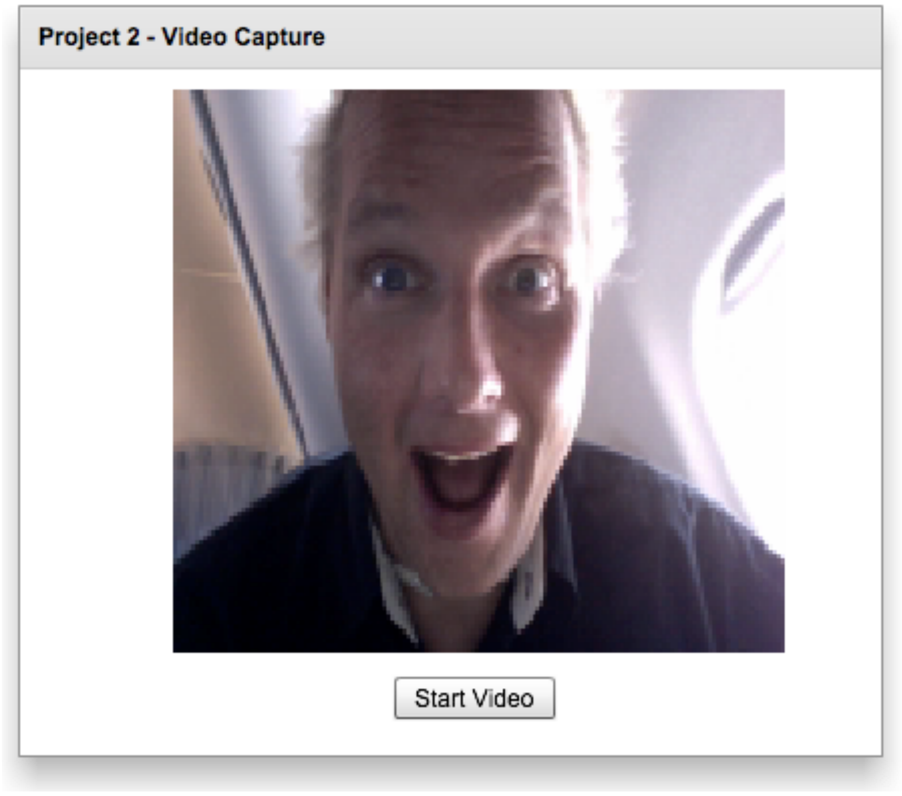
   A:

8. **Q: What is the purpose of the Application Descriptor file within AIR?**

   A:

Project 3: Video Capture

VIDEO – learning how to work with Video. AIR has the same basic video classes as Flash. You can access system resources such as cameras. We will build this file from the ground up.



The main point of this project is to demonstrate how to work with capturing video using the flash.media.camera class.

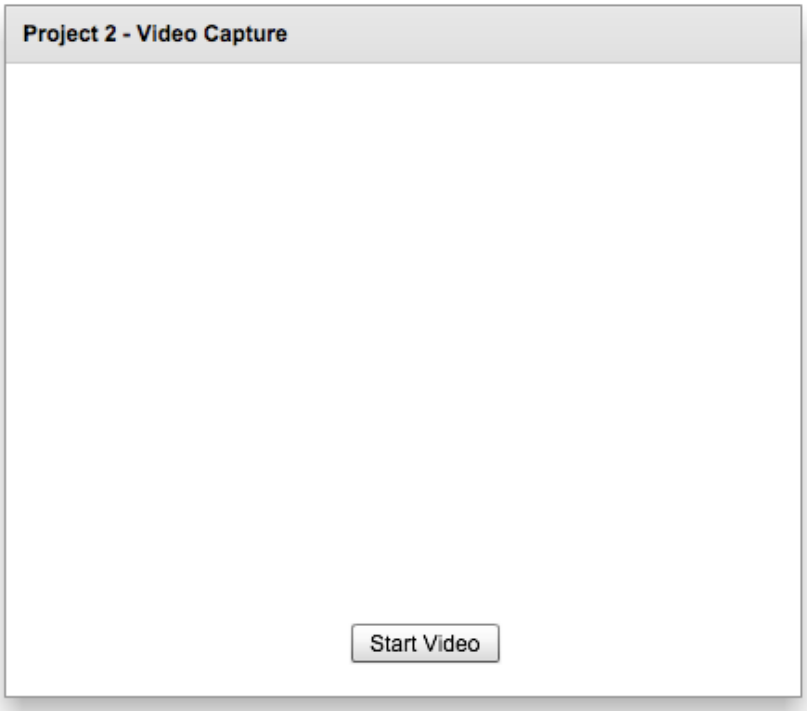
Secondary Goals: Layouts. This project makes use of layout managers including how to use "vertical" vs "absolute", using the anchors, using % vs absolute values etc.

This will be built from scratch so no attendee project exists.

NOTE: you need a camera for this to work!

Instructions:

1. Start a new project named "ABC3-VideoCapture" and switch to design mode.
2. Add a Panel, and a Button laid out as shown below.



3. Switch to code view and add an <mx:VideoDisplay> element inside the <Panel> and name the Video Display “myVideo”.

```
<s:Panel width="430" height="374" x="226" y="9"
         title="Project 2 - Video Capture">
    <mx:VideoDisplay id="myVid" width="304" height="280"  y="10" x="76"/>
    <s:Button x="186" y="302" label="Start Video"/>
</s:Panel>
```

4. Add an <fx:Script> script block and explicitly import the flash.media.Camera class:

```
       <fx:Script>
           <![CDATA[
           import flash.media.Camera;
```

5. Immediately below the import statement, create a new Camera variable and add the following function

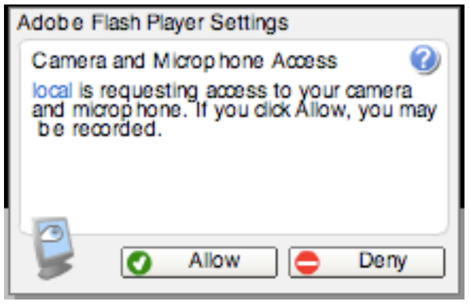
```
        public var myCamera:Camera;

        public function startCam():void
        {
            myCamera = Camera.getCamera();
            myVid.attachCamera(myCamera);
            myVid.autoPlay;
        }
```

6. Wire up the button to the function (click = startCam())
7. Run the program.

Project 3: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                  xmlns:s="library://ns.adobe.com/flex/spark"
                  xmlns:mx="library://ns.adobe.com/flex/halo"
                 minWidth="1024" minHeight="768">
      <fx:Script>
          <![CDATA[
          import flash.media.Camera;

          private var myCamera:Camera;


          private function StartCam():void
          {
                    myCamera = Camera.getCamera();
                    myVid.attachCamera(myCamera);
                    myVid.autoPlay;
             }
          ]]>
     </fx:Script>
     <s:Panel width="430" height="374" x="226" y="9"
               title="Project 2 - Video Capture">
      <mx:VideoDisplay id="myVid" width="304" height="280"  y="10" x="76"/>
      <s:Button  click="StartCam()" x="186" y="302" label="Start Video"/>
      </s:Panel>
</s:Application>
```

When you run this program, it will ask you for explicit permissions to use the camera on your system. You may have to configure the camera and select the correct camera. Click “allow”. On most systems, this will be the USB camera.



local is requesting access to your camera and microphone. If you click Allow, you may be recorded.

Project 3: test

1. **Q: What class must be used to work with a system camera?**

   A:

2. **Q: What is the first parameter the "attachCamera" can accept? Is it optional or mandatory?**

   A:

3. **Q: What is the difference between the VideoDisplay.autoplay() and play() methods?**

   A:

4. **Q: In the previous code example, why was the camera.getCamera method not called before the call to the function?**

   A:

5. **Q: Given the following code:**

```
<s:Panel width="430" height="374" x="226" y="9"
         title="Project 2 - Video Capture">
    <mx:VideoDisplay id="myVid" width="304" height="280"  y="10" x="76"/>
    <s:Button x="186" y="302" label="Start Video"/>
</s:Panel>
```

What will the X and Y positioning of the VideoDisplay and the Button use as base coordinates?

A:

Project 4: Video and Full Screen (BFS)

AIRBootCamp4: An extension of the video project to simply demonstrate the H.264 codecs. This project will make use of one asset – the HDTV video. Attendees will write it from scratch. This project will also demonstrate one new feature of AIR – **full screen mode.** The full screen mode combined with HDTV looks pretty slick.

Instructions:

1. Set up a new project called ABC4-FullScreenVideo
2. You will need to drag and drop a movie file to play in this project. There is one located in the folder ~/CompletedProjects/ABC4-FullScreenVideo/Note the movie is in HD but the audio might not play (codecs).
3. Add a function below to make this project run in full screen mode. The code sets the stage.displayState to FULL SCREEN INTERATIVE.

```
<mx:Script>
<![CDATA[
  private function init():void
  {
  stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
  }
]]>
</mx:Script>
```

4. Add code to call the newly created function on the ApplicationComplete event handler.
5. Run the code.

Project 4: Solution Code

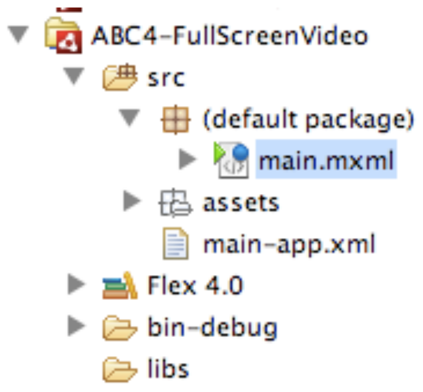
```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                          xmlns:s="library://ns.adobe.com/flex/spark"
                          xmlns:mx="library://ns.adobe.com/flex/halo"
                           applicationComplete="init()">
             <fx:Script>
             <![CDATA[
                    private function init():void
                    {
                     //nativeWindow.displayState =
StageDisplayState.FULL_SCREEN_INTERACTIVE;
             stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
                    }
             ]]>
      </fx:Script>

      <mx:VideoDisplay width="100%" height="100%"
             source="assets/ThermoPublic.mov"  autoPlay="true"/>

</s:WindowedApplication>
```

Project 4: Test Questions

1. **Q: Describe the difference between the applicationComplete event and the creationComplete events. When are each fired?**

2. **Q: What will the VideoDisplay use as the basis for calculating it's width and height in the previous code example?**

3. **Q: In the project shown below, briefly describe what each of the visible project components represent or is for?**



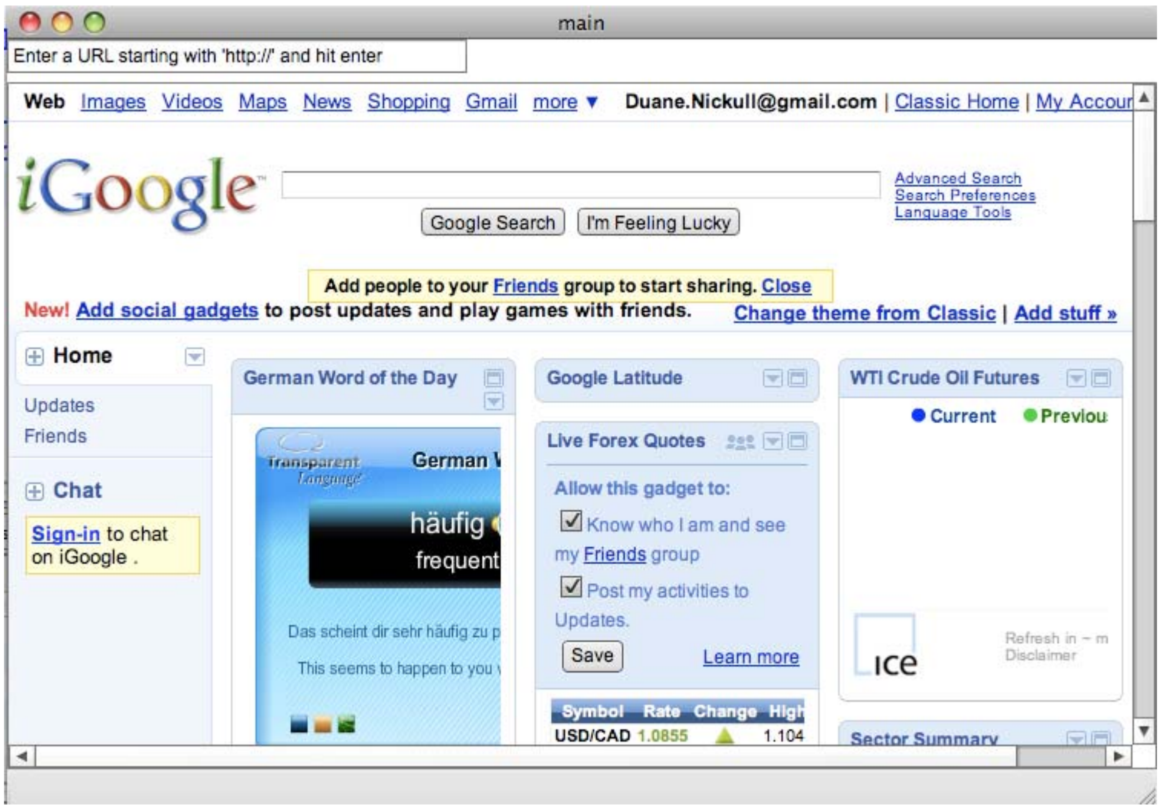
4. **Q: What does the green arrow beside the main.mxml above indicate?**

5. **Q: What is the difference between the states of FULL_SCREEN and FULL_SCREEN_INTERACTIVE?**

6. **Q: Will FULL_SCREEN_INTERACTIVE work in a Flash Player based application vs. AIR only?**

Project 5: HTML

Demonstrates the WebKit engine including a discussion on how much is available from Webkit in AIR. Attendees will write this project from scratch and learn how to set the URL, how it handles international characters, CSS and AJAX.


main
Enter a URL starting with 'http://' and hit enter
Web Images Videos Maps News Shopping Gmail more ▾ Duane.Nickull@gmail.com | Classic Home | My Accour
iGoogle
Google Search
I'm Feeling Lucky
Advanced Search
Search Preferences
Language Tools
Add people to your Friends group to start sharing. Close
New! Add social gadgets to post updates and play games with friends.
Change theme from Classic | Add stuff »
Home
Updates
Friends
Chat
Sign-in to chat on iGoogle .
German Word of the Day
Google Latitude
WTI Crude Oil Futures
Live Forex Quotes
Allow this gadget to:
Know who I am and see my Friends group
Post my activities to Updates.
Save
Learn more
Current
Previou
Sector Summary

Instructions:

Build this from Scratch (new -> Project -> AIR…). Note that the type must be of AIR as the HTML component is not available within a Browser based application.

1. Start the new project. Name it ABC5-WorkingWithHTML

2. Add width and height properties of 750 and 500 respective for the initial window. This is done in the root element using the following declarations.

```
 ...width="750" height="500">
```

3. Change to source code view and add the following code to change the Spark layout to vertical.

```
        <s:layout>
                <s:VerticalLayout/>
        </s:layout>
```

4. Add both a textInput and HTML component and set their properties as

shown below. Be sure to do this in order as components appear in a vertical layout in the order in which they are declared, unlike absolute layouts.

```
<s:TextInput width="300" text="Enter a URL starting with 'http://' and hit
enter" />

<mx:HTML width="100%" height="100%" location="http://www.google.com/ig" />
```

5. Now add identifiers for each component.

```
<s:TextInput id="myTI" width="300" text="Enter a URL starting with 'http://'
and hit enter" />

<mx:HTML  id="myHTML" width="100%" height="100%"
location="http://www.google.com/ig" />
```

6. Finally, add a line of code to catch the "enter" event from the TextInput and use it to set the location property of the HTML component to the text value of the TextInput component. This is accomplished with the following line of code:

```
<s:TextInput enter="myHTML.location=myTI.text" …
```

7. Run project

Project 5: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                         xmlns:s="library://ns.adobe.com/flex/spark"
                         xmlns:mx="library://ns.adobe.com/flex/halo"
                         width="750" height="500">
       <s:layout>
              <s:VerticalLayout/>
       </s:layout>

       <s:TextInput enter="myHTML.location=myTI.text" id="myTI" width="300"
text="Enter a URL starting with 'http://' and hit enter" />
```

```
       <mx:HTML  id="myHTML" width="100%" height="100%"
location="http://www.google.com/ig" />
</s:WindowedApplication>
```

Project 5: Test Questions

Q: What would be wrong with the following syntax in the project above?

```
enter="myTI.text=myHTML.location";
```

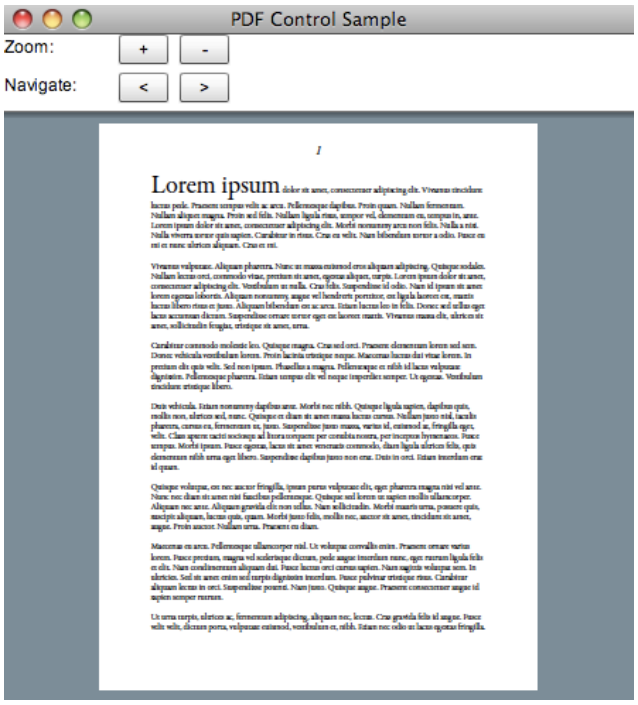
Q: Can you use the HTML component within a Bwoser based Flex Application? How can you tell?

Q: Why must the following line use single quotation marks?

```
text="Enter a URL starting with 'http://' and hit enter"
```

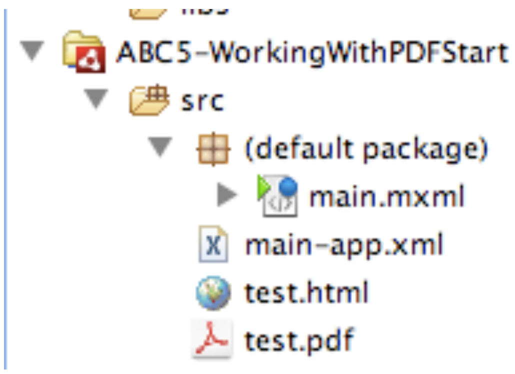
Project 6: Working with PDF

AIR has worked with PDF since version 1.0. This project shows a very simple way to control PDF in a rudimentary manner.



1. Import the project ABC5-WorkingWithPDFStart from the ~/AttendeeProjects folder.

2. The project has three main components – an main.mxml application, an HTML file and a PDF document all under the ~/src folder.



3. The HTML file is only tasked with loading up the PDF as an object. The file is very simplistic.

```
<html>
        <body>
```

```
                <object id="PDFObj"
                                data="test.pdf"
                                type="application/pdf"
                                width="100%"
                                height="100%">
                </object>
        </body>
</html>
```

4. Open the main.mxml file. The code has been almost completed but we will walk briefly through the logic behind the code. The first function call to init() is made when the application completes.

5. The next line of code adds an event listener for detecting when the HTML DOM is initialzed.

```
private function init():void
{
    pdfHtml.htmlLoader.addEventListener(Event.HTML_DOM_INITIALIZE, htmlLoaded);

}
```

6. If you look down to the event handler referenced in init for the HTML_DOM_INITIALIZE event, you will see that it basically enables the buttons.

7. The task before you is to talk to the PDF object. You will find the comments where to enter that code around line number 32.

8. We are going to use a new construct we have not yet shown in this course called a try-catch loop. This catches errors thrown when a declared block of logic fails to execute properly. The try-catch construct is the same as in Java. Add the following construct to your code at line 44.

```
try
        {
```

```
      // something here to try
      }
      catch (error:Error)
      {
      // do something with the error
      }
```

9. Now add in the lines of code to get the PDF object by reference and catch the error and trace it out.

```
try
      {
      var pdfObj:Object =
          pdfHtml.htmlLoader.window.document.getElementById("PDFObj");
      pdfObj.postMessage([message]);
      }
      catch (error:Error)
      {
      trace( "Error: " + error.name + "\nError message: "
                        + error.message);
      }
```

10. Now run your code.

Project 6: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                        xmlns:s="library://ns.adobe.com/flex/spark"
                        xmlns:mx="library://ns.adobe.com/flex/halo"
                        title="PDF Control Sample"
                        initialize="init()">
<s:layout>
   <s:VerticalLayout />
</s:layout>
<fx:Script><![CDATA[

  private function init():void
  {
  pdfHtml.htmlLoader.addEventListener(Event.HTML_DOM_INITIALIZE, htmlLoaded);
  }

  private function htmlLoaded(event:Event):void
  {
       btn1.enabled = true;
       btn2.enabled = true;
       btn3.enabled = true;
       btn4.enabled = true;
  }
  /**
   * Sends a message to the JavaScript in the PDF page.
   */
  private function sendMessage(message:String):void
  {
  try
     {
        var pdfObj:Object =
            pdfHtml.htmlLoader.window.document.getElementById("PDFObj");
        pdfObj.postMessage([message]);
     }
     catch (error:Error)
     {
       trace( "Error: " + error.name + "\nError message: " +
                error.message);
     }
   }
]]></fx:Script>

<mx:HBox>
  <mx:Label text="Zoom:" width="75"/>
  <mx:Button id="btn1" label="+" click="sendMessage('ZoomIn')" width="35"
             enabled="false"/>
  <mx:Button id="btn2" label="-" click="sendMessage('ZoomOut')" width="35"
             enabled="false"/>
</mx:HBox>
<mx:HBox>
  <mx:Label text="Navigate:" width="75"/>
  <mx:Button id="btn3" label="&lt;" click="sendMessage('PageUp')" width="35"
             enabled="false"/>
  <mx:Button id="btn4" label=">" click="sendMessage('PageDn')" width="35"
             enabled="false"/>
</mx:HBox>
<mx:HTML id="pdfHtml" location="test.html" width="100%" height="100%" />
</s:WindowedApplication>
```

Project 6: Test Questions

Q: In the hat event is handed to the `htmlLoaded()` function and where will the event trigger be fired?

Q: When will the event be fired?

Q: Does the `HTML_DOM_INITIALIZE` event mean that the HTML and PDF has finished rendering?

Q: How will the buttons be laid out in the following code?

```
<mx:HBox>
  <mx:Label text="Zoom:" width="75"/>
  <mx:Button id="btn1" label="+" click="sendMessage('ZoomIn')" width="35"
             enabled="false"/>
  <mx:Button id="btn2" label="-" click="sendMessage('ZoomOut')" width="35"
             enabled="false"/>
</mx:HBox>
```

Q: What logic must be present in a try-catch construct and what is the syntax?

Q: How is the Flex application communicating with the specific PDF document?

Project 7: Reading and Writing to local Disk

In this example when the user clicks the save button the application pops up the "Save As" dialog which lets the user select which file they would like to save the contents of the RichTextEditor to. When the user has selected a file the "saveData" method is called. The "saveData" method creates a new FileStream object which allows for read and write operations on a file. Next the file which the user selected is opened in write mode, the bytes from the htmlText property of the RichTextEditor are written to the file, and finally the file is closed.

Similar to the "Save As" dialog which is opened by calling the "browseForSave" function on a File object there are also functions for opening files and directories. Those functions are "browseForDirectory", "browseForOpen" which can take a filter parameter so that only specific file types can be selected, and "browseForOpenMultiple" which can open multiple files at once.

There are also a number of out-of-the-box Flex components for browsing the file system such as FileSystemComboBox, FileSystemDataGrid, FileSystemList, and FileSystemTree.

Instructions:

Build From Scratch:

1. Start a new Flex Project and choose type of AIR

2. Add a width and height attribute of 750 and 550 respectively to the root WindowedApplication element.

```
...width="750" height="550">
```

3. Add an <fx:script> block and a new function called saveData(). The function needs to take an Event and returns void.

```
<fx:Script><![CDATA[
  private function saveData(event:Event):void
  {
  }
  ]]></fx:Script>
```

4. Complete the saveData() function by adding four lines of code to create a new FileStream object, open the stream to write, write and close the stream.

```
     var stream:FileStream = new FileStream();
    stream.open((event.target as File),FileMode.WRITE);
    stream.writeUTFBytes(rte.htmlText);
    stream.close();
```

5. Add a RichTextEditor to your application with the following properties:

```
<mx:RichTextEditor id="rte" width="100%" height="480" title="Text Editor"/>
```

6. Below rte, add a Button with a label of Save Document, then add an inline click handler. Within the click handler, create a new File object, invoke the browseForSave() method and pass the SELECT event off to the saveData() function.

```
<s:Button label="Save Document" textAlign="center" x="327" y="503">
    <s:click>
    var f:File = File.desktopDirectory;
    f.browseForSave("Save As");
    f.addEventListener(Event.SELECT, saveData);
    </s:click>
  </s:Button>
```

7. Run your application. Try saving a file to your desktop then opening it up with a browser to see if it worked. Make sure to save the file at <someFileName>.html

Project 7: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                          xmlns:s="library://ns.adobe.com/flex/spark"
                           xmlns:mx="library://ns.adobe.com/flex/halo"
width="750" height="550">

       <fx:Script>
              <![CDATA[
                     private function saveData(event:Event):void
                     {
```

```
                              var stream:FileStream = new FileStream();
                              stream.open((event.target as File),FileMode.WRITE);
                              stream.writeUTFBytes(rte.htmlText);
                              stream.close();
                        }
                  ]]>
            </fx:Script>

      <mx:RichTextEditor id="rte" width="100%" height="480" title="Text Editor"/>
      <s:Button label="Save Document" textAlign="center" x="327" y="503">
        <s:click>
        var f:File = File.desktopDirectory;
        f.browseForSave("Save As");
        f.addEventListener(Event.SELECT, saveData);
        </s:click>
      </s:Button>
</s:WindowedApplication>
```

Project 7: Test Questions

Q: What are the advantages and disadvantages of using handling the click event for the button using the MXML syntax inline?

Q: What piece of syntax might be missing from the above code sample within the <s:click> fragment of MXML?

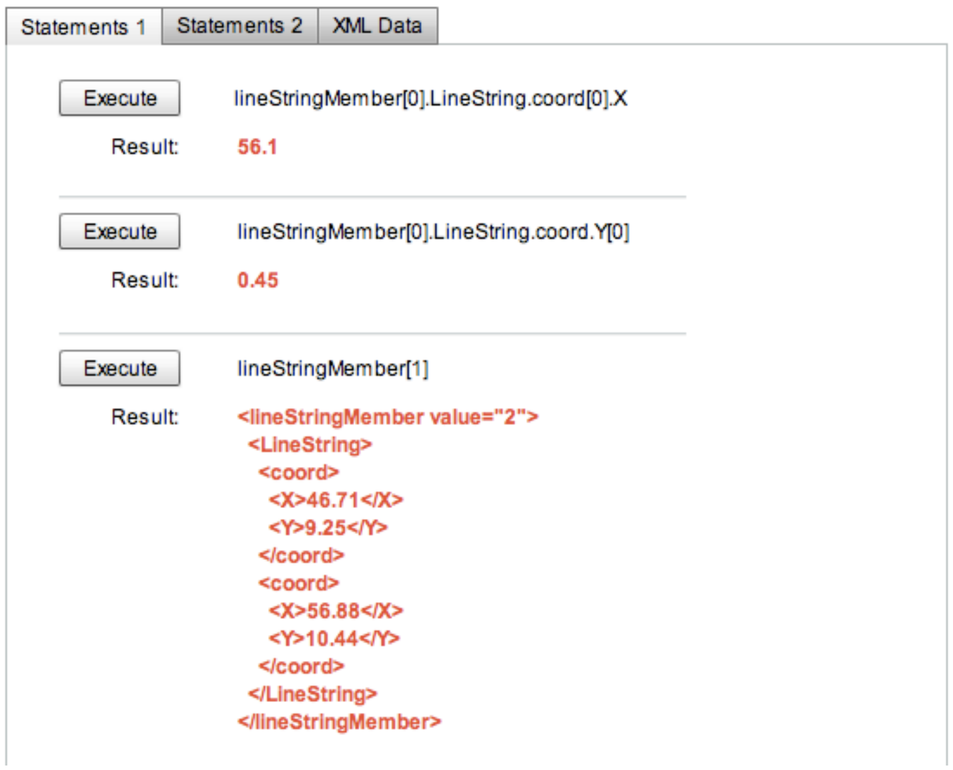
Q: What is the `File.desktopDirectory` and what issues may present themselves on unix based systems (AIR runs on Mac OSX and Linux)?

Q: What does the browseForSave() method do and when is the SELECT event dispatched?

Project 8: Working with XML

This project will teach students how to work with XML, continuing from the syntax of the previous project. This will be presented to them and they will have to code up the solutions.

Discussion Point: use this project to show how to create the illusion of fast execution by using the visible attribute on components.



```
lineStringMember[0].LineString.coord[0].X
Result: 56.1

lineStringMember[0].LineString.coord.Y[0]
Result: 0.45

lineStringMember[1]
Result: <lineStringMember value="2">
         <LineString>
          <coord>
           <X>46.71</X>
           <Y>9.25</Y>
          </coord>
          <coord>
           <X>56.88</X>
           <Y>10.44</Y>
          </coord>
         </LineString>
        </lineStringMember>
```

Problem:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                  xmlns:s="library://ns.adobe.com/flex/spark"
                  xmlns:mx="library://ns.adobe.com/flex/halo"
                  minWidth="1024" minHeight="768">
```

```
<!--Alternative loading of XML from remote call
<mx:HTTPService id="myGMLData"
                            url="http://www.nickull.net/blogimages/GML.mxml"
                             resultFormat="e4x">
</mx:HTTPService>
-->
<fx:Script>
      <![CDATA[
[Bindable]
public var myGMLData:XML =
<MultiLineString srsName="http://www.opengis.net/gml/srs/epsg.xml">
  <lineStringMember value="1">
    <LineString>
      <coord>
        <X>56.1</X>
        <Y>0.45</Y>
      </coord>
      <coord>
        <X>67.23</X>
        <Y>0.98</Y>
      </coord>
    </LineString>
  </lineStringMember>

  <lineStringMember value="2">
    <LineString>
      <coord><X>46.71</X><Y>9.25</Y></coord>
      <coord><X>56.88</X><Y>10.44</Y></coord>
    </LineString>
  </lineStringMember>

  <lineStringMember value="3">
    <LineString>
      <coord><X>324.1</X><Y>219.7</Y></coord>
      <coord><X>0.45</X><Y>4.56</Y></coord>
    </LineString>
  </lineStringMember>
</MultiLineString>
]]>
</fx:Script>


    <mx:TabNavigator x="12" y="10" width="544" height="565">
        <mx:Canvas label="Statements 1" width="100%" height="632">

<!--TODO: Access 67.23 from the XML above and bind it to a label-->
    <mx:Label id="XCoordinate" visible="false" text="{ }" x="131" y="40"
width="305" fontWeight="bold" color="#e82d1b"/>
    <mx:Button x="30" y="10" label="Execute" click="XCoordinate.visible=true"/>
    <mx:Label x="129" y="12" text="lineStringMember[0].LineString.coord[0].X"
width="327.5"/>
    <mx:Text x="30" y="40" text="Result:" width="72" textAlign="right"/>
    <mx:HRule x="30" y="77" width="362"/>
```

```
<!--TODO: Access 0.45 from the XML above and bind it to a label-->
    <mx:Label id="YCoordinate"  visible="false" text="{}" x="131" y="117"
width="307" fontWeight="bold" color="#e82d1b"/>
    <mx:Button x="30" y="87" label="Execute" click="YCoordinate.visible=true"/>
    <mx:Label x="131" y="89" text="lineStringMember[0].LineString.coord.Y[0]"
width="327.5"/>
    <mx:Text x="30" y="117" text="Result:" width="72" textAlign="right"/>
    <mx:HRule x="30" y="156" width="362"/>

<!--TODO: Return the entire XML Fragment of the second instance of LineStringMember-->
    <mx:Label id="FragmentReturned" visible="false" text="{}" height="263"
width="233" x="131" y="196" fontWeight="bold" color="#e8351c"/>
    <mx:Button x="30" y="166" label="Execute"
click="FragmentReturned.visible=true"/>
    <mx:Label x="131" y="168" text="lineStringMember[1]" width="168"/>
    <mx:Text x="30" y="196" text="Result:" width="72" textAlign="right"/>
</mx:Canvas>

<mx:Canvas label="Statements 2" width="100%" height="100%">

<!--TODO: use parenthesis operator to grab the instance of X where is equals 324.1 and
return the coord fragment-->
    <mx:Label id="Parenthesis"  visible="false" text="{}" height="68"
width="233" x="111" y="50" fontWeight="bold" color="#e8371c"/>
    <mx:Button x="10" y="20" label="Execute" click="Parenthesis.visible=true"/>
    <mx:Text x="10" y="50" text="Result:" width="72" textAlign="right"/>
    <mx:Label x="111" y="22" text="lineStringMember.LineString.coord.(X ==
324.1)"/>
    <mx:HRule x="10" y="126" width="362"/>

<!--TODO: Return any fragment for coord where Y has a specific value of "10.44" -->
<mx:Label id="combination" visible="false" text="{}" height="80" width="326"
x="111" y="164" fontWeight="bold" color="#e9441c"/>
    <mx:Button x="10" y="138" label="Execute"
click="combination.visible=true"/>
    <mx:Text x="10" y="168" text="Result:" width="72" textAlign="right"/>
    <mx:Label x="111" y="140" text="lineStringMember..coord.(Y == 10.44)"/>
    <mx:HRule x="10" y="243" width="362"/>

    <!–TODO: DESCENDENT operator (..) return all instances of Y that are
descendents of lineStringMember-->
    <mx:Label id="Descendent" visible="false" text="{}" height="87" width="235"
x="111" y="281" fontWeight="bold" color="#e9471b"/>
    <mx:Button x="10" y="253" label="Execute" click="Descendent.visible=true"/>
    <mx:Text x="10" y="283" text="Result:" width="72" textAlign="right"/>
    <mx:Label x="111" y="255" text="lineStringMember..Y (all instances of Y)"
width="247"/>
    <mx:HRule x="10" y="369" width="362"/>

<!--TODO: Return the lineStringMember XML fragment where the attribute value == 2 -->
<mx:Label id="Attribute"  visible="false" text="{}" x="111" y="405" width="261"
height="105" fontWeight="bold" color="#e9431b"/>
    <mx:Text x="10" y="405" text="Result:" width="72" textAlign="right"/>
    <mx:Button x="10" y="375" label="Execute" click="Attribute.visible=true"/>
    <mx:Label x="111" y="377" text="lineStringMember.(@value == 2)"
width="272"/>        </mx:Canvas>

        <mx:Canvas label="XML Data" width="100%" height="100%">
        <!-- YEAH - this is really lame but I was sitting on a plane and wanted
to make the XML look like it was formatted nicely.  If anyone wants to redo,
please be my guest. I hate entity references....  -->

            <mx:Text x="10" y="10" text="&lt;? xml version=&quot;1.0&quot;
encoding=&quot;UTF-8&quot; ?&gt;"/>
            <mx:Text x="26" y="23" text="&lt;MultiLineString
srsName=&quot;http://www.opengis.net/gml/srs/epsg.xml&quot;&gt;"/>
            <mx:Text x="44" y="36" text="&lt;lineStringMember
value=&quot;1&quot;&gt;"/>
            <mx:Text x="60" y="49" text="&lt;lineStringMember
value=&quot;1&quot;&gt;"/>
            <mx:Text x="60" y="166" text="&lt;lineStringMember
value=&quot;2&quot;&gt;"/>
            <mx:Text x="60" y="283" text="&lt;/lineStringMember&gt;"/>
            <mx:Text x="60" y="296" text="&lt;lineStringMember
value=&quot;3&quot;&gt;"/>
            <mx:Text x="60" y="416" text="&lt;/lineStringMember&gt;"/>
            <mx:Text x="26" y="426" text="&lt;/MultiLineString&gt;"/>
            <mx:Text x="77" y="62" text="&lt;LineString&gt;"/>
            <mx:Text x="73" y="309" text="&lt;LineString&gt;"/>
            <mx:Text x="73" y="179" text="&lt;LineString&gt;"/>
            <mx:Text x="68" y="153" text="&lt;/LineString&gt;"/>
            <mx:Text x="73" y="270" text="&lt;/LineString&gt;"/>
            <mx:Text x="73" y="400" text="&lt;/LineString&gt;"/>
            <mx:Text x="98" y="75" text="&lt;coord&gt;"/>
            <mx:Text x="93" y="101" text="&lt;/coord&gt;"/>
            <mx:Text x="93" y="114" text="&lt;coord&gt;"/>
            <mx:Text x="93" y="140" text="&lt;/coord&gt;"/>
            <mx:Text x="93" y="192" text="&lt;coord&gt;"/>
            <mx:Text x="93" y="218" text="&lt;/coord&gt;"/>
            <mx:Text x="93" y="231" text="&lt;coord&gt;"/>
            <mx:Text x="93" y="257" text="&lt;/coord&gt;"/>
            <mx:Text x="93" y="322" text="&lt;coord&gt;"/>
            <mx:Text x="88" y="390" text="&lt;/coord&gt;"/>
            <mx:Text x="88" y="364" text="&lt;coord&gt;"/>
            <mx:Text x="88" y="348" text="&lt;/coord&gt;"/>
            <mx:Text x="107" y="88" text="&lt;X&gt;56.1&lt;/X&gt;
&lt;Y&gt;0.45&lt;/Y&gt;"/>
            <mx:Text x="107" y="127" text="&lt;X&gt;62.73&lt;/X&gt;
&lt;Y&gt;0.98&lt;/Y&gt;"/>
            <mx:Text x="107" y="205" text="&lt;X&gt;46.71&lt;/X&gt;
&lt;Y&gt;9.25&lt;/Y&gt;"/>
            <mx:Text x="107" y="244" text="&lt;X&gt;56.88&lt;/X&gt;
&lt;Y&gt;10.44&lt;/Y&gt;"/>
            <mx:Text x="107" y="335" text="&lt;X&gt;324.1&lt;/X&gt;
&lt;Y&gt;219.7&lt;/Y&gt;"/>
            <mx:Text x="107" y="374" text="&lt;X&gt;0.45&lt;/X&gt;
&lt;Y&gt;4.56&lt;/Y&gt;"/>
```

```
        </mx:Canvas>
    </mx:TabNavigator>
</s:Application>
```

Project 8: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                  xmlns:s="library://ns.adobe.com/flex/spark"
                  xmlns:mx="library://ns.adobe.com/flex/halo"
                  minWidth="1024" minHeight="768">

<!--Alternative loading of XML from remote call
<mx:HTTPService id="myGMLData"
                           url="http://www.nickull.net/blogimages/GML.mxml"
                            resultFormat="e4x">
</mx:HTTPService>
-->
<fx:Script>
       <![CDATA[
[Bindable]
public var myGMLData:XML =
<MultiLineString srsName="http://www.opengis.net/gml/srs/epsg.xml">
  <lineStringMember value="1">
    <LineString>
      <coord>
        <X>56.1</X>
        <Y>0.45</Y>
      </coord>
      <coord>
        <X>67.23</X>
        <Y>0.98</Y>
      </coord>
    </LineString>
  </lineStringMember>

  <lineStringMember value="2">
    <LineString>
      <coord><X>46.71</X><Y>9.25</Y></coord>
      <coord><X>56.88</X><Y>10.44</Y></coord>
    </LineString>
  </lineStringMember>

  <lineStringMember value="3">
    <LineString>
      <coord><X>324.1</X><Y>219.7</Y></coord>
      <coord><X>0.45</X><Y>4.56</Y></coord>
    </LineString>
  </lineStringMember>
</MultiLineString>
]]>
</fx:Script>

    <mx:TabNavigator x="12" y="10" width="544" height="565">
        <mx:Canvas label="Statements 1" width="100%" height="632">

    <mx:Label id="XCoordinate" visible="false"
text="{myGMLData.lineStringMember[0].LineString.coord[0].X}" x="131" y="40"
width="305" fontWeight="bold" color="#e82d1b"/>
    <mx:Button x="30" y="10" label="Execute" click="XCoordinate.visible=true"/>
    <mx:Label x="129" y="12" text="lineStringMember[0].LineString.coord[0].X"
width="327.5"/>
    <mx:Text x="30" y="40" text="Result:" width="72" textAlign="right"/>
    <mx:HRule x="30" y="77" width="362"/>

    <mx:Label id="YCoordinate"  visible="false"
text="{myGMLData.lineStringMember[0].LineString.coord.Y[0]}" x="131" y="117"
width="307" fontWeight="bold" color="#e82d1b"/>
    <mx:Button x="30" y="87" label="Execute" click="YCoordinate.visible=true"/>
    <mx:Label x="131" y="89" text="lineStringMember[0].LineString.coord.Y[0]"
width="327.5"/>
    <mx:Text x="30" y="117" text="Result:" width="72" textAlign="right"/>
    <mx:HRule x="30" y="156" width="362"/>

    <!--DOT::FRAGMENT: returns an XML Fragment-->
    <mx:Label id="FragmentReturned" visible="false"
text="{myGMLData.lineStringMember[1]}" height="263" width="233" x="131" y="196"
fontWeight="bold" color="#e8351c"/>
    <mx:Button x="30" y="166" label="Execute"
click="FragmentReturned.visible=true"/>
    <mx:Label x="131" y="168" text="lineStringMember[1]" width="168"/>
    <mx:Text x="30" y="196" text="Result:" width="72" textAlign="right"/>
</mx:Canvas>


<mx:Canvas label="Statements 2" width="100%" height="100%">

    <!--PARENTHESIS: Filters the XML fragment based on X having an exact value
of 324.1-->
    <mx:Label id="Parenthesis"  visible="false"
text="{myGMLData.lineStringMember.LineString.coord.(X == 324.1)}" height="68"
width="233" x="111" y="50" fontWeight="bold" color="#e8371c"/>
    <mx:Button x="10" y="20" label="Execute" click="Parenthesis.visible=true"/>
    <mx:Text x="10" y="50" text="Result:" width="72" textAlign="right"/>
    <mx:Label x="111" y="22" text="lineStringMember.LineString.coord.(X ==
324.1)"/>
    <mx:HRule x="10" y="126" width="362"/>

    <!--COMBINATION example.  Note use of math operators which cast values to
integers-->
    <mx:Label id="combination" visible="false"
text="{myGMLData.lineStringMember..coord.(Y == 10.44)}" height="80" width="326"
x="111" y="164" fontWeight="bold" color="#e9441c"/>
```

```
    <mx:Button x="10" y="138" label="Execute"
click="combination.visible=true"/>
    <mx:Text x="10" y="168" text="Result:" width="72" textAlign="right"/>
    <mx:Label x="111" y="140" text="lineStringMember..coord.(Y == 10.44)"/>
    <mx:HRule x="10" y="243" width="362"/>

    <!--DESCENDENT:  returns all instances of Y that are descendents of
lineStringMember-->
    <mx:Label id="Descendent" visible="false"
text="{myGMLData.lineStringMember..Y}" height="87" width="235" x="111" y="281"
fontWeight="bold" color="#e9471b"/>
    <mx:Button x="10" y="253" label="Execute" click="Descendent.visible=true"/>
    <mx:Text x="10" y="283" text="Result:" width="72" textAlign="right"/>
    <mx:Label x="111" y="255" text="lineStringMember..Y (all instances of Y)"
width="247"/>
    <mx:HRule x="10" y="369" width="362"/>

    <!--ATTRIBUTE: returns the attribute-->
    <mx:Label id="Attribute"  visible="false"
text="{myGMLData.lineStringMember.(@value = 2)}" x="111" y="405" width="261"
height="105" fontWeight="bold" color="#e9431b"/>
    <mx:Text x="10" y="405" text="Result:" width="72" textAlign="right"/>
    <mx:Button x="10" y="375" label="Execute" click="Attribute.visible=true"/>
    <mx:Label x="111" y="377" text="lineStringMember.(@value == 2)"
width="272"/>
        </mx:Canvas>

        <mx:Canvas label="XML Data" width="100%" height="100%">
        <!-- YEAH - this is really lame but I was sitting on a plane and wanted
to make the XML
             look like it was formatted nicely.  If anyone wants to redo,
please be my guest.
             I hate entity references....  -->

            <mx:Text x="10" y="10" text="&lt;? xml version=&quot;1.0&quot;
encoding=&quot;UTF-8&quot; ?&gt;"/>
            <mx:Text x="26" y="23" text="&lt;MultiLineString
srsName=&quot;http://www.opengis.net/gml/srs/epsg.xml&quot;&gt;"/>
            <mx:Text x="44" y="36" text="&lt;lineStringMember
value=&quot;1&quot;&gt;"/>
            <mx:Text x="60" y="49" text="&lt;lineStringMember
value=&quot;1&quot;&gt;"/>
            <mx:Text x="60" y="166" text="&lt;lineStringMember
value=&quot;2&quot;&gt;"/>
            <mx:Text x="60" y="283" text="&lt;/lineStringMember&gt;"/>
            <mx:Text x="60" y="296" text="&lt;lineStringMember
value=&quot;3&quot;&gt;"/>
            <mx:Text x="60" y="416" text="&lt;/lineStringMember&gt;"/>
            <mx:Text x="26" y="426" text="&lt;/MultiLineString&gt;"/>
            <mx:Text x="77" y="62" text="&lt;LineString&gt;"/>
            <mx:Text x="73" y="309" text="&lt;LineString&gt;"/>
            <mx:Text x="73" y="179" text="&lt;LineString&gt;"/>
            <mx:Text x="68" y="153" text="&lt;/LineString&gt;"/>
            <mx:Text x="73" y="270" text="&lt;/LineString&gt;"/>
            <mx:Text x="73" y="400" text="&lt;/LineString&gt;"/>
            <mx:Text x="98" y="75" text="&lt;coord&gt;"/>
            <mx:Text x="93" y="101" text="&lt;/coord&gt;"/>
            <mx:Text x="93" y="114" text="&lt;coord&gt;"/>
            <mx:Text x="93" y="140" text="&lt;/coord&gt;"/>
            <mx:Text x="93" y="192" text="&lt;coord&gt;"/>
            <mx:Text x="93" y="218" text="&lt;/coord&gt;"/>
            <mx:Text x="93" y="231" text="&lt;coord&gt;"/>
            <mx:Text x="93" y="257" text="&lt;/coord&gt;"/>
            <mx:Text x="93" y="322" text="&lt;coord&gt;"/>
            <mx:Text x="88" y="390" text="&lt;/coord&gt;"/>
            <mx:Text x="88" y="364" text="&lt;coord&gt;"/>
            <mx:Text x="88" y="348" text="&lt;/coord&gt;"/>
            <mx:Text x="107" y="88" text="&lt;X&gt;56.1&lt;/X&gt;
&lt;Y&gt;0.45&lt;/Y&gt;"/>
            <mx:Text x="107" y="127" text="&lt;X&gt;62.73&lt;/X&gt;
&lt;Y&gt;0.98&lt;/Y&gt;"/>
            <mx:Text x="107" y="205" text="&lt;X&gt;46.71&lt;/X&gt;
&lt;Y&gt;9.25&lt;/Y&gt;"/>
            <mx:Text x="107" y="244" text="&lt;X&gt;56.88&lt;/X&gt;
&lt;Y&gt;10.44&lt;/Y&gt;"/>
            <mx:Text x="107" y="335" text="&lt;X&gt;324.1&lt;/X&gt;
&lt;Y&gt;219.7&lt;/Y&gt;"/>
            <mx:Text x="107" y="374" text="&lt;X&gt;0.45&lt;/X&gt;
&lt;Y&gt;4.56&lt;/Y&gt;"/>
        </mx:Canvas>
    </mx:TabNavigator>
</s:Application>
```

Project 8: Test Questions

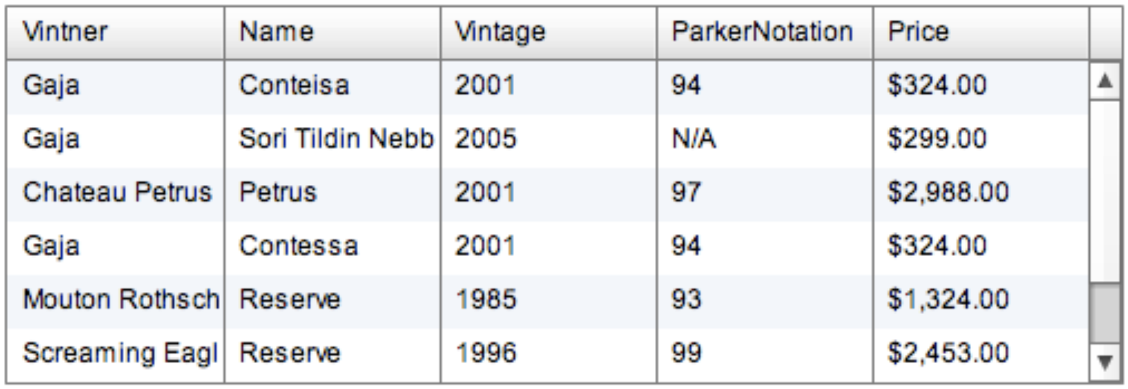
Q: What unique performance can the TabNavigator bring to your application upon initialization?

Q: What will the following E4X statement return?

```
RootElement.Foo..Bar.(bah == 33.23)
```

Q: What disadvantages might using E4X bring into your program vs a binary format such as AMF?

Project 9: REST



Vintner	Name	Vintage	ParkerNotation	Price
Gaja	Conteisa	2001	94	$324.00
Gaja	Sori Tildin Nebb	2005	N/A	$299.00
Chateau Petrus	Petrus	2001	97	$2,988.00
Gaja	Contessa	2001	94	$324.00
Mouton Rothsch	Reserve	1985	93	$1,324.00
Screaming Eagl	Reserve	1996	99	$2,453.00

Representational State Transfer refers to a collection of network architecture principles that outline how resources are defined and addressed (as opposed to HTTP, which is strictly limited to one abstract interaction pattern with minor variations).

A REST interface describes any service that transmits domain-specific data over HTTP without an additional messaging layer such as SOAP, or session tracking via HTTP cookies.

REST is worth reading about in further detail as it contains many subtle nuances. Claims have been made (as noted at http://en.wikipedia.org/wiki/Representational_State_Transfer)
that it is possible to:

1. Design a service in accordance with the REST architectural style without using HTTP, and

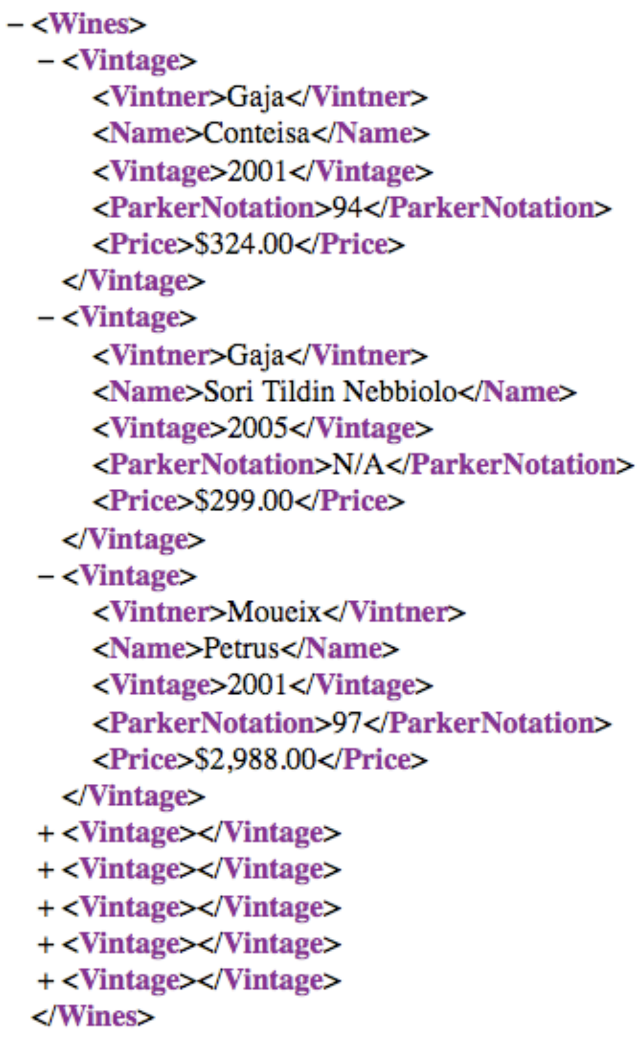
2. That it is possible to design simple XML + HTTP interfaces that do not conform to REST principles http://en.wikipedia.org/wiki/Representational_State_Transfer

Instructions:

Build from scratch

1. Grab a web browser and navigate to the following URL:
   http://www.nickull.net/xml/Wines.xml
   Or optionally (BlazeDS must be started and running) to:
   http://localhost:8400/xml/Wines.xml

2. You should see the following XML:



```
- <Wines>
  - <Vintage>
      <Vintner>Gaja</Vintner>
      <Name>Conteisa</Name>
      <Vintage>2001</Vintage>
      <ParkerNotation>94</ParkerNotation>
      <Price>$324.00</Price>
    </Vintage>
  - <Vintage>
      <Vintner>Gaja</Vintner>
      <Name>Sori Tildin Nebbiolo</Name>
      <Vintage>2005</Vintage>
      <ParkerNotation>N/A</ParkerNotation>
      <Price>$299.00</Price>
    </Vintage>
  - <Vintage>
      <Vintner>Moueix</Vintner>
      <Name>Petrus</Name>
      <Vintage>2001</Vintage>
      <ParkerNotation>97</ParkerNotation>
      <Price>$2,988.00</Price>
    </Vintage>
  + <Vintage></Vintage>
  + <Vintage></Vintage>
  + <Vintage></Vintage>
  + <Vintage></Vintage>
  + <Vintage></Vintage>
  </Wines>
```

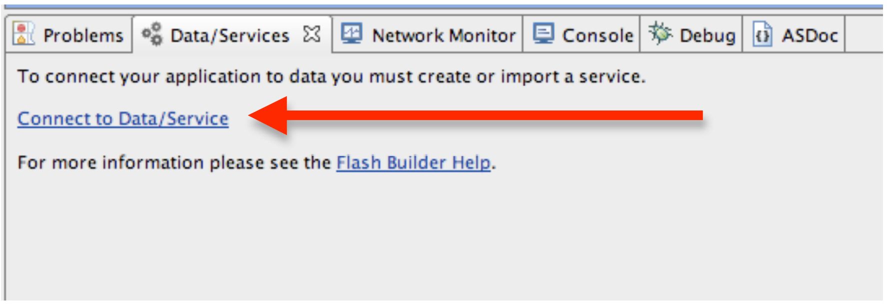
In order for this to work, a small file called crossdomain.xml is in the same directory as the Wines.xml on the server. This file looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <allow-access-from domain="*" />
  <site-control permitted-cross-domain-policies="master-only"/>
</cross-domain-policy>
```

Describing what this file does is beyond the scope of this tutorial however more can be read at http://livedocs.adobe.com/flash/8/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00001621.html

3. With Flash Builder 4, start a new Flex 4 Project called "ABC8-REST". Make it a web-browser based project (Flex) and select no server type. Click "Finish".

4. At the bottom of the Flash Builder IDE< look for and click on the hyperlink that says "Connect to Data Service" as shown below.

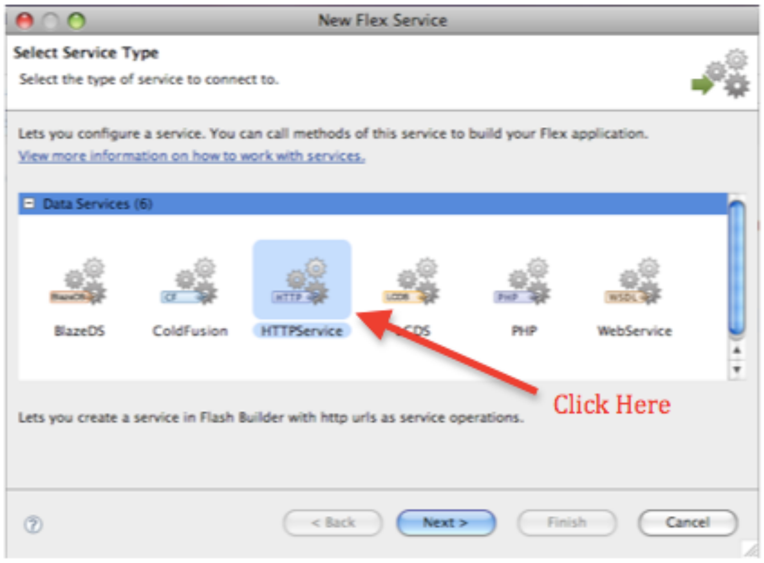


To connect your application to data you must create or import a service.

Connect to Data/Service

For more information please see the Flash Builder Help.

5. This will bring up the Dialog window with a number of choices for services. Select the HTTPService option as shown below:



Select Service Type

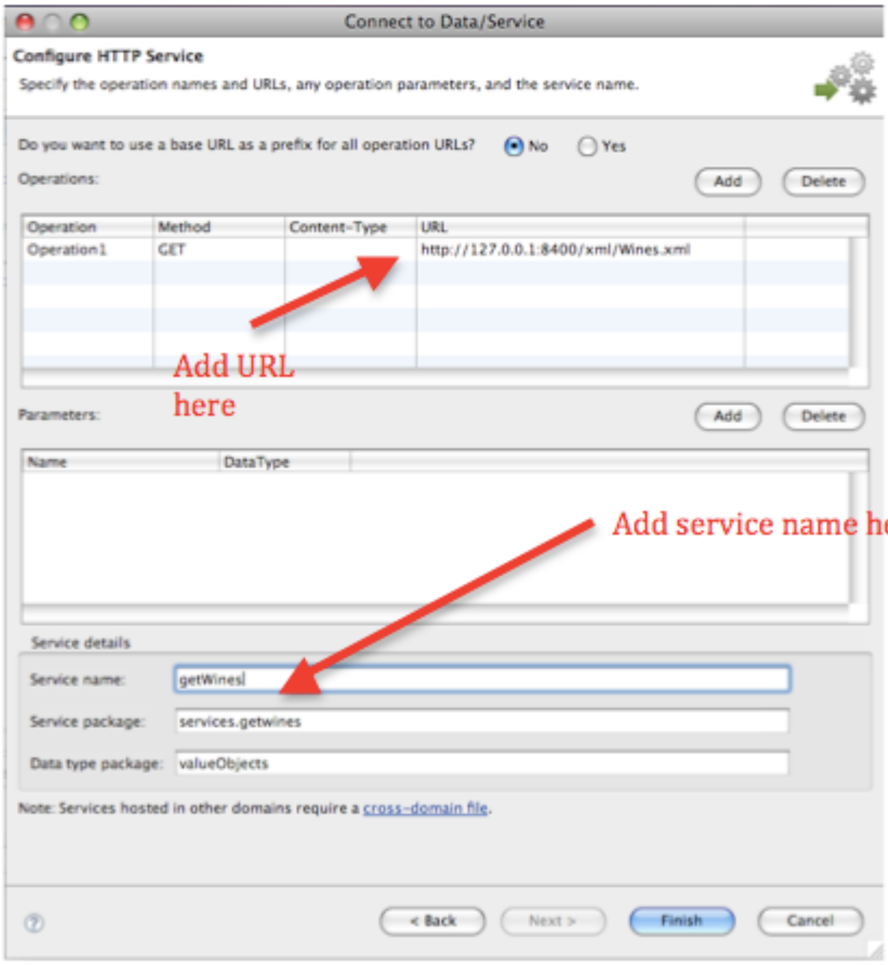
Select the type of service to connect to.

Lets you configure a service. You can call methods of this service to build your Flex application.

View more information on how to work with services.

Lets you create a service in Flash Builder with http urls as service operations.

6. Click “Next” and then fill in the next screen by adding the URL of the XML to the URL column near the top and add the name of the service (“getWines” in this case) as shown below:



Configure HTTP Service

Specify the operation names and URLs, any operation parameters, and the service name.

Do you want to use a base URL as a prefix for all operation URLs? [✓ No] Yes

Operations:

Operation	Method	Content-Type	URL
Operation1	GET		http://127.0.0.1:8400/xml/Wines.xml

Parameters:

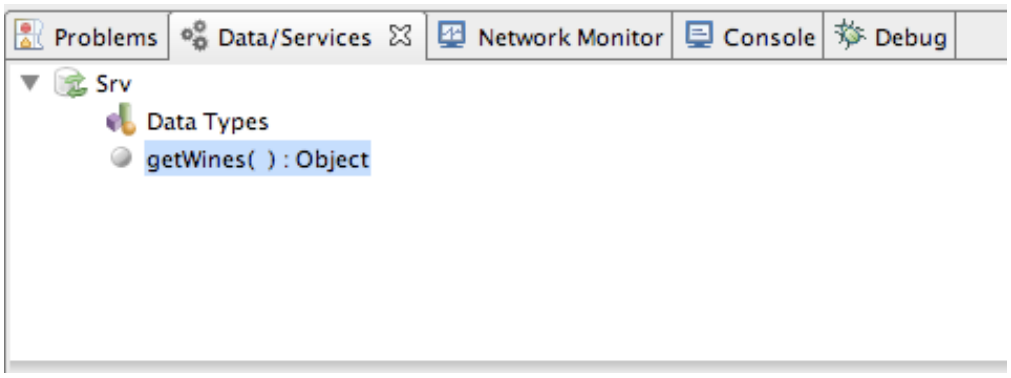
Name	DataType

Service details

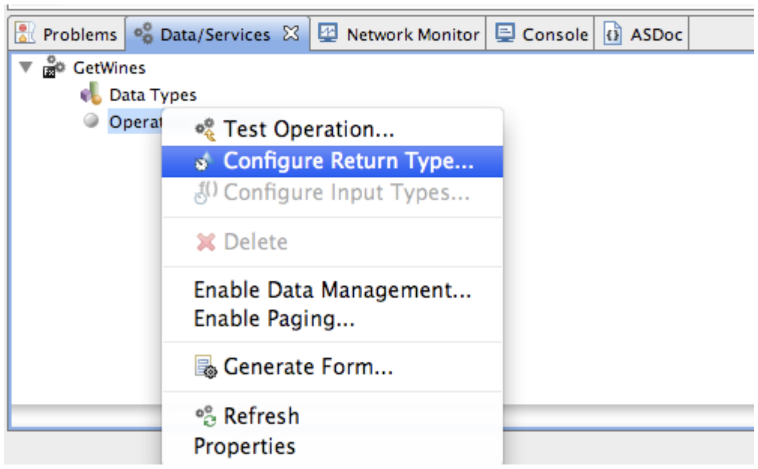
Service name:	getWines
Service package:	services.getwines
Data type package:	valueObjects

Note: Services hosted in other domains require a cross-domain file.

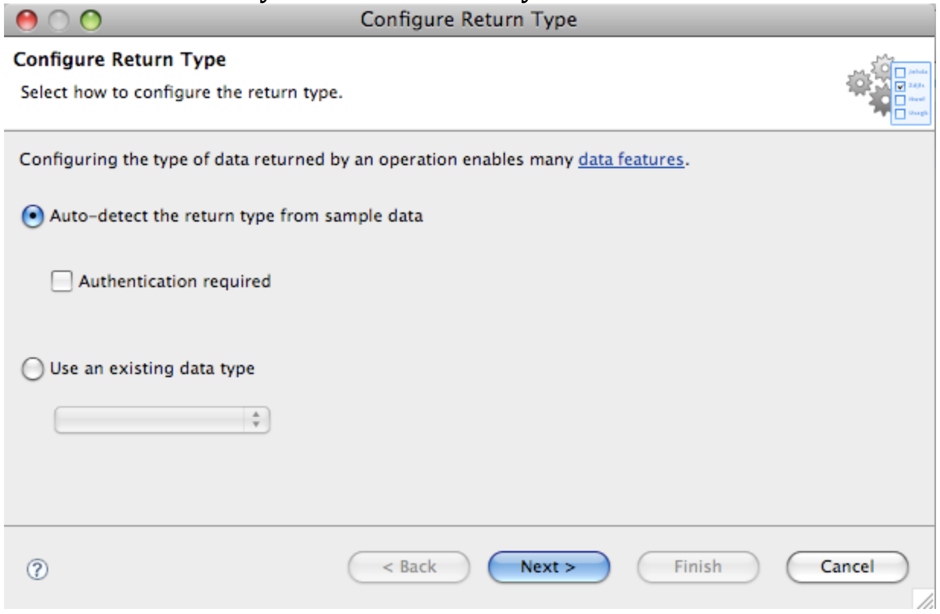
7. After completing this operation by clicking “Finish”, you will see the data services in the panel at the bottom.



8. Now the return type will have to be configured. On the bottom of the IDE you will see a number of tabs. One is called "Data Services" and under that tab you should see your operation. Right click (PC) or Command Click (mac) on the Operation and select "Configure Return Type" as shown below.



9. The dialog box that opens will ask you how you want to configure the return data type. By default, the "Auto-detect..." should be selected. Leave this selected as this is by far the easiest way to use Flash Builder 4.


Configure Return Type

Configure Return Type

Select how to configure the return type.

Configuring the type of data returned by an operation enables many data features.

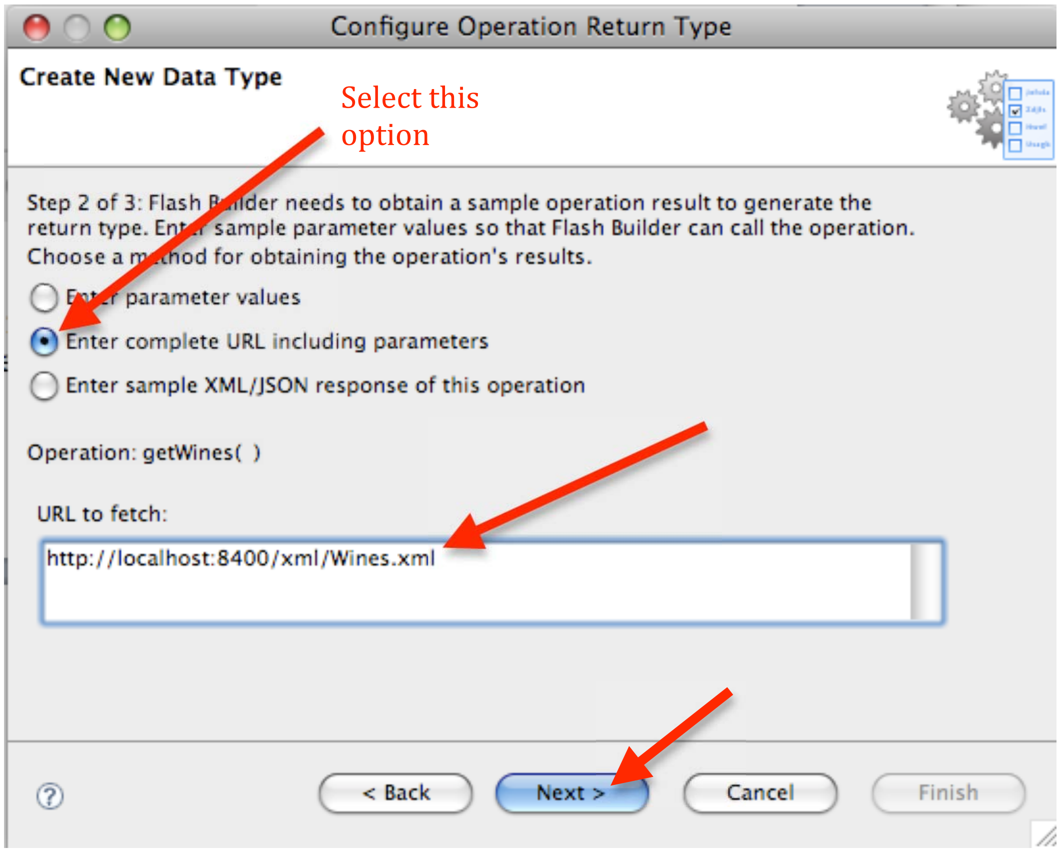
[✓ Auto-detect the return type from sample data]

☐ Authentication required

Use an existing data type

10. Hit the "Next" button. The next dialog will invoke the service request to get a sample of the data to inspect. For this lab exercise, select the radio button entitled "Enter complete URL ..." and enter the URL of the XML file in the

"URL to fetch" box below and hit "Next".


Configure Operation Return Type

Create New Data Type

Step 2 of 3: Flash Builder needs to obtain a sample operation result to generate the return type. Enter sample parameter values so that Flash Builder can call the operation. Choose a method for obtaining the operation's results.

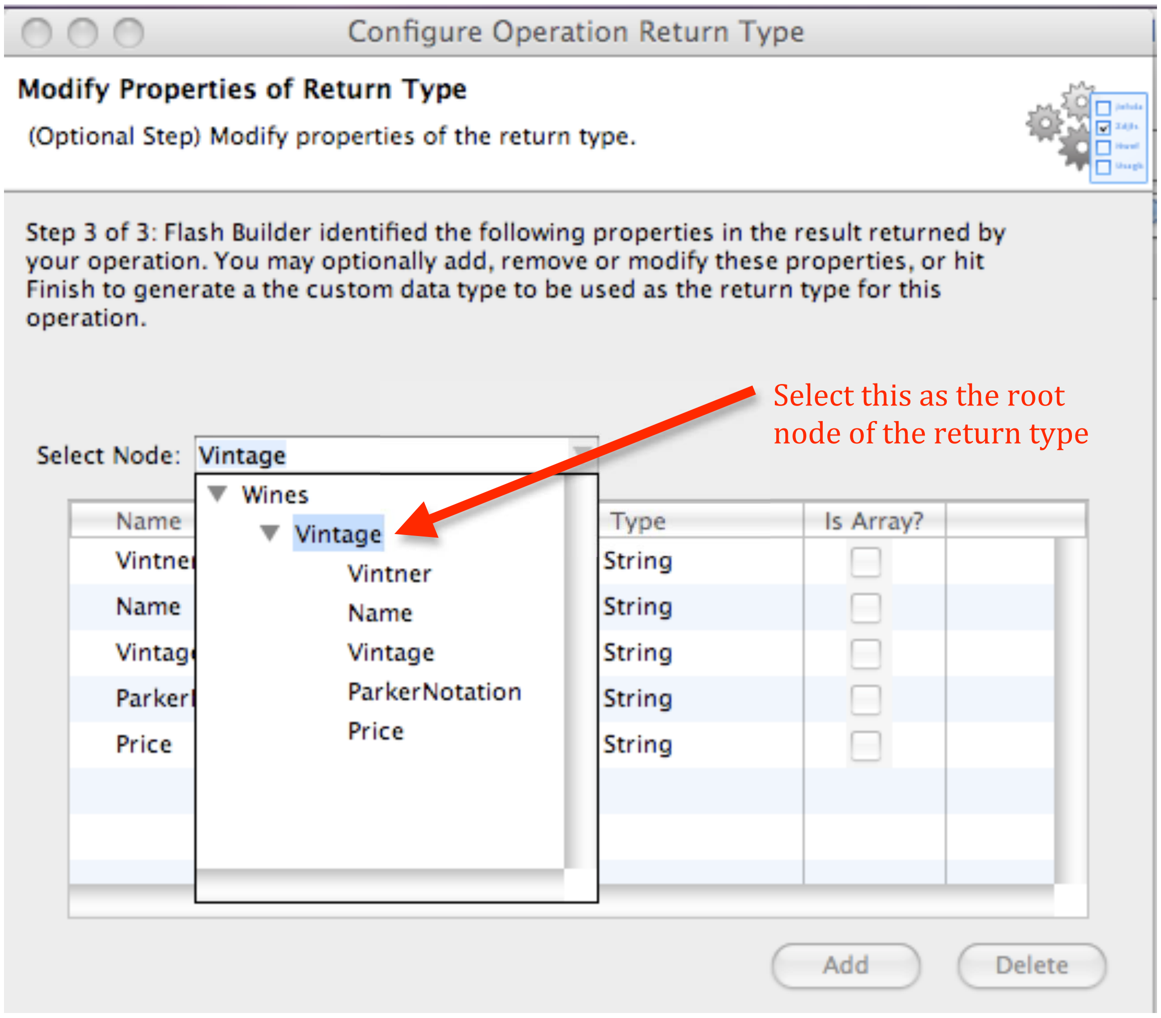
- Enter parameter values
- [✓ Enter complete URL including parameters]
- Enter sample XML/JSON response of this operation

Operation: getWines( )

URL to fetch:

http://localhost:8400/xml/Wines.xml

11. Flash Builder will go to the URL and bring back a sample of data and break down the structure of it for you. In this case, it has selected that it is an array[] of complex object types occurring under the root element. Rather than accept the first option as is, use the drop down list to select a context root of "Vintage" as shown below.



Configure Operation Return Type

Modify Properties of Return Type

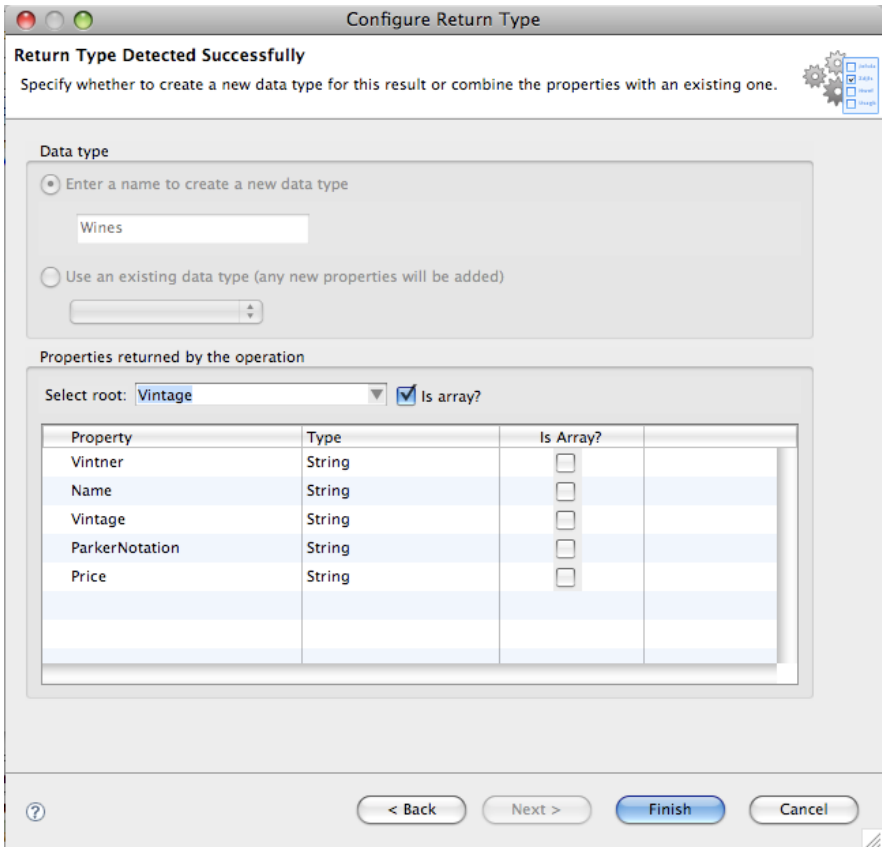
(Optional Step) Modify properties of the return type.

Step 3 of 3: Flash Builder identified the following properties in the result returned by your operation. You may optionally add, remove or modify these properties, or hit Finish to generate a the custom data type to be used as the return type for this operation.

Select Node: Vintage

Name	Type	Is Array?
Vintner	String	☐
Name	String	☐
Vintage	String	☐
ParkerNotation	String	☐
Price	String	☐

12. Now you will have the following:


Configure Return Type

Return Type Detected Successfully

Specify whether to create a new data type for this result or combine the properties with an existing one.

Data type

[✓ Enter a name to create a new data type]

Wines

Use an existing data type (any new properties will be added)

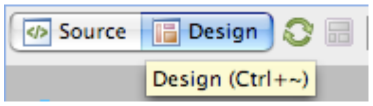
Properties returned by the operation

Select root: Vintage [✓ Is array?]

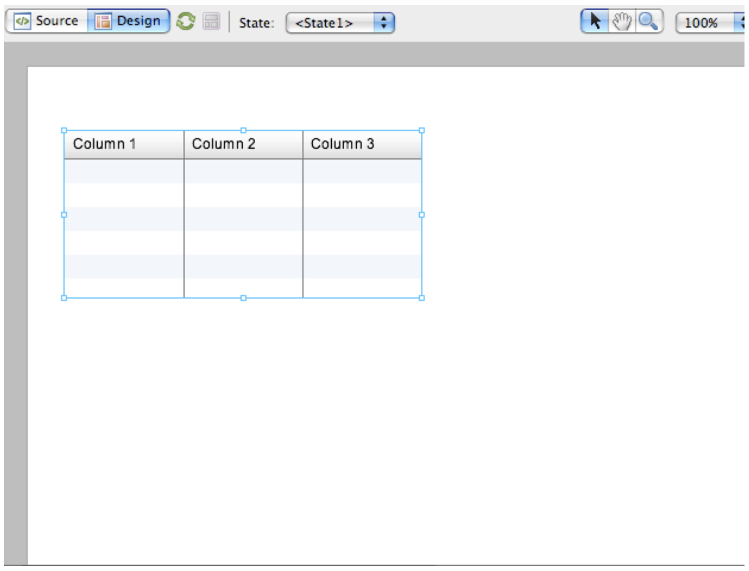
Property	Type	Is Array?
Vintner	String	☐
Name	String	☐
Vintage	String	☐
ParkerNotation	String	☐
Price	String	☐

13. Click "Finish"

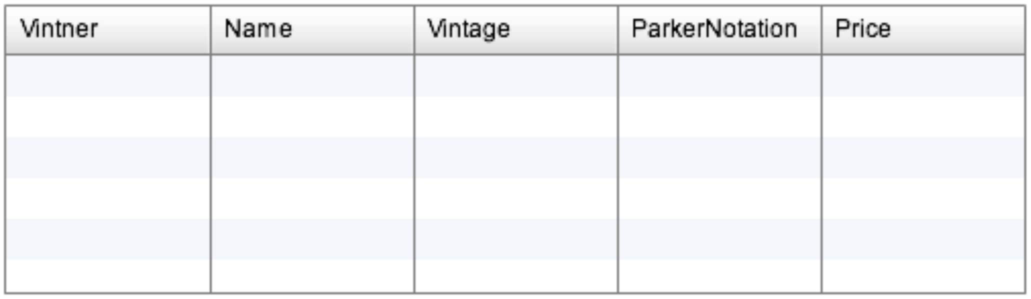
14. Now switch to the "Design" perspective



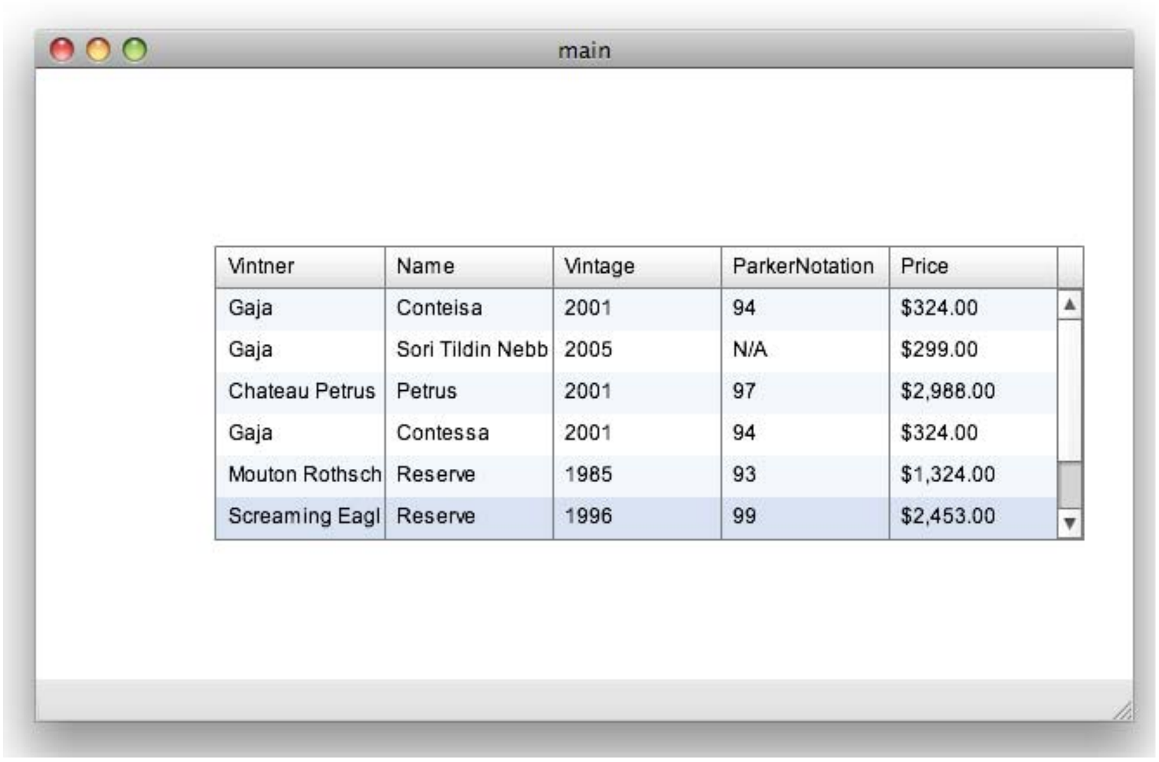
and add a data grid to the upper left hand side of your application as show below.



15. Now drag the operation onto the datagrid and it should re-define itself based on the return data type.   The project should now reflect the DataGrid being bound to the data as shown below:



16. Run the project and you will build and display the XML from the server as shown below.



Vintner	Name	Vintage	ParkerNotation	Price
Gaja	Conteisa	2001	94	$324.00
Gaja	Sori Tildin Nebb	2005	N/A	$299.00
Chateau Petrus	Petrus	2001	97	$2,988.00
Gaja	Contessa	2001	94	$324.00
Mouton Rothsch	Reserve	1985	93	$1,324.00
Screaming Eagl	Reserve	1996	99	$2,453.00

Project 9: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/halo" xmlns:srv="services.srv.*">
      <fx:Script>
      <![CDATA[
      import mx.events.FlexEvent;
      import mx.controls.Alert;

      protected function
dataGrid_creationCompleteHandler(event:FlexEvent):void
      {
            getWinesResult.token = srv.getWines();
      }
      ]]>
      </fx:Script>
      <fx:Declarations>
            <s:CallResponder id="getWinesResult"/>
            <srv:Srv id="srv" fault="Alert.show(event.fault.faultString)"
showBusyCursor="true"/>
      </fx:Declarations>
    <mx:DataGrid x="103" y="102" id="dataGrid"
creationComplete="dataGrid_creationCompleteHandler(event)"
dataProvider="{getWinesResult.lastResult}" editable="true">
      <mx:columns>
      <mx:DataGridColumn headerText="Vintner" dataField="Vintner"/>
      <mx:DataGridColumn headerText="Name" dataField="Name"/>
      <mx:DataGridColumn headerText="Vintage" dataField="Vintage"/>
      <mx:DataGridColumn headerText="ParkerNotation"
                         dataField="ParkerNotation"/>
      <mx:DataGridColumn headerText="Price" dataField="Price"/>
      </mx:columns>
   </mx:DataGrid>

</s:WindowedApplication>
```

Project 9: Text Questions

Q: In the preceding project, what would be returned if we had selected the root return node as <Wines> instead of <Vintages>?

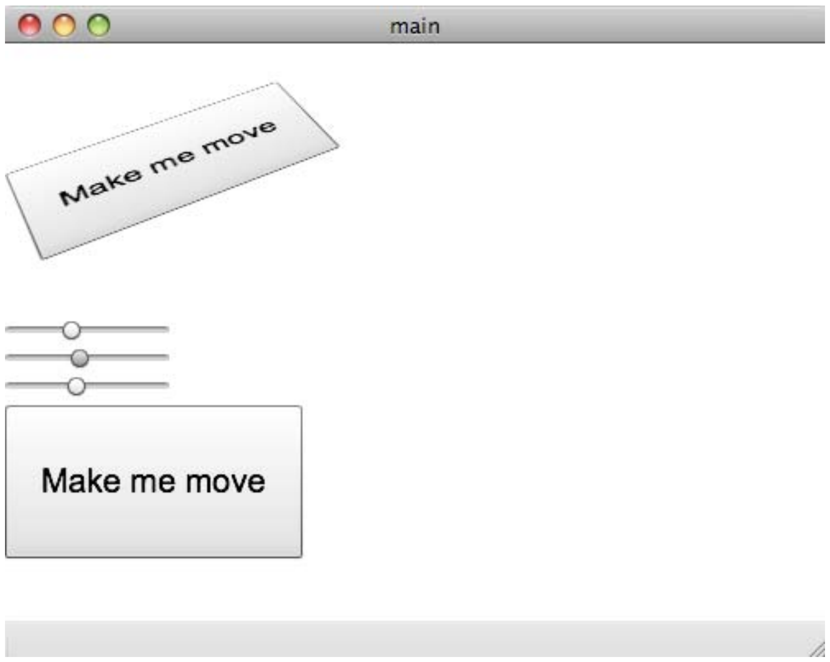
Q: What exactly is "lastResult"?

Q: Where is the generated code stored and how to I access it?

Q: What HTTP method did we use in the previous project and how does a developer know?

Q: How can you modify the source code to control the sending of the service request and link it to a button?

Project 10: AIR 3D



The Flash Player and AIR introduced some cool ways to manipulate graphic components in 3 axis. These give developers the power to do 3D type effects. This is a rudimentary example only and proper 3D requires a full 3D modeling environment.

1. Start a new project (AIR or Flex) and call it ABC10-3D.
2. Add a button and resize it until it is around 300 pixels wide by 100 pixels high. Change the button text to say "Twist Me:

3. Give the button an ID of "guineaPig". Your code should look something similar to this:

```
<s:Button id="guineaPig" x="269" y="74" label="Make me move" height="93"
width="181" fontSize="20"/>
```

4. Add a single HSlider below the button and set the following properties on it:

```
x="287"
y="236"
id="xControl"
change="twistMe()"
minimum="-180"
maximum="+180"
stepSize="1"
liveDragging="true"
```

5. Copy and paste the HSlider two times, pasting each one slightly below the previous.

6. Change the ID attributes of the new ones to yControl and zControl.

7. Add the twistMe() function. It returns void.

8. Within the function, add the following declarations

```
        guineaPig.rotationX = xControl.value;
        guineaPig.rotationY = yControl.value;
        guineaPig.rotationZ = zControl.value;
```

9. Your code is ready to run. It should look like the code below.

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                         xmlns:s="library://ns.adobe.com/flex/spark"
                         xmlns:mx="library://ns.adobe.com/flex/halo">
<fx:Script>
<![CDATA[
      private function twistMe():void
      {
        guineaPig.rotationX = xControl.value;
        guineaPig.rotationY = yControl.value;
```

```
         guineaPig.rotationZ = zControl.value;
       }
]]>
</fx:Script>

<s:Button id="guineaPig" x="269" y="74" label="Make me move" height="93"
width="181" fontSize="20"/>
<s:HSlider x="287" y="236" id="xControl" change="twistMe()" minimum="-180"
maximum="+180" stepSize="1" liveDragging="true"/>
<s:HSlider x="288" y="280" id="yControl" change="twistMe()" minimum="-180"
maximum="+180" stepSize="1" liveDragging="true"/>
<s:HSlider x="289" y="334" id="zControl" change="twistMe()" minimum="-180"
maximum="+180" stepSize="1" liveDragging="true"/>

</s:WindowedApplication>
```

10. Now we are going to show the dynamic layout manager. Change your layout to be a vertical layout by adding the following lines of code beneath the root element:

```
<s:layout>
       <s:VerticalLayout/>
</s:layout>
```

11. Cut and paste your button and add it to the bottom of the code below your last HSlider declaration.
12. Run the project again and move the button. Note how the layout flows dynamically to adjust size.

Project 10: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                          xmlns:s="library://ns.adobe.com/flex/spark"
                          xmlns:mx="library://ns.adobe.com/flex/halo">
<s:layout>
       <s:VerticalLayout/>
</s:layout>
<fx:Script>
<![CDATA[
       private function twistMe():void
```

```
        {
          guineaPig.rotationX = xControl.value;
          guineaPig.rotationY = yControl.value;
          guineaPig.rotationZ = zControl.value;
        }
]]>
</fx:Script>

<s:Button id="guineaPig" x="269" y="74" label="Make me move" height="93"
width="181" fontSize="20"/>
<s:HSlider x="287" y="236" id="xControl" change="twistMe()" minimum="-180"
maximum="+180" stepSize="1" liveDragging="true"/>
<s:HSlider x="288" y="280" id="yControl" change="twistMe()" minimum="-180"
maximum="+180" stepSize="1" liveDragging="true"/>
<s:HSlider x="289" y="334" id="zControl" change="twistMe()" minimum="-180"
maximum="+180" stepSize="1" liveDragging="true"/>
<s:Button id="guineaPig0" x="260" y="367" label="Make me move" height="93"
width="181" fontSize="20"/>

</s:WindowedApplication>
```

Project 10: Test Questions

Q: What is rotationX and what does it enable?

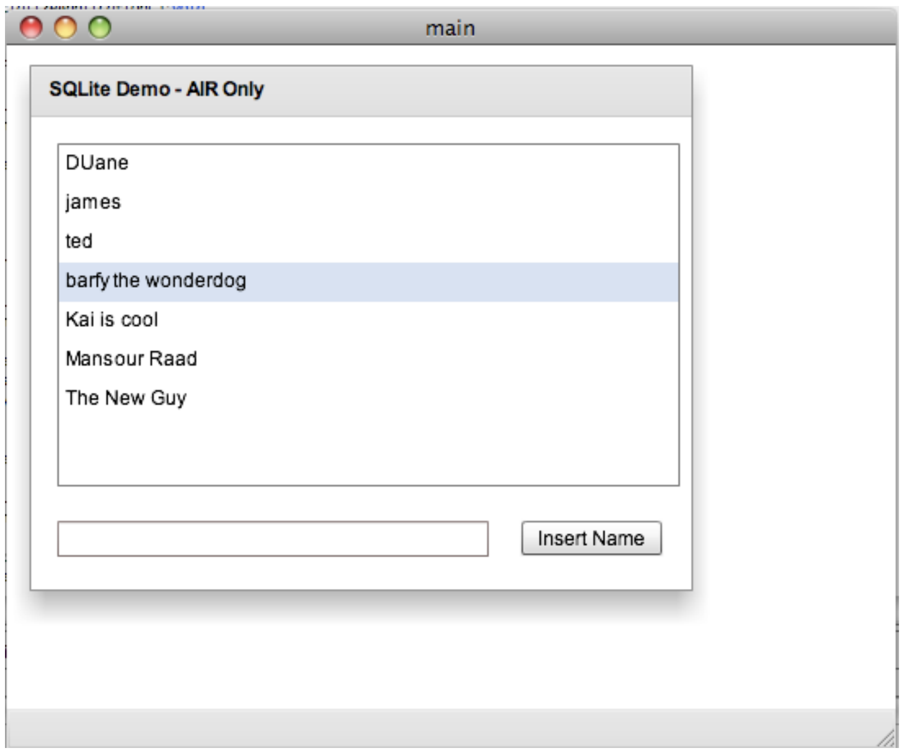
Q: what is “liveDragging=”true”?

Q: When could using an HSlider’s “changing” event result in impaired performance of an application?

Project 11: SQLite

The following project gives attendees a quick overview of how SQLite works in AIR. In this example when the application has fully initialized the applicationComplete event is dispatched which calls the initApp function. In this function a connection to a database is opened. There can be multiple open connections to a single database and those connections can either be synchronous or asynchronous. In the default synchronous method database operations will block the UI while they are being performed. This means that for long running database transactions the UI will be unresponsive. The asynchronous method does not block the UI but requires the developer to specify call-backs for when the database transaction completes. The synchronous method is more straightforward and should probably be used for most interactions. If you do want to use the asynchronous method simply call "sqlConn.openAsync" instead of "sqlConn.open". The database also supports transactions. To begin a transaction just call the "sqlConn.begin" function. To commit, call the "sqlConn.commit" function. And to rollback, call the "sqlConn.rollback" function. If you do have long running database operations it's best to wrap them in a transaction since it will perform faster.

In the example the "queryNames" function does a simple SQL query for the items in the "names" table. Since the connection is synchronous the results are available after the statement has been executed and the results have been fetched. The SQL query syntax is standard SQL-92 and supports all the typical SQL grammar.



The Button in the example has a click event handler which creates a new SQLStatement, specifies the SQL to be an insert statement using parameters which are then replaced automatically by the SQL engine. This is the preferred approach to doing inserts, updates, and deletes since it limits the potential for SQL injection attacks in your application. After the statement has executed the queryNames function is called so that the list of names refreshes.

Many times applications which need to work offline will maintain a cache of the server data on the local machine. These two different datasets then need to be synchronized. This can be a complex and time consuming problem to solve. LiveCycle Data Services now has this functionality built-in and should be considered before attempting to custom build a data synchronization framework.

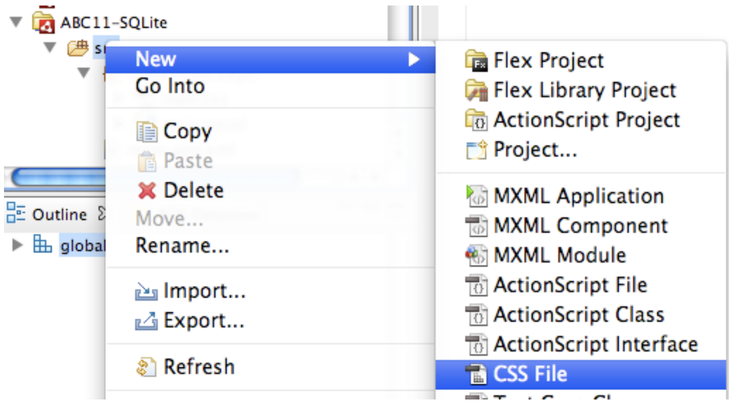
Additionally, this lab will show how to use CSS externally with a project.

The Lab

1. Set up a new project in AIR and call it ABC11-SQLite
2. Add the following visual components:

```
<mx:Panel x="14" y="12" height="322" width="406" layout="absolute"
title="SQLite Demo - AIR Only">
<s:TextInput  id="newname" x="16" y="247" width="264" baseColor="#EB0E0E"/>
<s:Button x="300" y="247" label="Insert Name" click="insertName()"/>
<mx:List dataProvider="{names}" labelField="name"  height="210" width="381"
x="16" y="16"/>
</mx:Panel>
```

3. Right click (PC) or control click (OSX) on the new src folder (in the Package Explorer) and select “new -> CSS” as shown below.



4. IN the new CSS file, change it to read as per the following text:

```
/* CSS file */
@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/halo";


global
{
	symbolColor: #921010;
}
```

5. Save and close the CSS file and add a reference to it back in your MXML file with the following line of code:

```
	<fx:Style source="main.css"/>
```

6. Now add a Script element just below the Style declaration and create the following import declarations and functions:

```
<fx:Script>
	<![CDATA[
		import mx.collections.ArrayCollection;
		import flash.data.*;

		[Bindable]
```

```
            private var names:ArrayCollection;
            private var connection:SQLConnection;

            private function initializeApplication():void
            {

            }

            private function queryName():void
            {
            }

            private function insertName():void
            {
            }
        ]]>
</fx:Script>
```

7. Add a line of code to your root element to call `initializeApplication()` when your application completes initialization.

8. First we will write the initializeApplication() function. This function must do two things. First – it must establish a connection to the database. This is done in the first and second lines. A database named example.db is used for this code and contains the names as we add them.

   The second set of 4 lines prepares a new SQL statement and executes the statement. The final line calls queryName() which then queries the name and updates the names variable. When that names variable is updated, the <mx:List bound to it will display the new state of the variables.

```
private function initializeApplication():void
{
  connection = new SQLConnection();
  connection.open(File.applicationStorageDirectory.resolvePath("example.db"));

 var statement:SQLStatement = new SQLStatement();
  statement.sqlConnection = connection;
  statement.text = "create table if not exists names (name string)";
  statement.execute();
  queryName();
}
```

9. Next we will write the queryName function. This function prepares a statement to query the SQLite database and bind the result to the names variable (ArrayCollection).

```
private function queryName():void
{
	var statement:SQLStatement = new SQLStatement();
	statement.sqlConnection = connection;
	statement.text = "select * from names";
	statement.execute();
	var result:SQLResult = statement.getResult();
	names = new ArrayCollection(result.data);
}
```

10. The final function to write is the insertName() function to write to our database. This is called by the Button when the user wishes to add a new name to the database. It then calls queryName() to update the client side data model after the operation which resets the view of any graphical component bound to the model.

```
private function insertName():void
{
	var statement:SQLStatement = new SQLStatement();
	statement.sqlConnection = connection;
	statement.text = "insert into names (name) values (:name)";
	statement.parameters[":name"] = newname.text;
	statement.execute();
	queryName();
}
```

11. Run your application. Add some names, then quite your application and run it again. The second time you run it, it should start with the names from the previous session already present.

Project 11: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
       xmlns:s="library://ns.adobe.com/flex/spark"
       xmlns:mx="library://ns.adobe.com/flex/halo" height="430" width="544"
       applicationComplete="initializeApplication()">

     <fx:Style source="main.css"/>
```

```
       <fx:Script>
       <![CDATA[
       import mx.collections.ArrayCollection;
       import flash.data.*;

       [Bindable]
       private var names:ArrayCollection;
       private var connection:SQLConnection;

       private function initializeApplication():void
       {
       connection = new SQLConnection();
       connection.open(File.applicationStorageDirectory.resolvePath("example.d
b"));

       var statement:SQLStatement = new SQLStatement();
       statement.sqlConnection = connection;
       statement.text = "create table if not exists names (name string)";
       statement.execute();
       queryName();
       }

       private function queryName():void
       {
       var statement:SQLStatement = new SQLStatement();
       statement.sqlConnection = connection;
       statement.text = "select * from names";
       statement.execute();
       var result:SQLResult = statement.getResult();
       names = new ArrayCollection(result.data);
       }

       private function insertName():void
       {
       var statement:SQLStatement = new SQLStatement();
       statement.sqlConnection = connection;
       statement.text = "insert into names (name) values (:name)";
       statement.parameters[":name"] = newname.text;
       statement.execute();
       queryName();
       }
       ]]>
</fx:Script>
<mx:Panel x="14" y="12" height="322" width="406" layout="absolute"
          title="SQLite Demo - AIR Only">
s:TextInput  id="newname" x="16" y="247" width="264" baseColor="#EB0E0E"/>
<s:Button x="300" y="247" label="Insert Name" click="insertName()"/>
<mx:List dataProvider="{names}" labelField="name"  height="210" width="381"
         x="16" y="16"/>
</mx:Panel>
</s:WindowedApplication>
```

Project 11: Test Questions

Q: Are SQL statements case sensitive (eg. Is “steve” == “Steve”)?

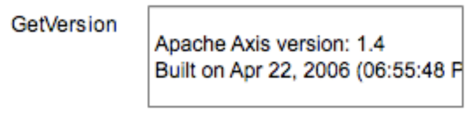
Q: How could you add a delete statement to the project to remove names?

Q: Given the code statement below, what will the Flash Builder IDE look for?

```
      <fx:Style source="main.css"/>
```

Q: Why might this application be poorly designed from an architectural security perspective?

Project 12: Web Service Introspection and Consumption


GetVersion

```
Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 P
```

Background Information

Flex applications can interact with web services that define their interfaces in a Web Services Description Language 1.1 (WSDL 1.1) document, which is available as a

URL. WSDL is a standard format for describing the messages that a web service understands, the format of its responses to those messages, the protocols that the web service supports, and where to send messages. The Flex web service API generally supports Simple Object Access Protocol (SOAP) 1.1, XML Schema 1.0 (versions 1999, 2000, and 2001), and WSDL 1.1 RPC-encoded, RPC-literal, and document-literal (bare and wrapped style parameters). The two most common types of web services use remote procedure call (RPC) encoded or document-literal SOAP bindings; the terms encoded and literal indicate the type of WSDL-to-SOAP mapping that a service uses.

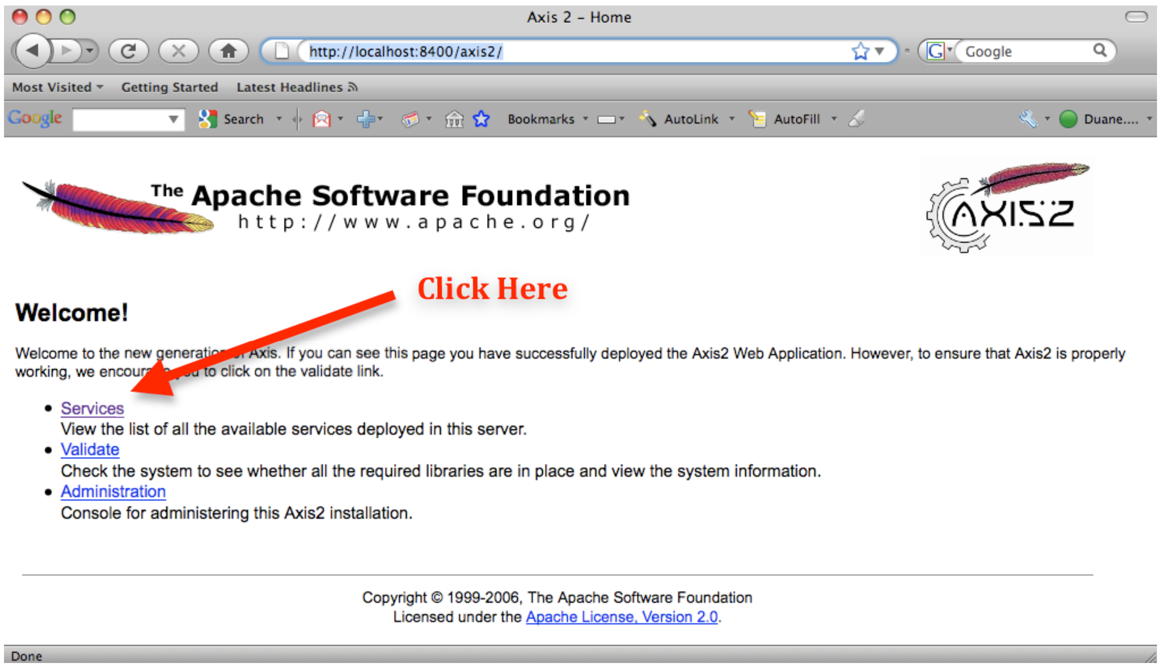
Flex applications support web service requests and results that are formatted as SOAP messages. SOAP provides the definition of the XML-based format that you can use for exchanging structured and typed information between a web service client, such as a Flex application, and a web service.

Adobe® Flash® Player operates within a security sandbox that limits what Flex applications and other applications built with Flash can access over HTTP. Applications built with Flash are allowed HTTP access only to resources on the same domain and by the same protocol from which they were served. This presents a problem for web services, because they are typically accessed from remote locations. The proxy service, available in LiveCycle Data Services ES and Vega, intercepts requests to remote web services, redirects the requests, and then returns the responses to the client.

If you are not using LiveCycle Data Services ES or Vega, you can access web services in the same domain as your Flex application; or a crossdomain.xml (cross-domain policy) file that allows access from your application's domain must be installed on the web server hosting the RPC service.

The Lab

We are going to use a WSDL, import it and wire it up to make a remote call. This will demonstrate the WSDL wizard that is based on Apache Axis 2. To run this lab you will need to ensure that you have the special version of BLazeDS up and running for this lab. To try this, grab a browser and hit the URL http://localhost:8400/axis2/ You should see a screen like below:



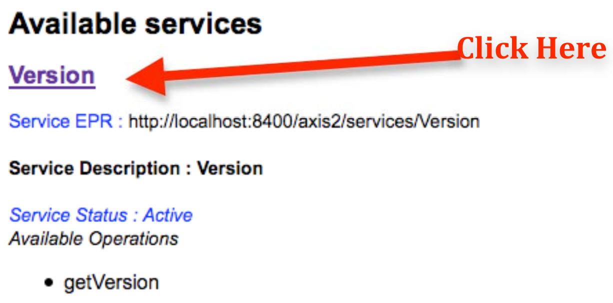
Welcome!

Welcome to the new generation of Axis. If you can see this page you have successfully deployed the Axis2 Web Application. However, to ensure that Axis2 is properly working, we encourage you to click on the validate link.

- Services
  View the list of all the available services deployed in this server.
- Validate
  Check the system to see whether all the required libraries are in place and view the system information.
- Administration
  Console for administering this Axis2 installation.

Copyright © 1999-2006, The Apache Software Foundation
Licensed under the Apache License, Version 2.0.

Click on the "Services" hyperlink and you should see the getVersion() operation which returns the version of Apache Axis being run on the server.



Available services

Version

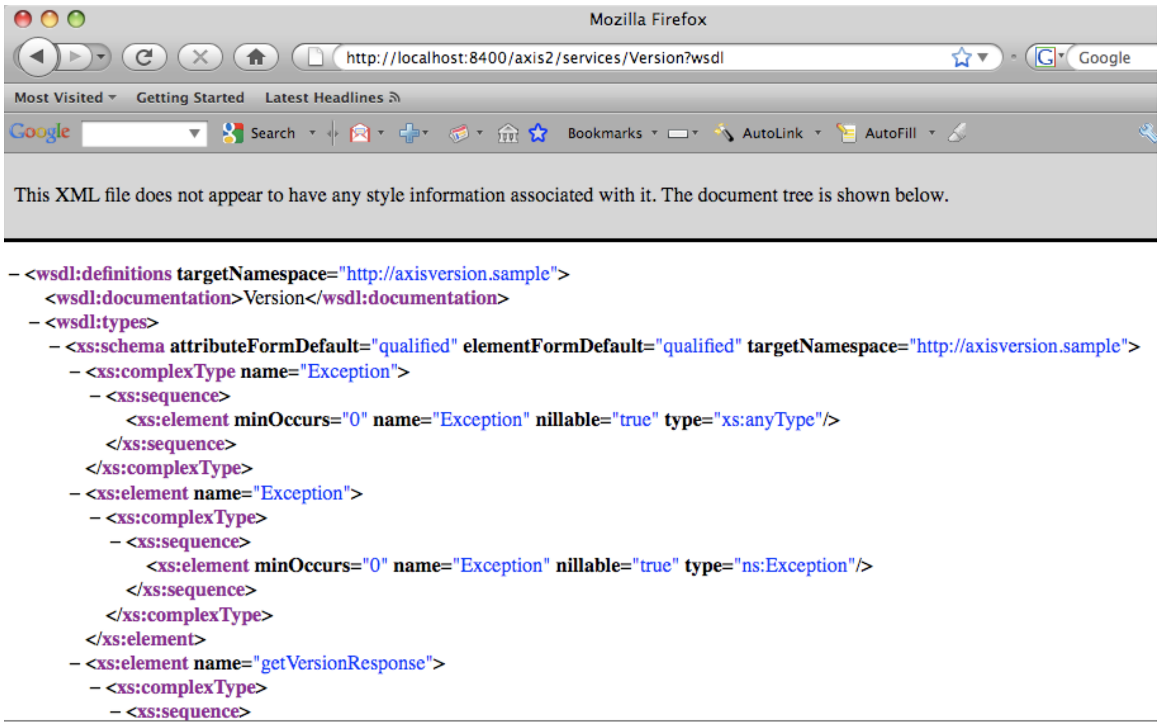
Service EPR : http://localhost:8400/axis2/services/Version

Service Description : Version

Service Status : Active
Available Operations

- getVersion

Click on the "Version" hyperlink and you should see the WSDL (Web Services Description Document) as seen below.



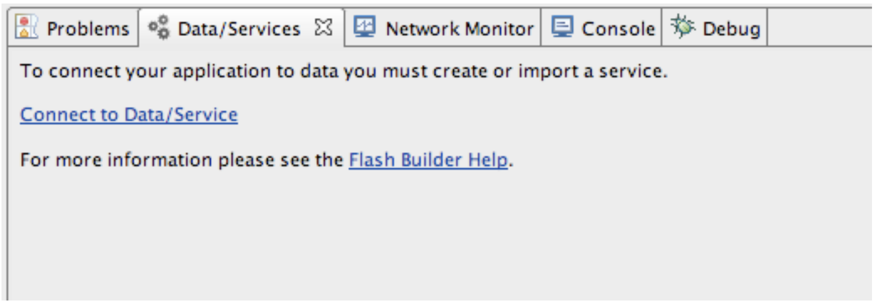
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
– <wsdl:definitions targetNamespace="http://axisversion.sample">
    <wsdl:documentation>Version</wsdl:documentation>
  – <wsdl:types>
    – <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="http://axisversion.sample">
      – <xs:complexType name="Exception">
        – <xs:sequence>
            <xs:element minOccurs="0" name="Exception" nillable="true" type="xs:anyType"/>
          </xs:sequence>
        </xs:complexType>
      – <xs:element name="Exception">
        – <xs:complexType>
          – <xs:sequence>
              <xs:element minOccurs="0" name="Exception" nillable="true" type="ns:Exception"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      – <xs:element name="getVersionResponse">
        – <xs:complexType>
          – <xs:sequence>
```

1. Make new project and name is ABC12-WebServices.

2. Make sure your local BlazeDS is still running.

3. Select ***Data/Services*** > ***Connect to Data/Service*** from the bottom menu in Flash Builder 4. This will open a dialog (wizard).

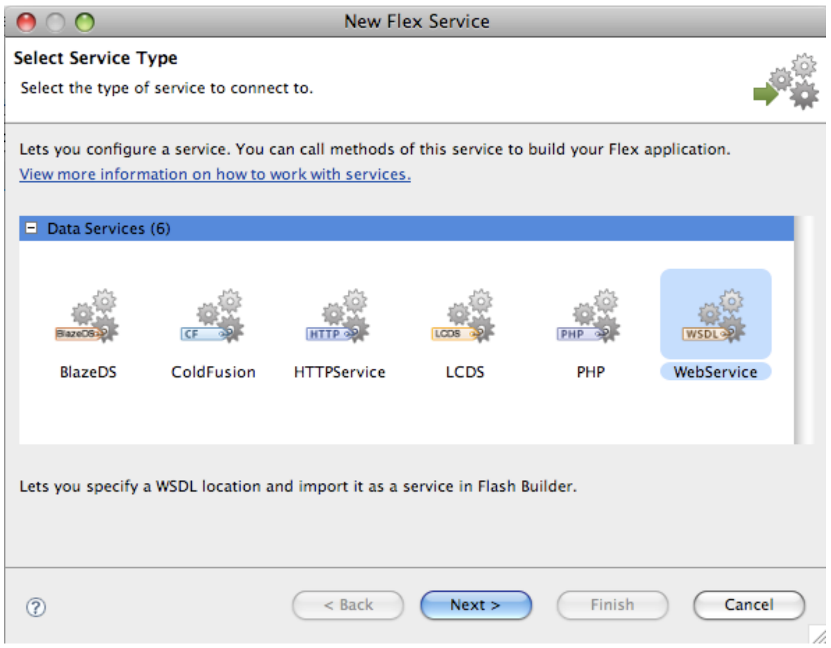


To connect your application to data you must create or import a service.

Connect to Data/Service

For more information please see the Flash Builder Help.

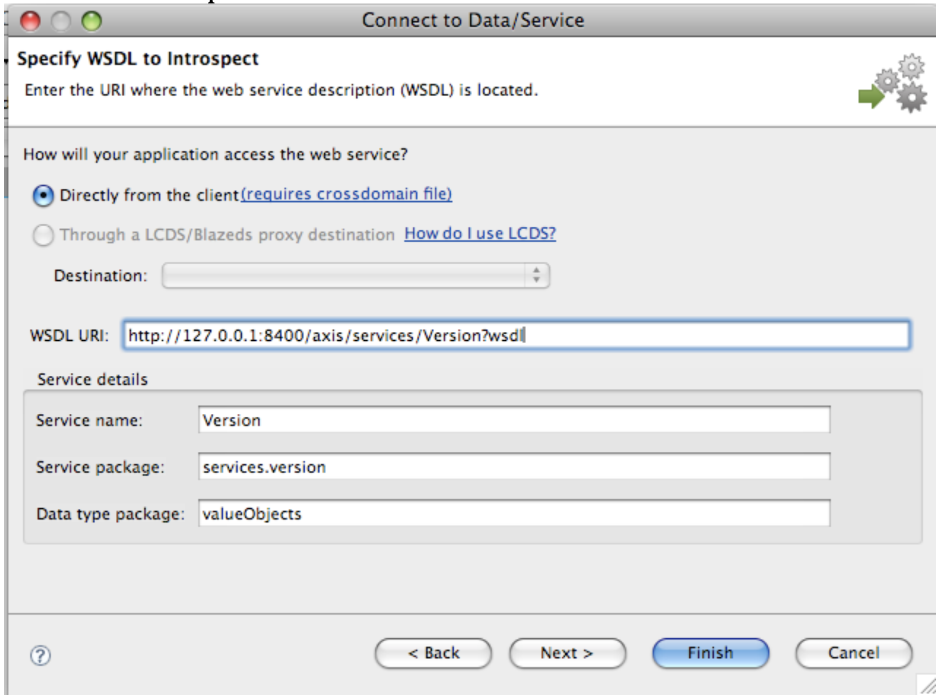
4. In the next dialog box, give your service a name and also enter the URL of the WSDL so Flash Builder 4 can inspect it as shown below. Hit Next.



Lets you configure a service. You can call methods of this service to build your Flex application.
View more information on how to work with services.

Lets you specify a WSDL location and import it as a service in Flash Builder.

5. Cut and paste the URL of the WSDL into the "WSDL URI" box as shown below.



Enter the URI where the web service description (WSDL) is located.

How will your application access the web service?

[✓ Directly from the client] (requires crossdomain file)

Through a LCDS/Blazeds proxy destination How do I use LCDS?

Destination:

WSDL URI: http://127.0.0.1:8400/axis/services/Version?wsdl

Service details

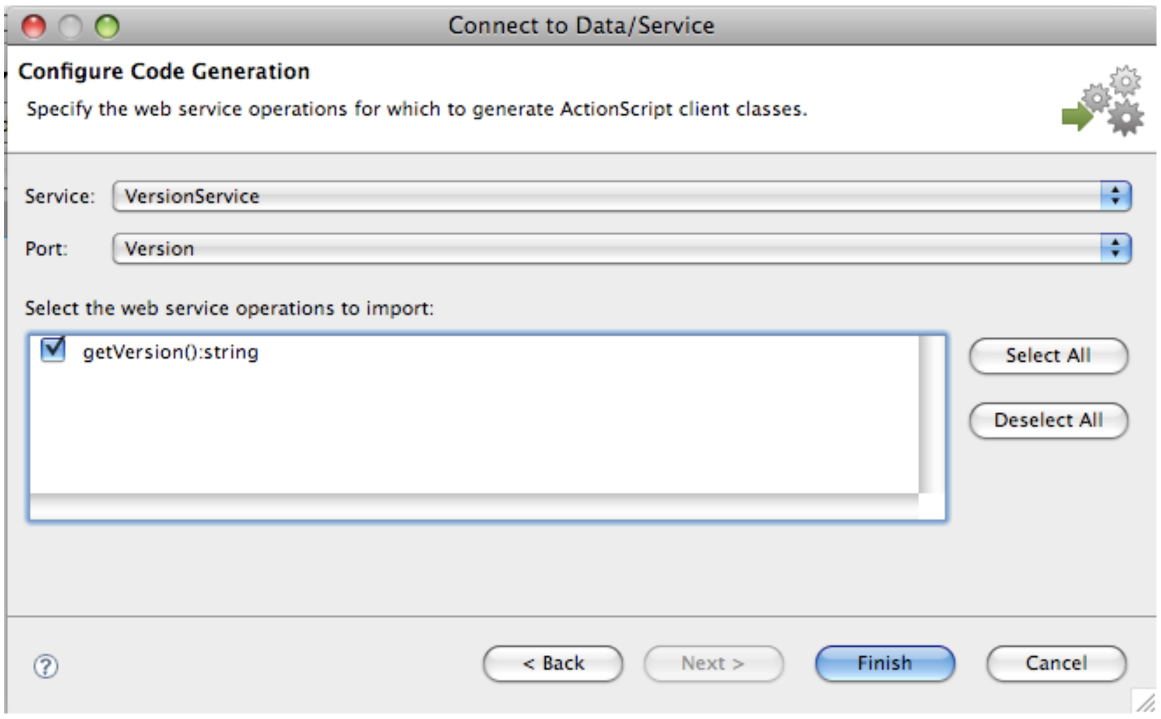
Service name: Version

Service package: services.version

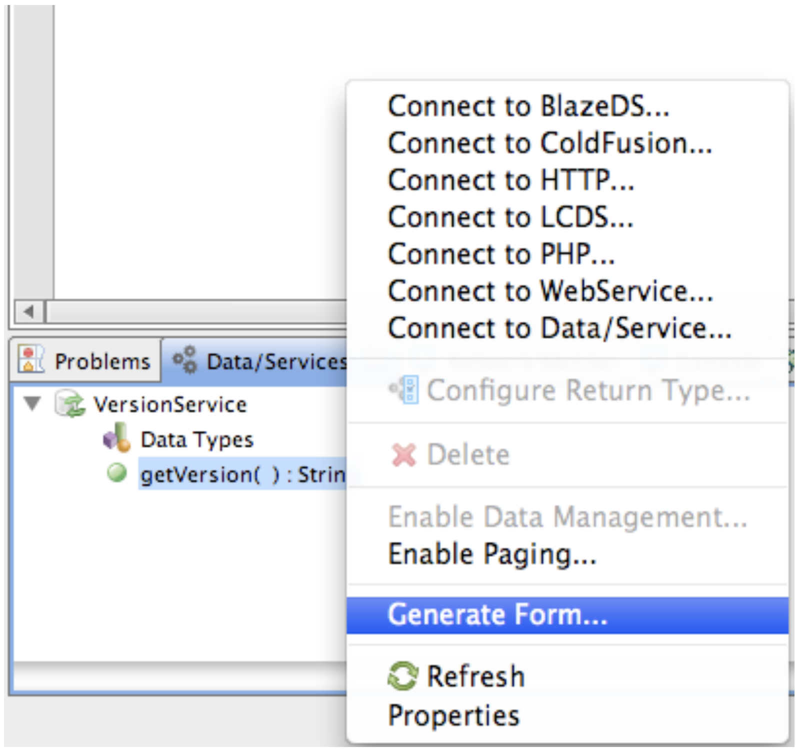
Data type package: valueObjects

6. Hit "Next".

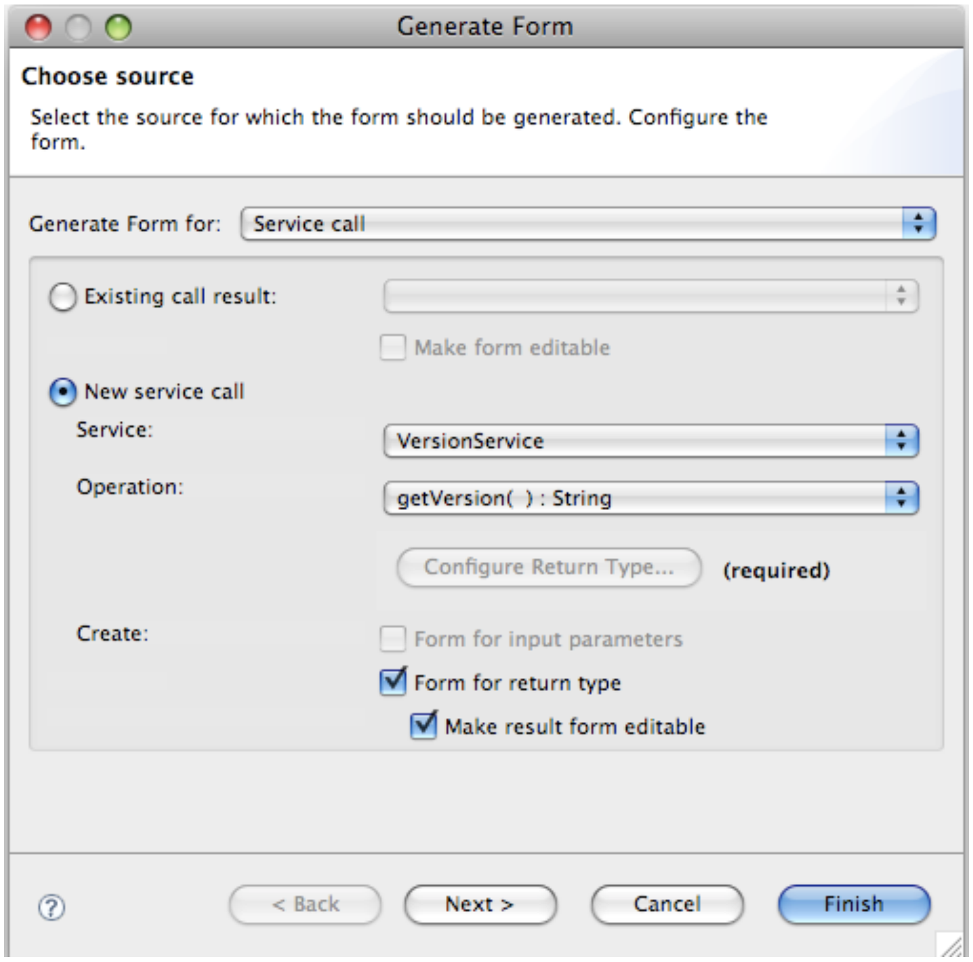
7. The next dialog will appear after Flash Builder has inspected the WSDL and will offer you a choice of ports and services. By default, you will not have to change anything. Click "Finish".


Connect to Data/Service
Configure Code Generation
Specify the web service operations for which to generate ActionScript client classes.
Service: VersionService
Port: Version
Select the web service operations to import:
getVersion():string
Select All
Deselect All
< Back
Next >
Finish
Cancel

8. You now have to bind the return from the web service call to a visual component. To do this, right click (PC and Linux) or Command Click (Mac) on the getVersion() service in the Data/Services tab and select "generate form" as shown below.



9. A dialog box will pop up giving you some options. The options will allow you to select operation parameters and settings. Leave the defaults as shown below:


Generate Form

Choose source

Select the source for which the form should be generated. Configure the form.

Generate Form for: Service call

( ) Existing call result:

[ ] Make form editable

(•) New service call

Service: VersionService

Operation: getVersion( ) : String

Configure Return Type... **(required)**

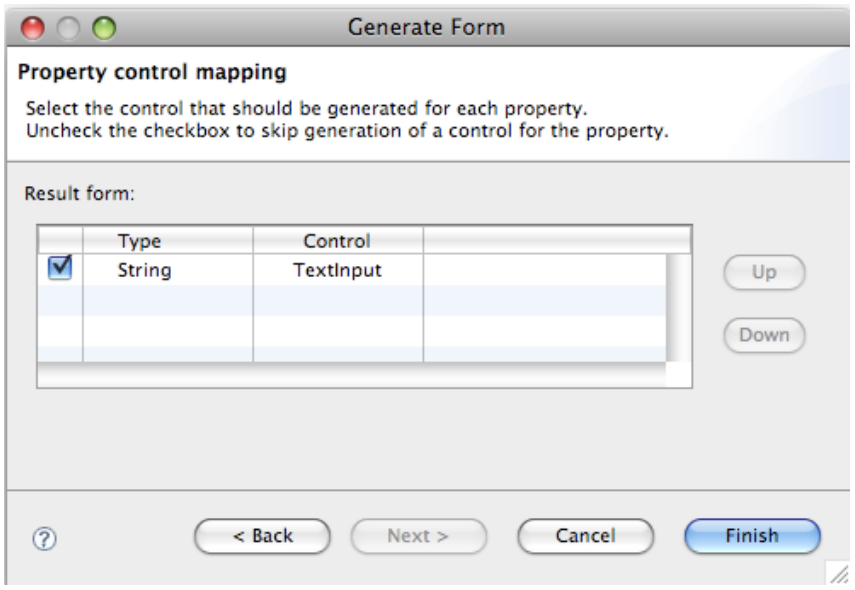
Create:

[ ] Form for input parameters

[✓] Form for return type

[✓] Make result form editable

10. Hit "Next" and the next dialog allows you to control where each returned object maps to in terms of form components. In this case, we are simply allowing the String (the value returned from the call) to be bound to a TextInput. Select "Finish".


Generate Form

Property control mapping

Select the control that should be generated for each property.
Uncheck the checkbox to skip generation of a control for the property.

Result form:

	Type	Control
✓	String	TextInput

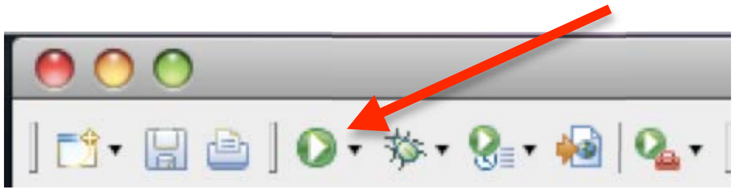
11. The last step is to perform a minor tweak to the textInput to ensure it is big enough to hold the string value. Switch to source code view and change this line:

```
<mx:FormItem label="GetVersion">
       <s:TextInput id="getVersionTextInput"
                    text="{getVersionResult.lastResult as String}"/>
</mx:FormItem>
```

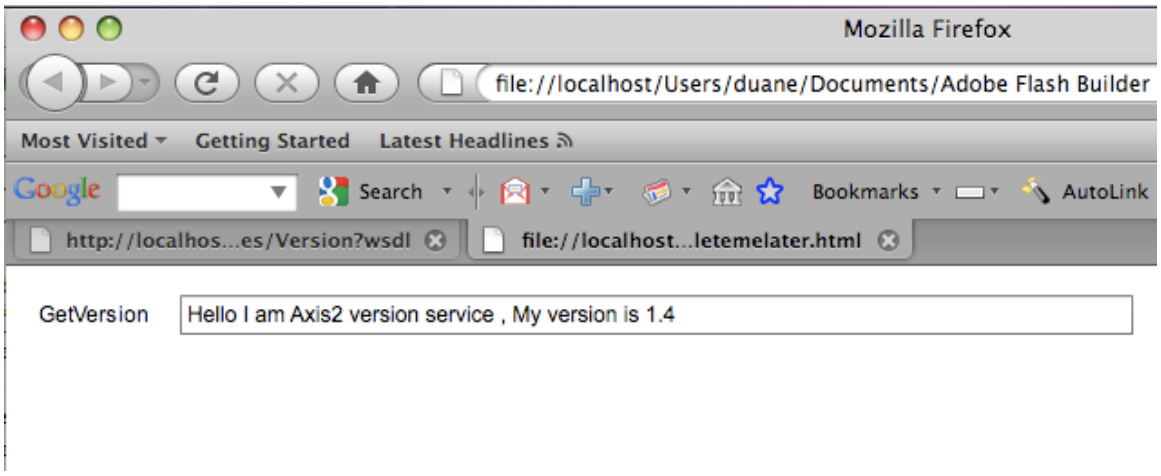
12. To add the width attributes as shown below:

```
<mx:Form id="form" creationComplete="form_creationCompleteHandler(event)">
   <mx:FormItem label="GetVersion">
       <s:TextInput id="getVersionTextInput" text="{getVersionResult.lastResult
                    as String}" width="207" height="68"/>
   </mx:FormItem>
</mx:Form>
```

14: Now run the program by clicking the green arrow icon.



15: You should see the web service result in your application.


Mozilla Firefox
file://localhost/Users/duane/Documents/Adobe Flash Builder
Most Visited
Getting Started
Latest Headlines
Search
Bookmarks
AutoLink
http://localhos...es/Version?wsdl
file://localhost...letemelater.html

GetVersion Hello I am Axis2 version service , My version is 1.4

Project 12: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/halo" minWidth="1024"
minHeight="768" xmlns:versionservice="services.versionservice.*">
      <fx:Script>
      <![CDATA[
      import mx.events.FlexEvent;
      import mx.controls.Alert;

      protected function
            form_creationCompleteHandler(event:FlexEvent):void
            {
            getVersionResult.token = versionService.getVersion();
            }
      ]]>
      </fx:Script>
      <fx:Declarations>
            <versionservice:VersionService id="versionService"
fault="Alert.show(event.fault.faultString)" showBusyCursor="true"/>
            <s:CallResponder id="getVersionResult"/>
      </fx:Declarations>
      <mx:Form id="form"
creationComplete="form_creationCompleteHandler(event)" width="651">
      <mx:FormItem label="GetVersion" width="607">
```

```
            <s:TextInput id="getVersionTextInput"
text="{getVersionResult.lastResult as String}" width="526"/>
      </mx:FormItem>
      </mx:Form>
</s:Application>
```

Project 12: Test Questions

Q: How can the call be controlled by a button?

Q: Briefly describe protected as a function scope attribute keyword and how it differs from public and private?

Project 12a: Alternative Web Service

Web Services are an integral part of the SOA infrastructure and also a preferred way of exposing services, especially in the Enterprise. The new Services Wizards feature in Flash Builder 4 makes it extremely simple to work with Web Services.

To work with the Services Wizard which is part of the Data Centric Design feature in Flash Builder 4 (for connecting to HTTPService, Web Services or Remoting using any back-end technologies like Java, PHP, CF… etc) you need to do the following:

1. Add the service
2. Configure Return Type (create client-side Model classes by inspecting the service)
3. Call the service and bind it to UI Controls

Before I start to explain how to access Web Service (WS) from Flash Builder, let me discuss some aspects of the WS Architecture. A WSDL(Web Service Description Language) Document is what signify how a WS is exposed. On a high-level, each WS has the following:

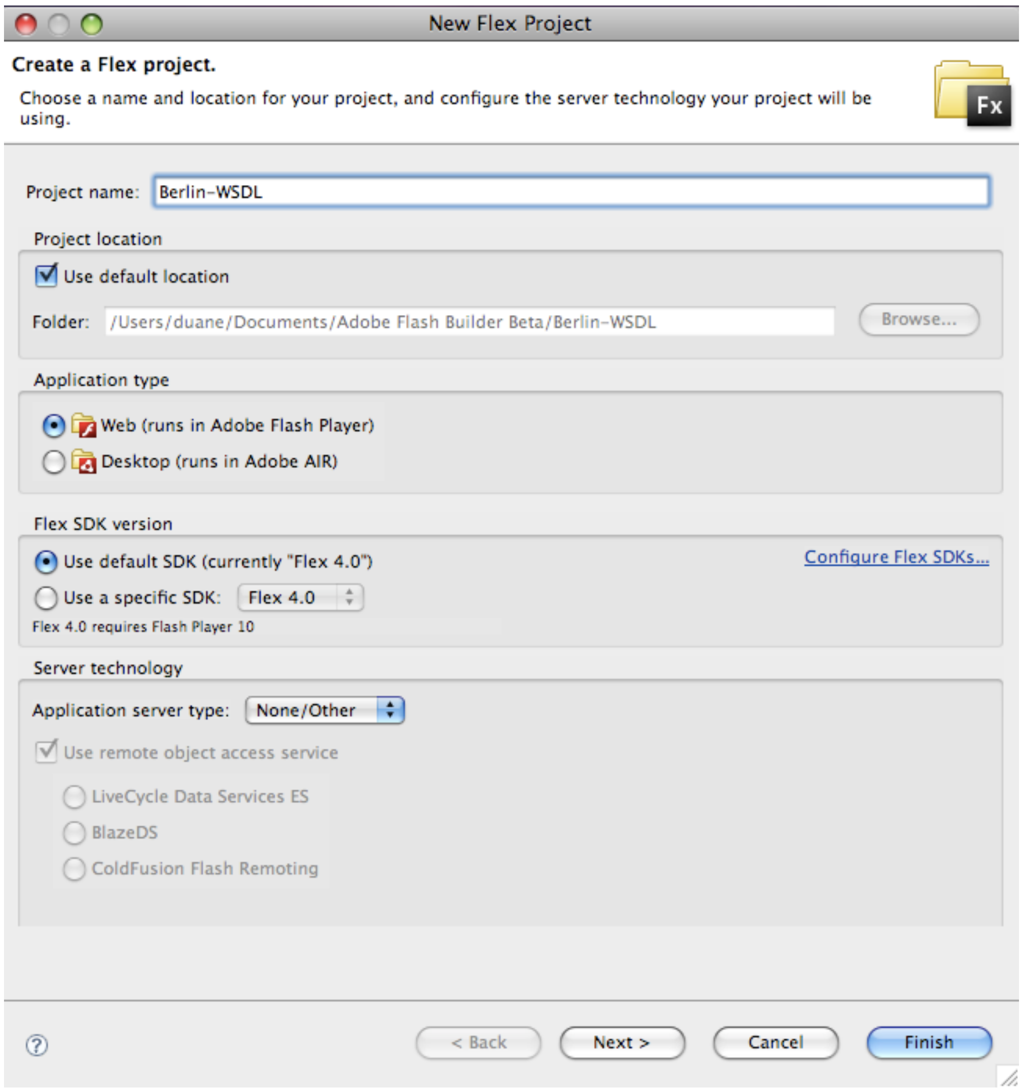
- Elements
- Complex-types
- Messages
- Operations

Each operation will have an input & output message and each message may be built with an element or complex-types. We will be using this Population WS (click to see the WSDL file) for serving data to our application. We will use the following operations:

- getCountries – to get the various countries for which data is available.
- getPopulation – for getting the population of a selected country.

Lab:

1. Create a new project.


New Flex Project

Create a Flex project.

Choose a name and location for your project, and configure the server technology your project will be using.

Project name: Berlin-WSDL

Project location

[✓] Use default location

Folder: /Users/duane/Documents/Adobe Flash Builder Beta/Berlin-WSDL

Application type

[● Web (runs in Adobe Flash Player)]
○ Desktop (runs in Adobe AIR)

Flex SDK version

[● Use default SDK (currently "Flex 4.0")]
○ Use a specific SDK: Flex 4.0

Flex 4.0 requires Flash Player 10

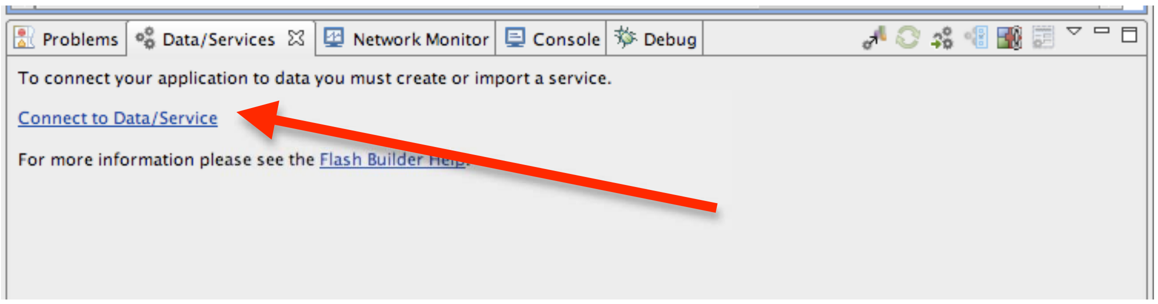
Server technology

Application server type: None/Other

[✓] Use remote object access service

○ LiveCycle Data Services ES
○ BlazeDS
○ ColdFusion Flash Remoting

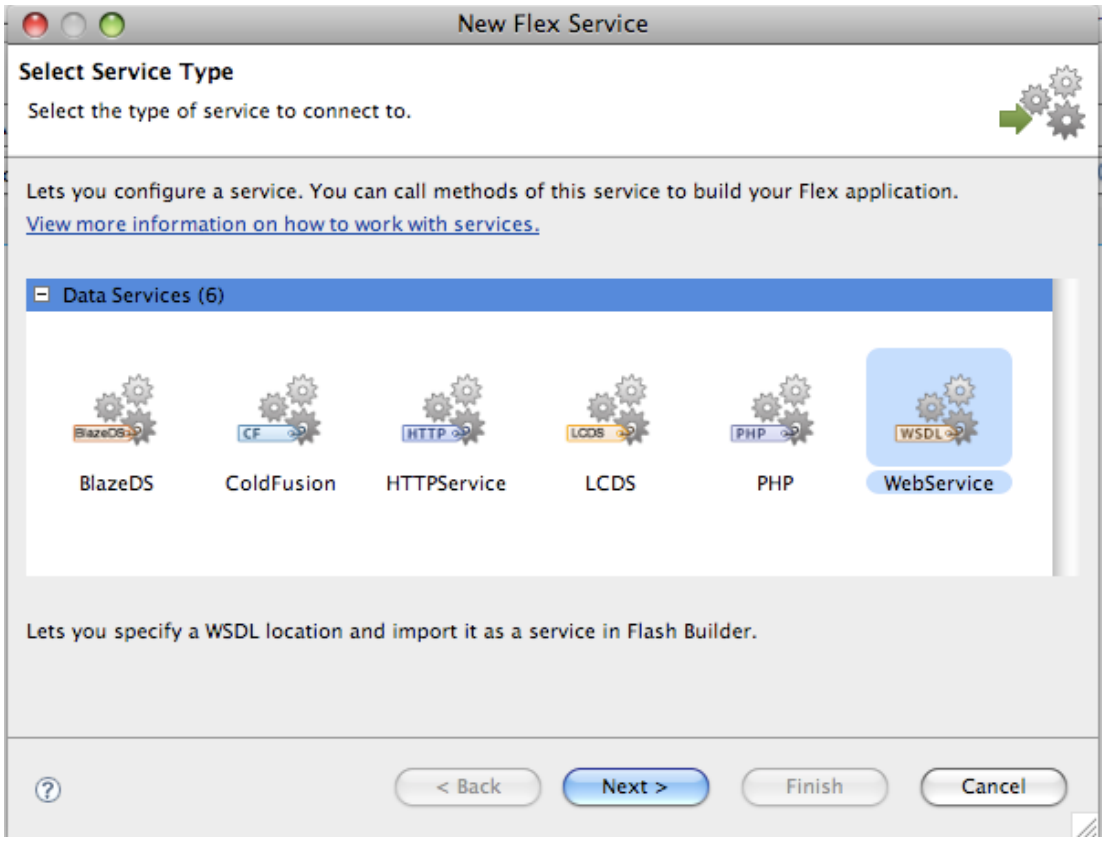
1. Hit Finish
2. Import the Service. In your Flash Builder 4 environment, you will see the new Data/Services Tab like below:


To connect your application to data you must create or import a service.

Connect to Data/Service

For more information please see the Flash Builder Help.

3. Click on “Connect to Data/ Service” and choose WebService from the pop-up


New Flex Service

Select Service Type

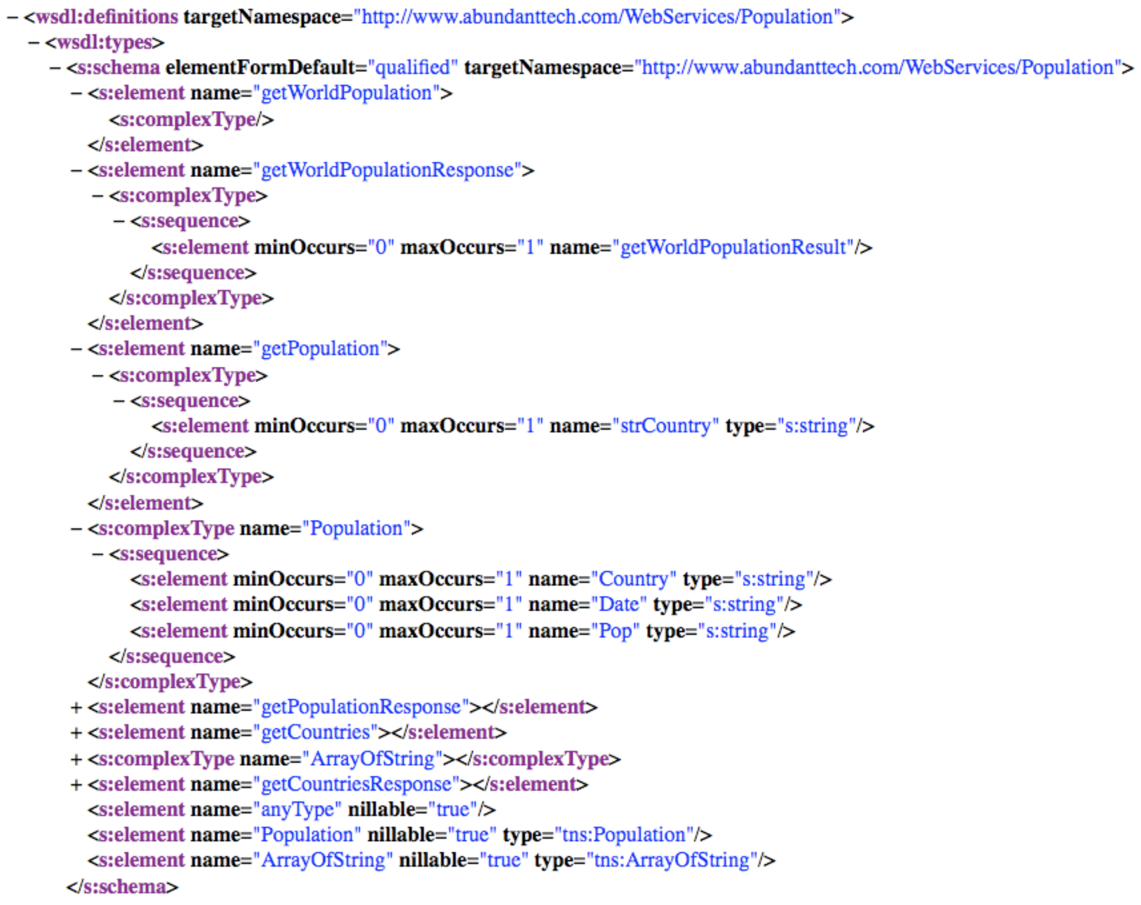
Select the type of service to connect to.

Lets you configure a service. You can call methods of this service to build your Flex application.
View more information on how to work with services.

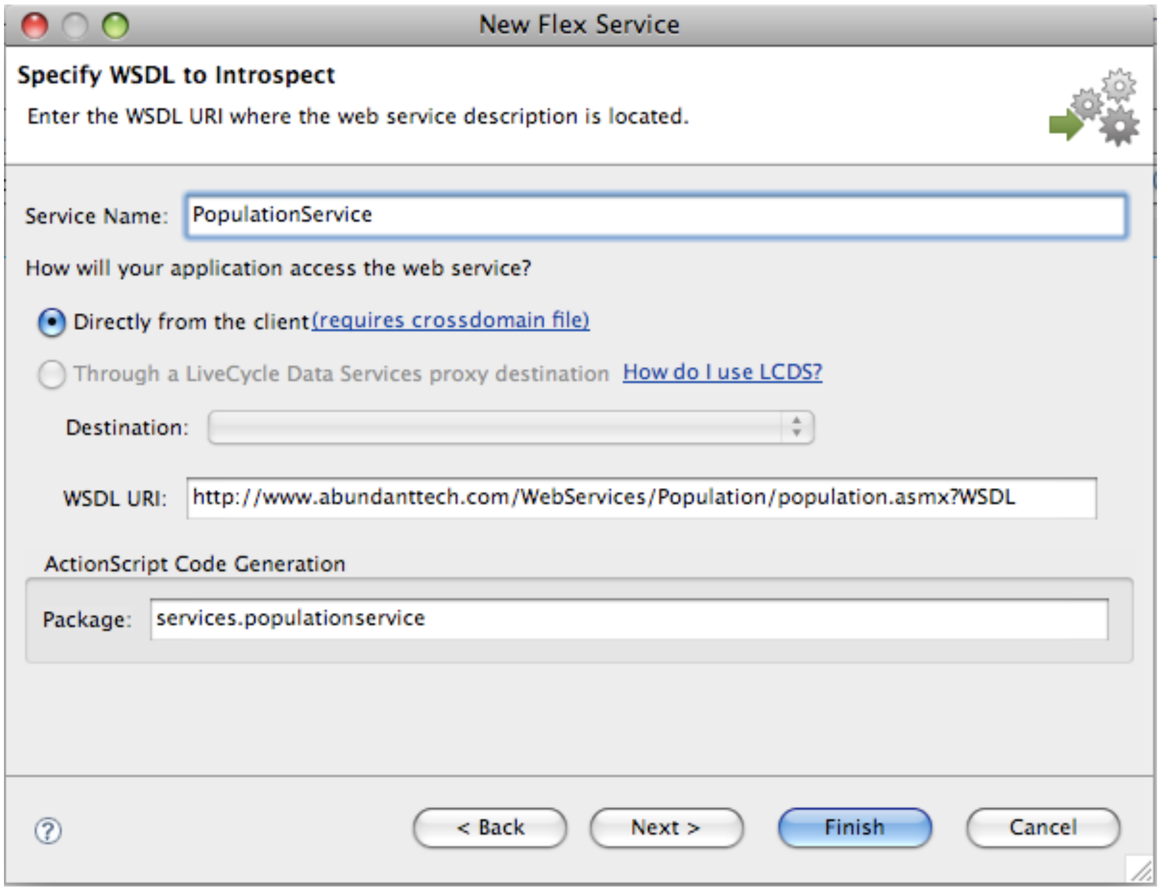
Lets you specify a WSDL location and import it as a service in Flash Builder.

4. In the next screen, enter the Service Name (that you want it to be called in your Flex application) and path to WSDL file, which in this case is:

   http://www.abundanttech.com/WebServices/Population/population.asmx?WSDL



```
- <wsdl:definitions targetNamespace="http://www.abundanttech.com/WebServices/Population">
  - <wsdl:types>
    - <s:schema elementFormDefault="qualified" targetNamespace="http://www.abundanttech.com/WebServices/Population">
      - <s:element name="getWorldPopulation">
          <s:complexType/>
        </s:element>
      - <s:element name="getWorldPopulationResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="getWorldPopulationResult"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="getPopulation">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="strCountry" type="s:string"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:complexType name="Population">
        - <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="Country" type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="Date" type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="Pop" type="s:string"/>
          </s:sequence>
        </s:complexType>
      + <s:element name="getPopulationResponse"></s:element>
      + <s:element name="getCountries"></s:element>
      + <s:complexType name="ArrayOfString"></s:complexType>
      + <s:element name="getCountriesResponse"></s:element>
        <s:element name="anyType" nillable="true"/>
        <s:element name="Population" nillable="true" type="tns:Population"/>
        <s:element name="ArrayOfString" nillable="true" type="tns:ArrayOfString"/>
      </s:schema>
```


New Flex Service

Specify WSDL to Introspect

Enter the WSDL URI where the web service description is located.

Service Name: PopulationService

How will your application access the web service?

[✓ Directly from the client] (requires crossdomain file)

Through a LiveCycle Data Services proxy destination How do I use LCDS?

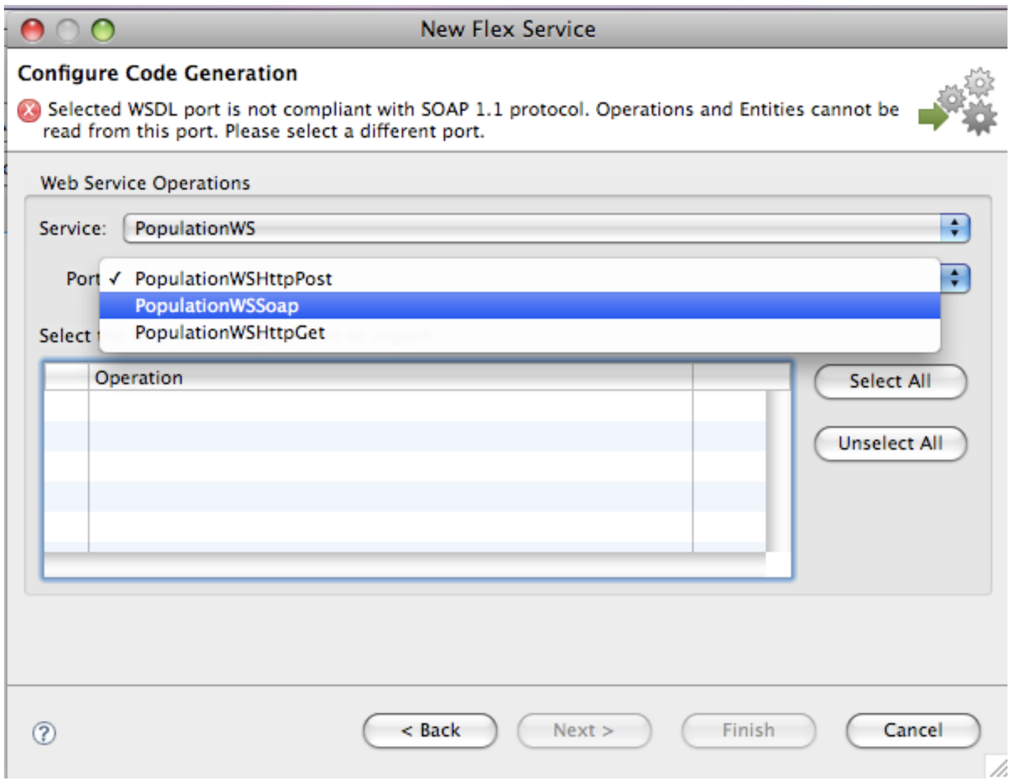
Destination:

WSDL URI: http://www.abundanttech.com/WebServices/Population/population.asmx?WSDL

ActionScript Code Generation

Package: services.populationservice

5. If your WSDL is SOAP 1.1 protocol compliant, it will directly show the operations screen (the screenshot after the one below). Otherwise, it will ask you to choose the appropriate SOAP port as below. Remember this screen will only appear if your WDSL is not WSDL1.1 compliant.



New Flex Service

Configure Code Generation

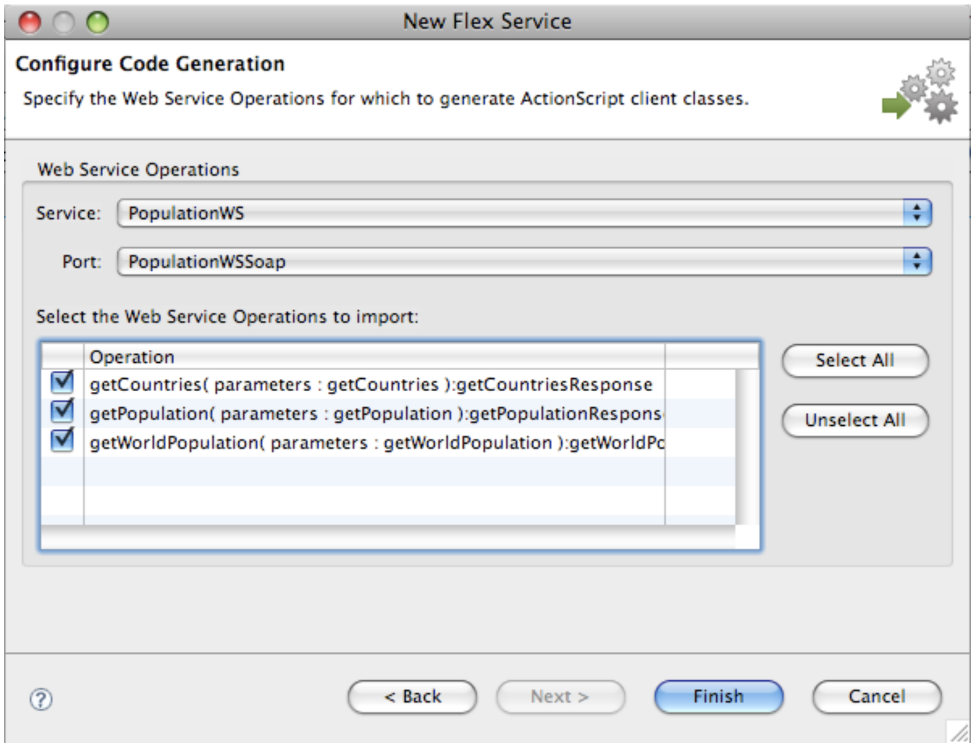
Selected WSDL port is not compliant with SOAP 1.1 protocol. Operations and Entities cannot be read from this port. Please select a different port.

Web Service Operations

Service: PopulationWS

Port: ✓ PopulationWSHttpPost / PopulationWSSoap / PopulationWSHttpGet

6. Then you can choose the operations that you are interested in. You can either choose just the operations you are interested in or press Select All and press "Finish".


New Flex Service

Configure Code Generation

Specify the Web Service Operations for which to generate ActionScript client classes.

Web Service Operations

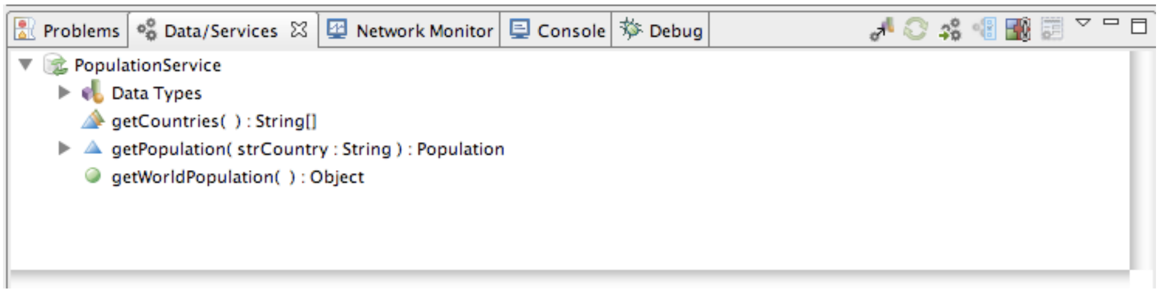
Service: PopulationWS

Port: PopulationWSSoap

Select the Web Service Operations to import:

	Operation
✓	getCountries(parameters : getCountries):getCountriesResponse
✓	getPopulation(parameters : getPopulation):getPopulationRespons
✓	getWorldPopulation(parameters : getWorldPopulation):getWorldPc

7. This will give you a Data/Services tab with all the operations imported and configured…



```
PopulationService
  Data Types
  getCountries( ) : String[]
  getPopulation( strCountry : String ) : Population
  getWorldPopulation( ) : Object
```

8. Flash Builder introspects the WSDL and the selected operations and generates the required required classes. But you will notice that while the return type for getPopulation() is created as a client side model class called Population, getWorldPopulation() has a return type of Object. The introspecter does the following:

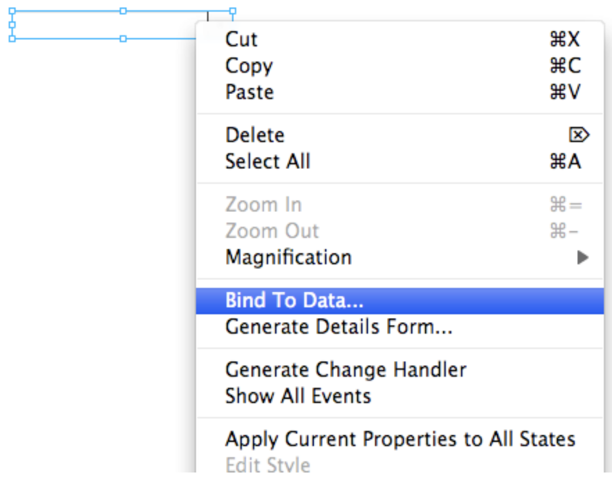
   a. If the operation is well formed Flash Builder 4 creates the required classes (as in the case of getPopulation() method it creates a Population

Class)

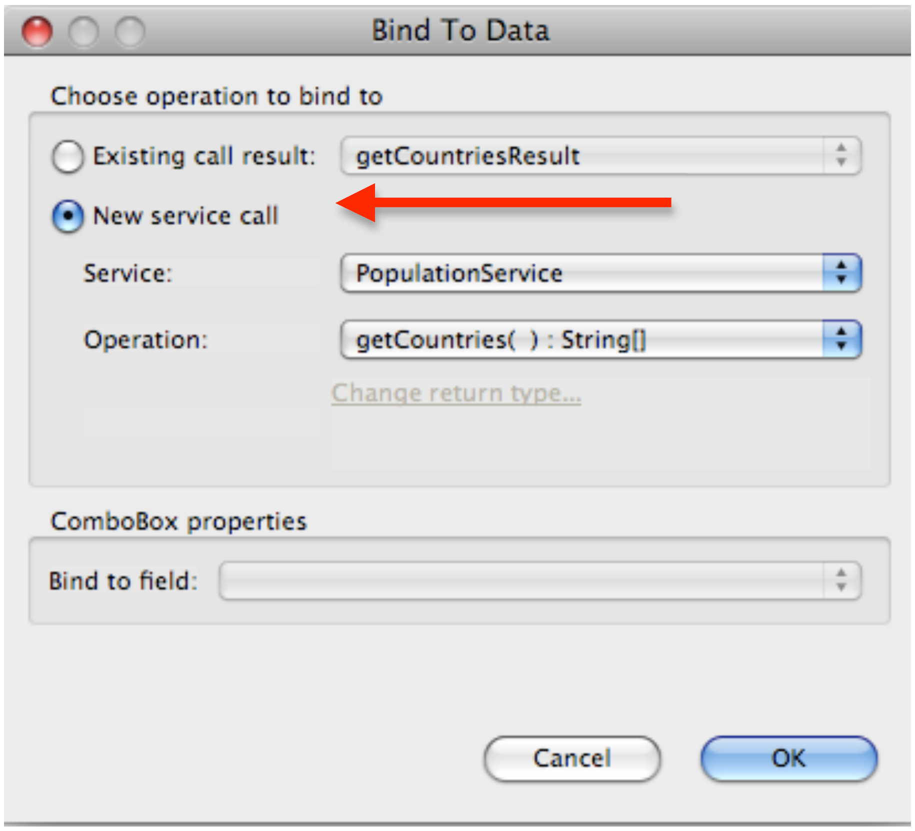
b. If the operation is not well formed then Flash Builder 4 keeps the return type as Object (as in the case of getWorldPopulation() Method)

Since WSDL itself defines the input and output types, it will create the required classes provided the operations are well formed.

9. Build the GUI. Go to the design view and drag and drop a ComboBox. Right click and click on "Bind to Data".



10. This throws a wizard where you can choose the service you want to bind it to, which in this case is getCountries(). Be sure to unselect the "Existing Call Result" and select "New Service Call" instead as shown below:



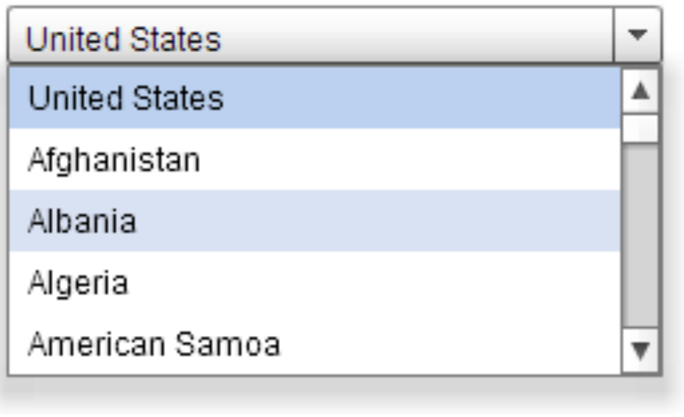
( ) Existing call result: getCountriesResult

(•) New service call

Service: PopulationService

Operation: getCountries( ) : String[]

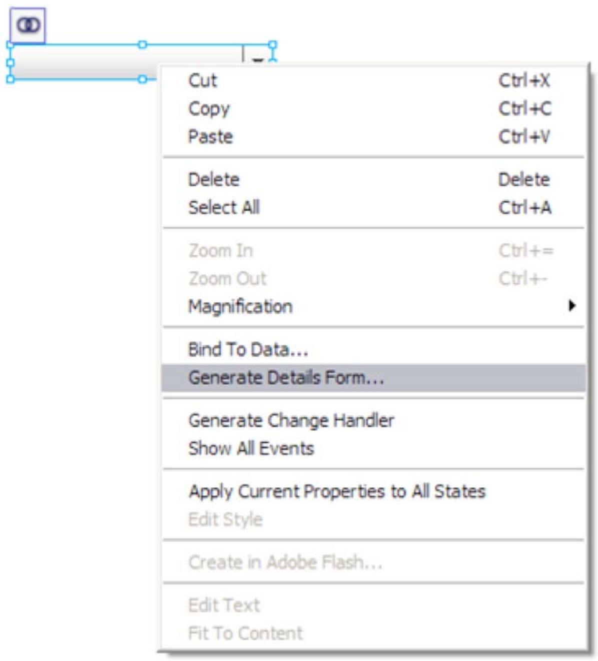
11. Click "OK" and run the application. You should see the following



United States

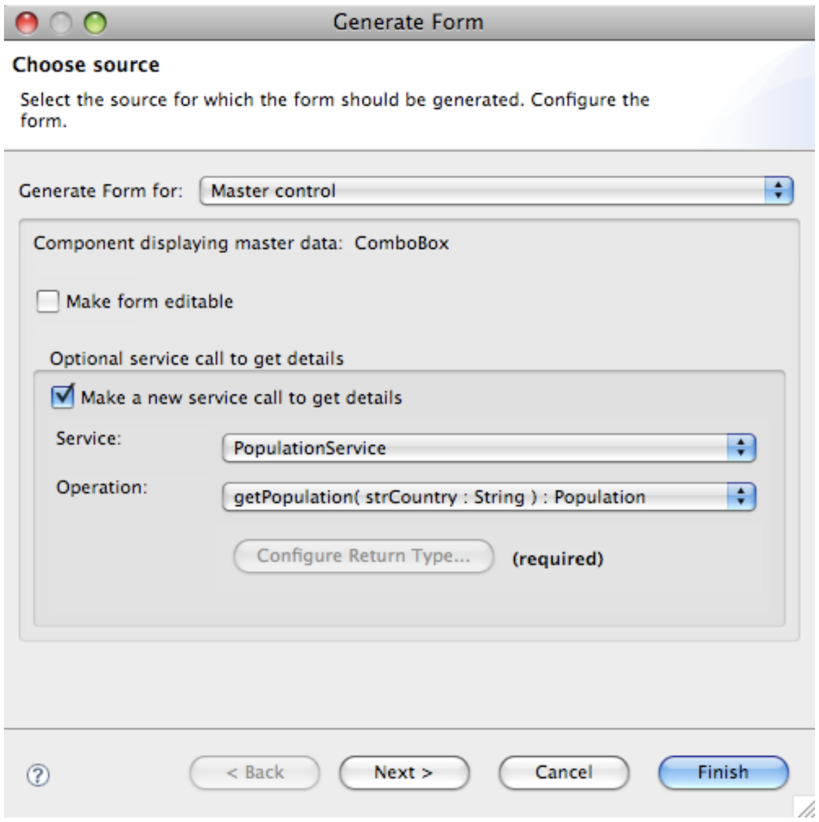
United States
Afghanistan
Albania
Algeria
American Samoa

12. Now lets get the population details for a selected item on the combobox. This is simple. Click on the ComboBox and click "Generate Details Form"



‘

13. This throws a pop-up where you can choose which operation to call on change and what to display.



Choose source

Select the source for which the form should be generated. Configure the form.

Generate Form for: Master control

Component displaying master data: ComboBox

☐ Make form editable

Optional service call to get details

✓ Make a new service call to get details

Service: PopulationService

Operation: getPopulation(strCountry : String) : Population

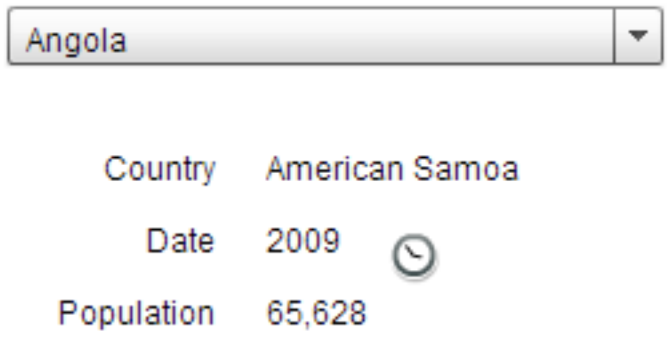
14. It automatically redirects to the code view and ask you to provide the parameter for the getPopulation() method. Change

```
…
protected function comboBox_changeHandler(event:ListEvent):void
…
{ getPopulationResult.token =
populationService.getPopulation(strCountry);}
…
```

To

```
protected function comboBox_changeHandler(event:ListEvent):void

{getPopulationResult.token =
populationService.getPopulation(comboBox.selectedItem.toString());}
```

15. Now go to design view, reposition the generated Form and run the app and we have a fully functional application served by a webservice. Just to make it pretty, I added a prompt to the ComboBox.



Project 12a: Solution code

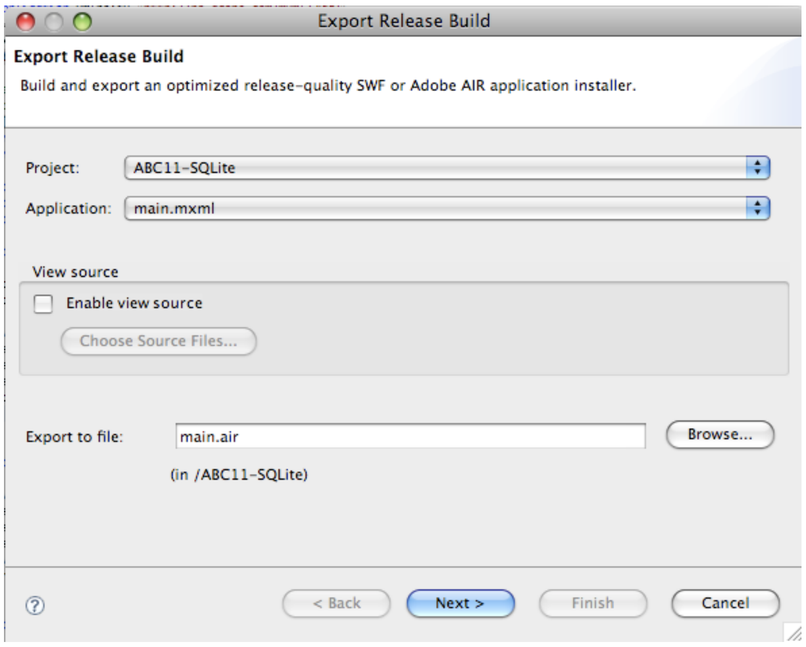
```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                       xmlns:s="library://ns.adobe.com/flex/spark"
                       xmlns:mx="library://ns.adobe.com/flex/halo"
                       xmlns:populationservice="services.populationservice.*"
                       minWidth="1024" minHeight="768" >
       <fx:Declarations>
              <s:CallResponder id="getCountriesResult"/>
              <populationservice:PopulationService id="populationService"
               fault="Alert.show(event.fault.faultString)"
showBusyCursor="true"/>
              <s:CallResponder id="getCountriesResult2"/>
              <s:CallResponder id="getCountriesResult3"/>
              <populationservice:Population id="population"/>
              <s:CallResponder id="getPopulationResult"
                  result="population = getPopulationResult.lastResult as
Population"/>
       </fx:Declarations>

       <fx:Script>
              <![CDATA[
                     import mx.events.ListEvent;
                     import mx.events.FlexEvent;
```

```
                    import mx.controls.Alert;

                    protected function
comboBox_creationCompleteHandler(event:FlexEvent):void
                    {
                          getCountriesResult.token =
populationService.getCountries();
                          getCountriesResult2.token =
populationService.getCountries();
                          getCountriesResult3.token =
populationService.getCountries();
                    }

                    protected function
comboBox_changeHandler(event:ListEvent):void

                    {getPopulationResult.token =
populationService.getPopulation(comboBox.selectedItem.toString());}

                    /*protected function
comboBox_changeHandler(event:ListEvent):void
                    {
                    getPopulationResult.token =
populationService.getPopulation(strCountry);
                    }*/

             ]]>
      </fx:Script>

      <mx:ComboBox x="214" y="132" editable="true" id="comboBox"
         creationComplete="comboBox_creationCompleteHandler(event)"
         dataProvider="{getCountriesResult3.lastResult}"
         change="comboBox_changeHandler(event)"/>
      <mx:Form x="215" y="171">
             <mx:FormItem label="Country">
                    <mx:Text id="countryText" text="{population.Country}"/>
             </mx:FormItem>
             <mx:FormItem label="_Date">
                    <mx:Text id="_DateText" text="{population._Date}"/>
             </mx:FormItem>
             <mx:FormItem label="Pop">
                    <mx:Text id="popText" text="{population.Pop}"/>
             </mx:FormItem>
      </mx:Form>


</s:Application>
```

Project 13: Exporting, Signing, Distributing and installing AIR applications

This exercise is to demonstrate how to export, sign and distribute AIR applications. Use the preceding project (keep loaded in FB3) and take the following steps.

1. Click "File -> Export -> Release Build" The following dialog will appear. Select your project and click "next".



Export Release Build

Export Release Build

Build and export an optimized release-quality SWF or Adobe AIR application installer.

Project: ABC11-SQLite

Application: main.mxml

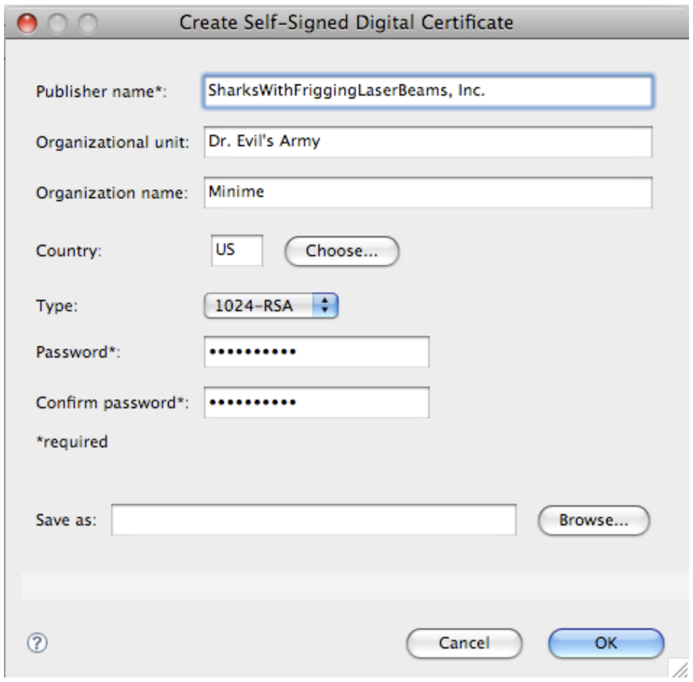
View source

☐ Enable view source

Export to file: main.air

(in /ABC11-SQLite)

2. You can now sign your application. For production, you will want a real certificate registered with a CA working with the AIR runtime. For now, select "Create". Fill out the following dialog box. NOTE: Self signed certificates are basically worthless. Only a top level CA issued certificate should be trusted.


Create Self-Signed Digital Certificate

Publisher name*: SharksWithFriggingLaserBeams, Inc.

Organizational unit: Dr. Evil's Army

Organization name: Minime

Country: US Choose...

Type: 1024-RSA

Password*: ••••••••••

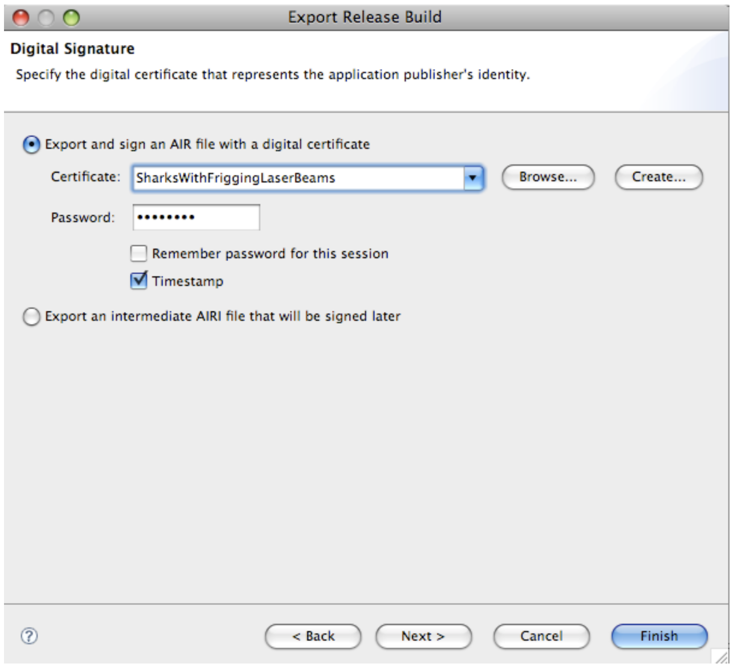
Confirm password*: ••••••••••

*required

Save as: Browse...

Cancel OK

3. Pick your cert and sign it. Click Next. Note: In a real production environment, you would select the CA assured cert.


Export Release Build

Digital Signature

Specify the digital certificate that represents the application publisher's identity.

(●) Export and sign an AIR file with a digital certificate

Certificate: SharksWithFriggingLaserBeams Browse... Create...

Password: ••••••••

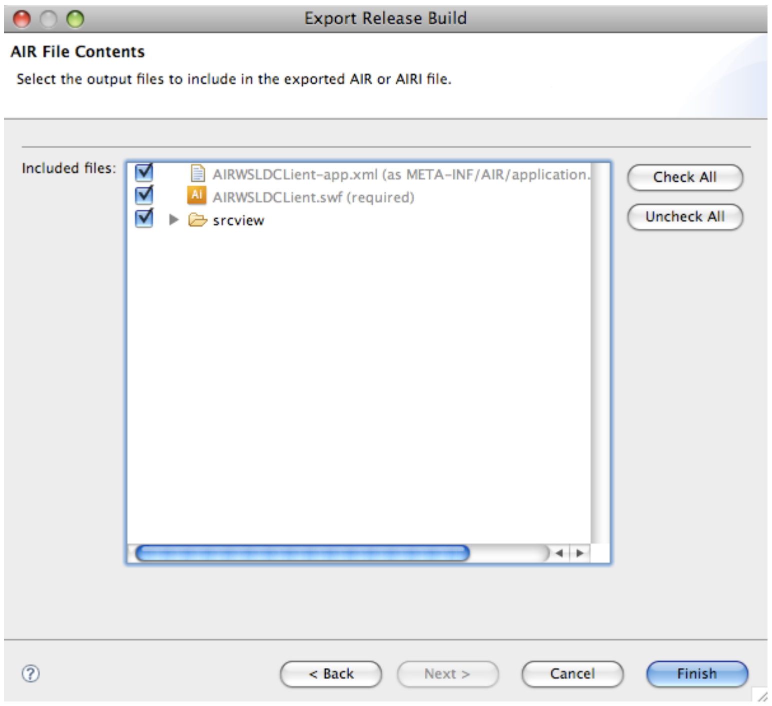
☐ Remember password for this session

☑ Timestamp

( ) Export an intermediate AIRI file that will be signed later

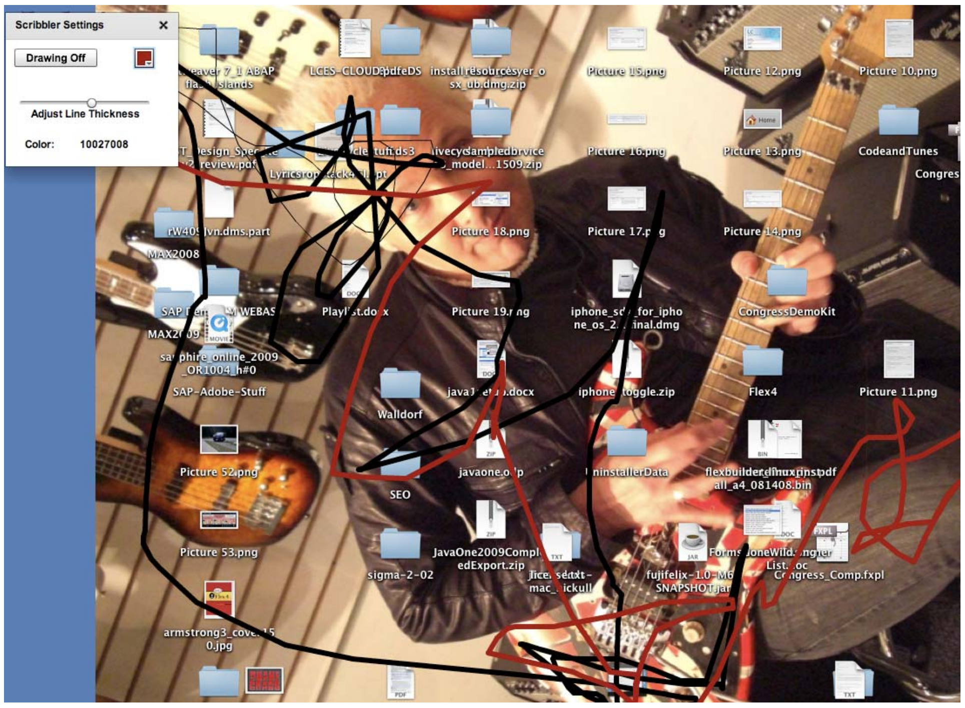
< Back Next > Cancel Finish

4. Select the parts you want to export and client “finish”


Export Release Build
AIR File Contents
Select the output files to include in the exported AIR or AIRI file.
Included files:
AIRWSLDCLient-app.xml (as META-INF/AIR/application.
AIRWSLDCLient.swf (required)
srcview
Check All
Uncheck All
< Back
Next >
Cancel
Finish

5. The *.AIR file is now ready to be distributed. Find it and install it.

Project 14: ScribblerApplication



This project will demonstrate how to draw all over the screen while in FULL_SCREEN_INTERACTIVE mode.

1. Start a new AIR Project and call it ABC14-ScribblerApp
2. Open the application descriptor file and set the application to be no chrome and transparent (look at the second lab – ChromelessApps, if you don't remember how)
3. Make sure your root element is <mx:> namespace qualified, not Spark.
4. Add the two lines of code to the root element

```
applicationComplete="init()"  showFlexChrome="false">
```

5. Add the following graphical components to your code. Note that the backgroundAlpha is something you may have to tweak for specific builds of Flash Builder and AIR. If your application does not run properly, change the value to be less transparent.

```
<mx:Canvas id="myCanvas" width="100%" height="100%" fontWeight="bold"
backgroundAlpha="0.00001"  backgroundColor="0xFF44FF" >

<mx:TitleWindow alpha="1.0" layout="absolute" close="close()" title="Scribbler
Settings" showCloseButton="true"  width="200" height="178"  cornerRadius="16">
	<s:Button id="myButton" x="10" y="10" width="95"
click="toggleDrawing()"/>
	<mx:ColorPicker x="148" y="10" id="myCP" change="changeColor()"/>
	<mx:HSlider x="10" y="57" id="mySlider" change="changeThickness()"
minimum="1" maximum="10" snapInterval="1" enabled="true"
allowTrackClick="true"/>
	<mx:Label  id="myLabel" x="83" y="111" text="Label" width="87"/>
	<mx:Label x="20" y="111" text="Color:"/>
	<mx:Label x="26.5" y="76" text="Adjust Line Thickness"/>
</mx:TitleWindow>
</mx:Canvas>
```

6. Set up a new Script block and make the following declarations:

```
import flash.events.MouseEvent;

private var lineWidth:Number = 1;
private var lineColor:uint = 0x000000;
private var prevX:Number;
private var prevY:Number;
```

7. Within the Script block, add the following five functions

```
        private function init():void
        {
        }

        private function drawingOff():void
        {
        }

        private function drawingOn(event:MouseEvent):void
        {
        }

        private function toggleDrawing():void
        {
        }

        private function changeColor():void
        {
           lineColor = myCP.selectedColor;
           myLabel.text = String(myCP.value);
        }

        private function changeThickness():void
        {
           lineWidth = mySlider.value;
        }
```

8. We will first do the init function. This has three lines of code – one to register an event handler, the second to make the application full screen interactive; the third to change the button.label to reflect the functionality.

```
    private function init():void
        {
           stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
           myCanvas.addEventListener(MouseEvent.MOUSE_MOVE, drawingOn);
           myButton.label = "Drawing Off";
        }
```

9. Next, complete the drawingOn function. This function is called by init() and takes an Event as a parameter. We have to use the graphics workhorse package to create the line using the lineTo() method. The final line sets the myButton label to “Drawing off” to allow users to toggle the drawing on or off. Note that this only controls the button label and does not actually control what the button does.

```
    private function drawingOn(event:MouseEvent):void
        {
          if(!isNaN(prevX))
                 {
                        myCanvas.graphics.lineStyle(lineWidth, lineColor);
                        myCanvas.graphics.moveTo(prevX, prevY);
                        myCanvas.graphics.lineTo(event.localX, event.localY);
                 }
                 prevX = event.localX;
                 prevY = event.localY;
                 myButton.label = "Drawing Off";
        }
```

10. The drawingOff() function can be built next. This function removes an event Listener as changes the button label to "Drawing on" to prompt users to toggle it back on.

```
        private function drawingOff():void
        {
          myCanvas.removeEventListener(MouseEvent.MOUSE_MOVE, drawingOn);
          myButton.label = "Drawing On"
        }
```

11. toggleDrawing is the function that will actually control the drawing directly. toggleDrawing() detects the button label and then decides to call either drawingOff() or init().

```
     private function toggleDrawing():void
        {
          if (myButton.label == "Drawing Off")
          {
                 drawingOff();
          } else if (myButton.label == "Drawing On")
          {
                 init();
          }
        }
```

12. Run your code.

Project 14: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                         xmlns:s="library://ns.adobe.com/flex/spark"
                         xmlns:mx="library://ns.adobe.com/flex/halo"
                         applicationComplete="init()"  showFlexChrome="false">


<fx:Script>
      <![CDATA[
          import flash.events.MouseEvent;

          private var lineWidth:Number = 1;
          private var lineColor:uint = 0x000000;
          private var prevX:Number;
          private var prevY:Number;

          private function init():void
          {
             stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
             myCanvas.addEventListener(MouseEvent.MOUSE_MOVE, drawingOn);
          }

          private function drawingOff():void
          {
             myCanvas.removeEventListener(MouseEvent.MOUSE_MOVE, drawingOn);
             myButton.label = "Drawing On"
          }

          private function drawingOn(event:MouseEvent):void
          {
             if(!isNaN(prevX))
                   {
                         myCanvas.graphics.lineStyle(lineWidth, lineColor);
                         myCanvas.graphics.moveTo(prevX, prevY);
                         myCanvas.graphics.lineTo(event.localX, event.localY);
                   }
                   prevX = event.localX;
                   prevY = event.localY;
                   myButton.label = "Drawing Off";
          }

          private function toggleDrawing():void
          {
             if (myButton.label == "Drawing Off")
             {
                   drawingOff();
                   myButton.label = "Drawing On";
             } else if (myButton.label == "Drawing On")
             {
                   init();
```

```
                    myButton.label = "Drawing Off";
                }
            }

            private function changeColor():void
            {
                lineColor = myCP.selectedColor;
                myLabel.text = String(myCP.value);
            }

            private function changeThickness():void
            {
                lineWidth = mySlider.value;
            }
        ]]>
</fx:Script>

<mx:Canvas id="myCanvas" width="100%" height="100%" fontWeight="bold"
backgroundAlpha="0.00001"  backgroundColor="0xFF44FF" >

<mx:TitleWindow alpha="1.0" layout="absolute" close="close()" title="Scribbler
Settings" showCloseButton="true"  width="200" height="178"  cornerRadius="16">
        <s:Button id="myButton" x="10" y="10" width="95"
click="toggleDrawing()"/>
        <mx:ColorPicker x="148" y="10" id="myCP" change="changeColor()"/>
        <mx:HSlider x="10" y="57" id="mySlider" change="changeThickness()"
minimum="1" maximum="10" snapInterval="1" enabled="true"
allowTrackClick="true"/>
        <mx:Label  id="myLabel" x="83" y="111" text="Label" width="87"/>
        <mx:Label x="20" y="111" text="Color:"/>
        <mx:Label x="26.5" y="76" text="Adjust Line Thickness"/>
</mx:TitleWindow>
</mx:Canvas>

</mx:WindowedApplication>
```

Project 14: Test Questions

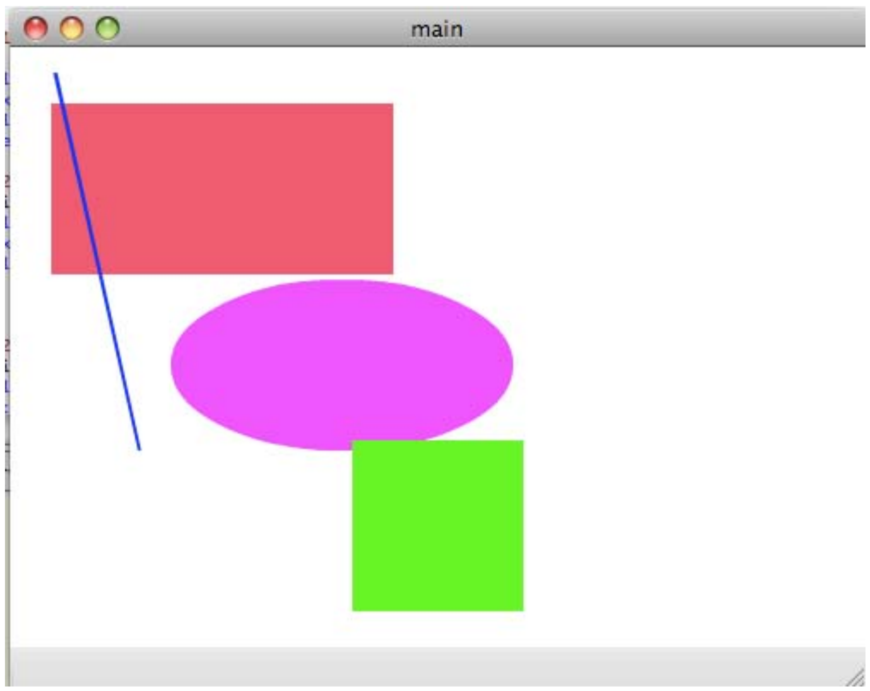
Q: What does the following line of code test?

```
if(!isNaN(prevX))
```

Q: What method do you call on mx.core.container to remove a listener from the EventDispatcher object and what minimal parameters must be declared?

Q: What does the attribute "allowClickTrack" control on the slider?

Project 15: Spark Graphics



The Spark Graphic control displays a set of graphic drawing commands.

You add a series of element tags such as <Rect>, <Path>, and <Ellipse> to the Graphic's elements Array to define the contents of the graphic.

Graphic controls do not have backgrounds or borders and cannot take focus.

When placed in a container, a Graphic is positioned by the rules of the container. However, the graphics in the Graphic control are always sized and positioned relative to the upper-left corner of the Graphics control.

The Graphic element can optionally contain a <Group> element.

1. Start a new project called ABC15-SparkGraphics
2. Set the layout to vertical.

3. Add a Canvas component and anchor it to the top, bottom, left and right by zero pixels.

4. Inside the Canvas, add a Graphic and declare the x and y to be zero.

5. Within the Graphic declaration, add a Spark Rectangle.

6. Add the following code to complete the shape.

```
		<s:Graphic x="0" y="0" >
			<s:Rect height="100" width="200" x="24" y="33">
				<s:fill>
					<mx:SolidColor color="0xFF3456"/>
				</s:fill>
			</s:Rect>
		</s:Graphic>
```

7. Run the project and see what happens. Now return to the source code and add a line, ellipse and square. Tilt the line from the vertical axis using the rotationX property setter.
8.

Project 15: Solution Code

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


	<s:layout>
		<s:VerticalLayout />
	</s:layout>

	<mx:Canvas top="0" bottom="0" left="0" right="0"
		     backgroundColor="0xFFFFFF">
		<s:Graphic x="0" y="0" >
```

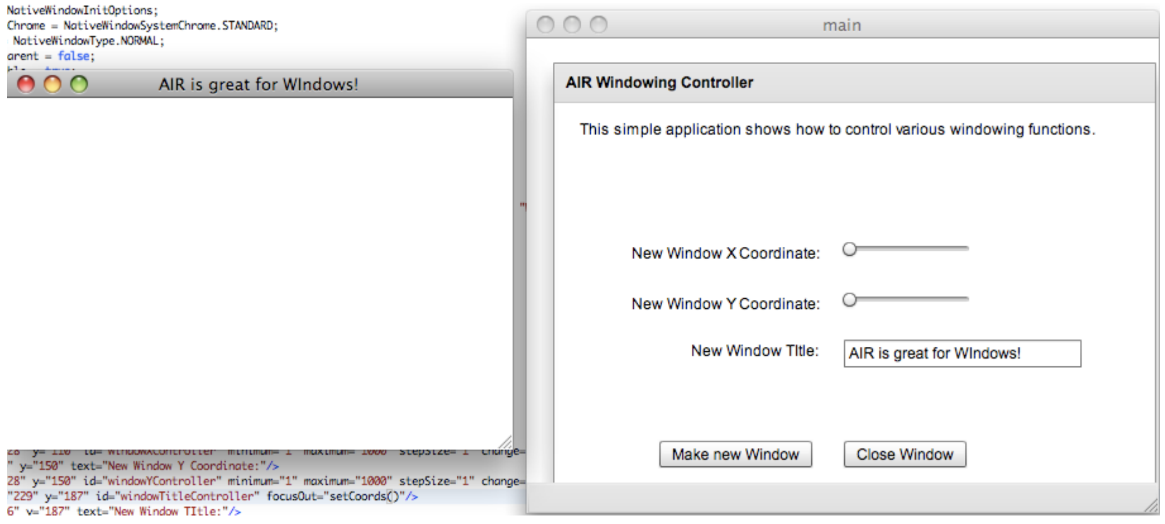
```
                    <s:Rect height="100" width="200" x="24" y="33">
                          <s:fill>
                                <mx:SolidColor color="0xFF3456"/>
                          </s:fill>
                    </s:Rect>
              </s:Graphic>

              <s:Graphic x="110" y="150"  >
                    <s:Ellipse height="100" width="200" x="-16" y="-14">
                          <s:fill>
                                <mx:SolidColor color="0xFF00FF"/>
                          </s:fill>
                    </s:Ellipse>
              </s:Graphic>

              <s:Graphic x="200" y="230" >
                    <s:Rect height="100" width="100">
                          <s:fill>
                                <mx:SolidColor color="0x00FF00"/>
                          </s:fill>
                    </s:Rect>
              </s:Graphic>

              <s:Graphic x="25" y="15">
                    <s:Rect height="270"  width="2" rotationX="30">
                          <s:fill>
                                <s:SolidColor color="0x0000FF" />
                          </s:fill>
                    </s:Rect>
              </s:Graphic>
        </mx:Canvas>


</s:WindowedApplication>
```

Project 15: Test Questions

Q: Why is there no circle and square and line (only Ellipse and Rect)?

Q: Should you use <mx:Canvas> still in Flex 4?

Project 16: Windowing API’s



```
NativeWindowInitOptions;
Chrome = NativeWindowSystemChrome.STANDARD;
 NativeWindowType.NORMAL;
arent = false;
```

AIR is great for WIndows!

main

AIR Windowing Controller

This simple application shows how to control various windowing functions.

New Window X Coordinate:

New Window Y Coordinate:

New Window TItle: AIR is great for WIndows!

```
" y="150" text="New Window Y Coordinate:"/>
28" y="150" id="windowYController" minimum="1" maximum="1000" stepSize="1" change=
"229" y="187" id="windowTitleController" focusOut="setCoords()"/>
6" y="187" text="New Window TItle:"/>
```

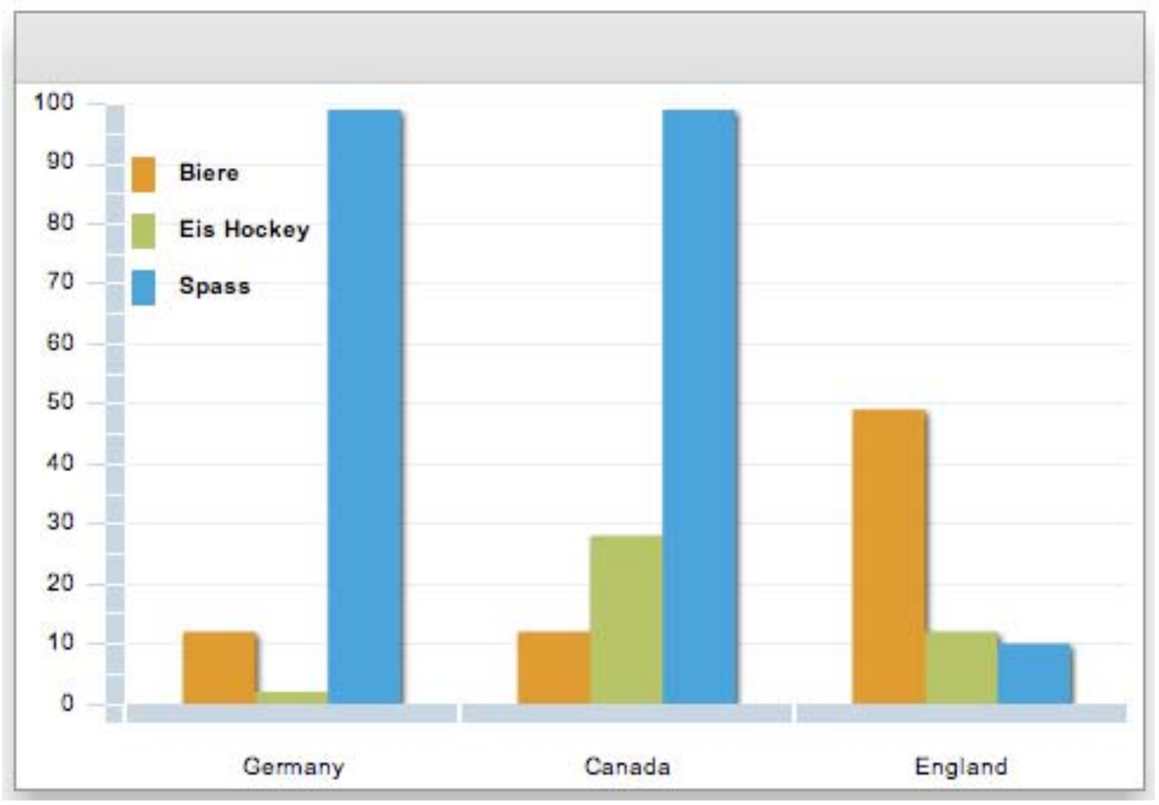
Project 16: Solution Code

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

        <fx:Script>
                <![CDATA[
                        import mx.controls.Alert;
                        import mx.core.*;

                        private var windowX:Number = 100;
                        private var windowY:Number = 100;
                        private var windowTitle:String;

                        private var myWindow:NativeWindow;
                        private var myNO:NativeWindowInitOptions;

                        private function createWindow():void
                        {
                        makeNativeOptions();
                        // Construct here to optimize application creation.
                        myWindow = new NativeWindow(myNO);
                        myWindow.height = 300;
                        myWindow.width = 400;
```

```
                    myWindow.x = windowX;
                    myWindow.y = windowY;
                    myWindow.title = windowTitleController.text;
                    myWindow.activate();
                    }

                    // The NativeWindow Constructor takes arguments of Window
initialization options.  Build here
                    // then pass to constructor. THe options are pretty bloody
obvious. ;-p
                    private function makeNativeOptions():void
                    {
                    myNO = new NativeWindowInitOptions;
                    myNO.systemChrome = NativeWindowSystemChrome.STANDARD;
            myNO.type = NativeWindowType.NORMAL;
            myNO.transparent = false;
            myNO.resizable = true;
            myNO.maximizable = true;
            myNO.minimizable = true;
                    }
                    private function closeWindows():void
                    {
                          myWindow.close();
                          if (myWindow.closed)
                          {
                                mx.controls.Alert.show("Hey man - your
window is like totally closed or something", "Window Closure
Notice.");
                          }
                    }
                    private function setCoords():void
                    {
                          windowX = windowXController.value;
                          windowY = windowYController.value;
                          windowTitle = windowTitleController.text;
                    }
             ]]>
      </fx:Script>

      <s:Panel x="21" y="19" width="477" height="407" title="AIR Windowing
Controller">
             <s:Button x="83" y="267" label="Make new Window"
click="createWindow()"/>
             <s:Button x="229" y="267" label="Close Window"
click="closeWindows()"/>
             <mx:Label x="59" y="110" text="New Window X Coordinate:"/>
             <s:HSlider x="228" y="110" id="windowXController" minimum="1"
maximum="1000" stepSize="1" change="setCoords()"/>
             <mx:Label x="59" y="150" text="New Window Y Coordinate:"/>
             <s:HSlider x="228" y="150" id="windowYController" minimum="1"
maximum="1000" stepSize="1" change="setCoords()"/>
             <s:TextInput x="229" y="187" id="windowTitleController"
text="Default Window Title"/>
```

```
            <mx:Label x="106" y="187" text="New Window TItle:"/>
            <mx:Label x="18" y="12" text="This simple application shows how to
control various windowing functions." height="69" width="449"/>
      </s:Panel>

</s:WindowedApplication>
```

Project 17: Charting



1. Start a new project called ABC17-Charting.
2. Create an <fx:Script> block and add the following code

```
<fx:Script><![CDATA[
                    import mx.collections.ArrayCollection;

                    [Bindable]
                    private var medalsAC:ArrayCollection = new
                    ArrayCollection([
                    {Country: "Germany", Beer:12, Hockey: 2, Fun: 99},
                    {Country: "Canada", Beer:12, Hockey: 28, Fun: 99},
                    {Country: "England", Beer:49, Hockey: 12, Fun: 10},
                        ]);
                        ]]>
                </fx:Script>
```

3. Add a Panel to your application and anchor it to the four edges of the application with ten pixels of padding.

```
</mx:Panel top="10" bottom="10" left="10" right="10"
                              layout="absolute">
</mx:Panel>
```

4. Inside of the Panel, add a column chart component. Set the dataProvider to match the ArrayCollection identifier and the showDataTips to true. For layout, ensure you set the right and left padding values to 5 pixels and the height and width to 100%.

```
<mx:ColumnChart id="column" height="100%" width="100%" paddingLeft="5"
             paddingRight="5" showDataTips="true" dataProvider="{medalsAC}">

</mx:ColumnChart>
```

5. We now need to set an axis for the chart. **mx.charts.chartClasses.CartesianChart** defines the labels, tick marks, and data position for items on the x-axis. This will allow a developer to use either the LinearAxis class or the CategoryAxis class to set the properties of the horizontalAxis as a child tag in MXML or create a LinearAxis or CategoryAxis object in ActionScript. For our project, we will add a Category Axis and set the value to "Country". Note this is case sensentive. Place this code inside the ColumnChart declaration.

```
       <mx:horizontalAxis>
              <mx:CategoryAxis categoryField="Country"/>
       </mx:horizontalAxis>
```

6. The ColumnChart now needs to know which series it will graph. For each Country laid out on the horizontal axis, bar charts can be drawn for various data available in the ArrayCollection. This project will demonstrate using all

the vailable data. Add the following code to your project right below the horizontal axis declaration and within the ColumnChart block

```
<mx:series>
  <mx:ColumnSeries xField="Country" yField="Beer" displayName="Biere" />
  <mx:ColumnSeries xField="Country" yField="Hockey" displayName="Eis Hockey" />
  <mx:ColumnSeries xField="Country" yField="Fun" displayName="Spass" />
</mx:series>
```

7. Finally, a legend should be included in a chart to explain to others what the chart means. The Legend control adds a legend to your charts, where the legend displays the label for each data series in the chart and a key showing the chart element for the series.   You can initialize a Legend control by binding a chart control identifier to the Legend control's dataProvider property (in this case medalsAC), or you can define an Array of LegendItem objects.

```
      <mx:Legend dataProvider="{column}" x="400" y="25"/>
```

8. Go ahead and run your project

Project 17: Solution Code

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

<fx:Script>
<![CDATA[
      import mx.collections.ArrayCollection;

      [Bindable]
      private var medalsAC:ArrayCollection = new
      ArrayCollection([
      {Country: "Germany", Beer:12, Hockey: 2, Fun: 99},
      {Country: "Canada", Beer:12, Hockey: 28, Fun: 99},
      {Country: "England", Beer:49, Hockey: 12, Fun: 10},
      ]);
]]>
</fx:Script>

<mx:Panel top="10" bottom="10" left="10" right="10" layout="absolute">
```

```
<mx:ColumnChart id="column" height="100%" width="100%" paddingLeft="5"
                paddingRight="5" showDataTips="true" dataProvider="{medalsAC}">

       <mx:horizontalAxis>
             <mx:CategoryAxis categoryField="Country"/>
       </mx:horizontalAxis>

<mx:series>
<mx:ColumnSeries xField="Country" yField="Beer" displayName="Biere" />
<mx:ColumnSeries xField="Country" yField="Hockey" displayName="Eis Hockey" />
<mx:ColumnSeries xField="Country" yField="Fun" displayName="Spass" />
</mx:series>

</mx:ColumnChart>
<mx:Legend dataProvider="{column}" x="44" y="25"/>

</mx:Panel>

</s:WindowedApplication>
```

Project 17: Test Questions

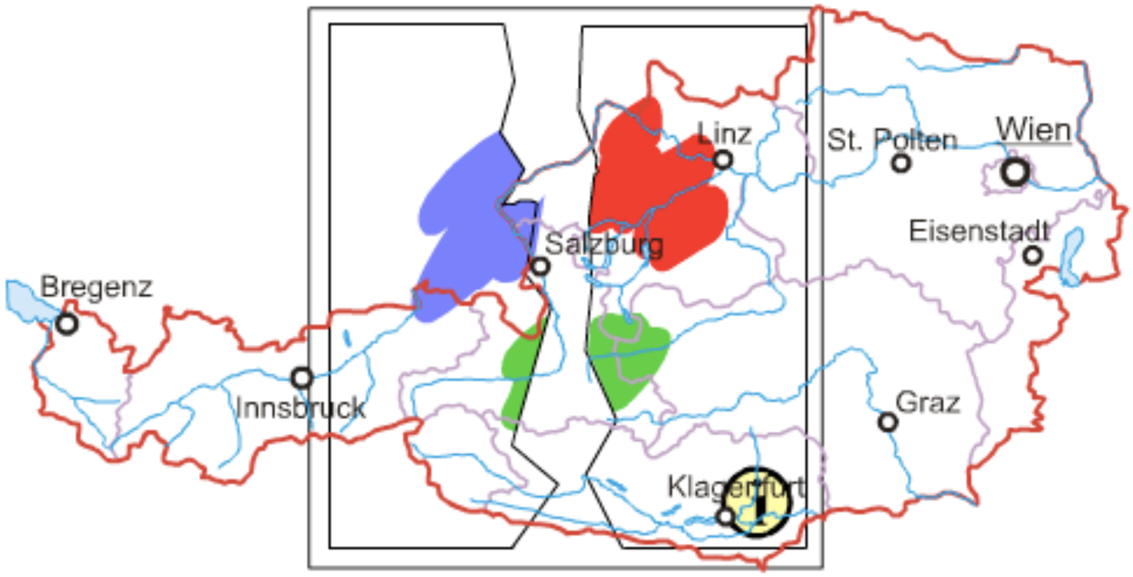
Q: What is the ArrayCollection and how does it differ from Array?

Q: How can I add another countries data to the Chart?

Q: What is the relationship between the Series and the CategoryAxis?

Q: Are the names of the ArrayCollection case sensitive?

Project 18: SVG Rendering



1. Import the File ABC18-SVG form the ~/attendeeProjects folder.

2. Open and inspect the SVG file.

3. Open the main.mxml file. We will learn to embed the SVG file into the project. To do this, use the following syntax within a Script block:

```
[Embed(source="assets/map.svg")]
```

4. Create a variable for holding the SVG data for the application and make it Bindable.

```
<fx:Script> <![CDATA[
      [Embed(source="assets/map.svg")]

      [Bindable]
      public var SvgMap:Class;
]]></fx:Script>
```

5. Add an Image component and bind the source to the SVG variable

```
<mx:Image  id="large" source="{SvgMap}" bottom="10" top="10" right="10"
          left="10"  x="10" themeColor="#ffffff" y="10"/>
```

6. Run your project

Project 18: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark"

xmlns:mx="library://ns.adobe.com/flex/halo">

      <fx:Script>
            <![CDATA[
                  [Embed(source="assets/map.svg")]

                  [Bindable]
                  public var SvgMap:Class;
            ]]>
      </fx:Script>

      <mx:Image  id="large" source="{SvgMap}" bottom="10" top="10"
          right="10" left="10"  x="10" themeColor="#ffffff" y="10"/>

</s:WindowedApplication>
```

Project 18: Test Questions

Q: What does the Embed tag do in this instance?

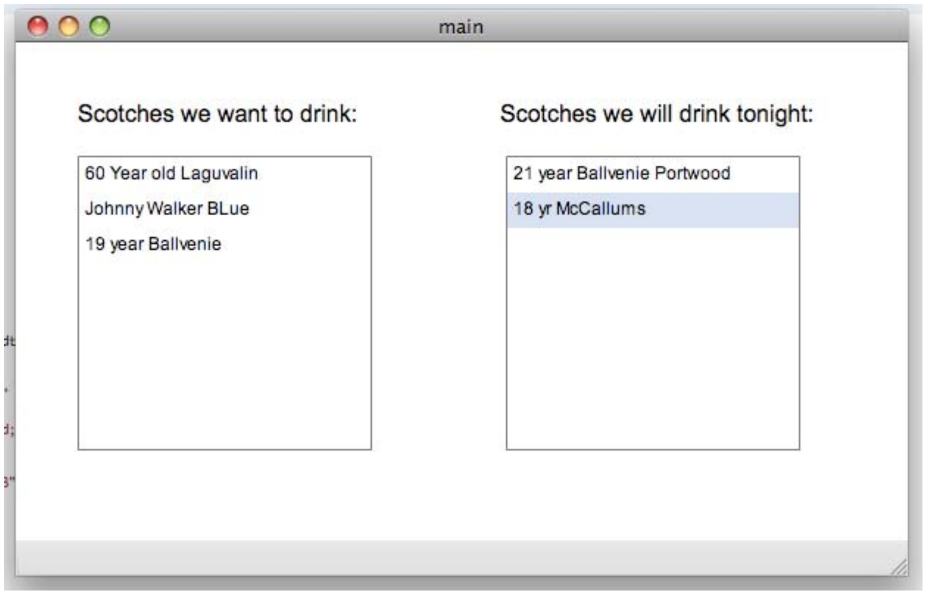
Q: What does the following code do and why is it a good idea for vector graphics?

```
bottom="10" top="10" right="10" left="10"
```

Q: when the vector graphic is bound to the top, right, bottom and left, will it scale out of proportion?

Q: What code in this project is redundant?

Project 19: Drag and Drop



60 Year old Laguvalin
Johnny Walker BLue
19 year Ballvenie

21 year Ballvenie Portwood
18 yr McCallums

1. Create a new project called ABC19-DragDrop.

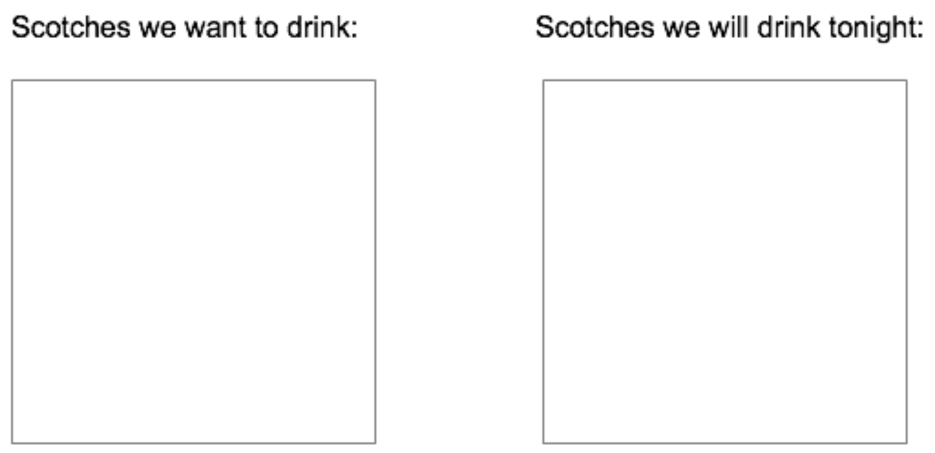
2. Add an Array declaration within the <fx:Declarations> element with an ID of "Scotch".

3. Add a second Array declaration below the first with the ID of "Tasted".

4. Within the first Array, add an array of Strings (at least 6) and add the names of some Scotch. Good single malt scotch is preferred but some Blends like Johnny Walker Blue are acceptable too.

```
<fx:Declarations>

      <fx:Array id="Scotch">
            <fx:String>60 Year old Laguvalin</fx:String>
            <fx:String>21 year Ballvenie Portwood</fx:String>
            <fx:String>Johnny Walker BLue</fx:String>
            <fx:String>18 yr McCallums</fx:String>
            <fx:String>19 year Ballvenie</fx:String>
      </fx:Array>

      <fx:Array id="Tasted" />

</fx:Declarations>
```

5. Add two labels and two lists as shown below



6. Give the first one an dataProvider of “Scotch” and the second a dataProvider of “Tasted”.
7. Set the following attributes for each of the Lists

```
         dragEnabled="true" dragMoveEnabled="true"
        dropEnabled="true" allowMultipleSelection="true"
```

8. Run your project and try dragging and dropping values from one list to the other.

Project 19: Solution Code

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

xmlns:mx="library://ns.adobe.com/flex/halo">

<fx:Declarations>

      <fx:Array id="Scotch">
            <fx:String>60 Year old Laguvalin</fx:String>
            <fx:String>21 year Ballvenie Portwood</fx:String>
            <fx:String>Johnny Walker BLue</fx:String>
            <fx:String>18 yr McCallums</fx:String>
            <fx:String>19 year Ballvenie</fx:String>
      </fx:Array>

      <fx:Array id="Tasted" />
```

```
</fx:Declarations>

<s:Label x="42" y="42" text="Scotches we want to drink:&#xd;" width="200"
        height="27" fontSize="16"/>

<mx:List  dataProvider="{Scotch}" height="200" width="200"
         dragEnabled="true" dragMoveEnabled="true"
         dropEnabled="true" allowMultipleSelection="true" x="42" y="77"/>

<s:Label x="329" y="42" text="Scotches we will drink tonight:&#xd;" width="221"
        height="27" fontSize="16"/>

<mx:List  dataProvider="{Tasted}" height="200" width="200"
        dragEnabled="true" dragMoveEnabled="true"
        dropEnabled="true" allowMultipleSelection="true" x="333" y="77"/>

</s:WindowedApplication>
```

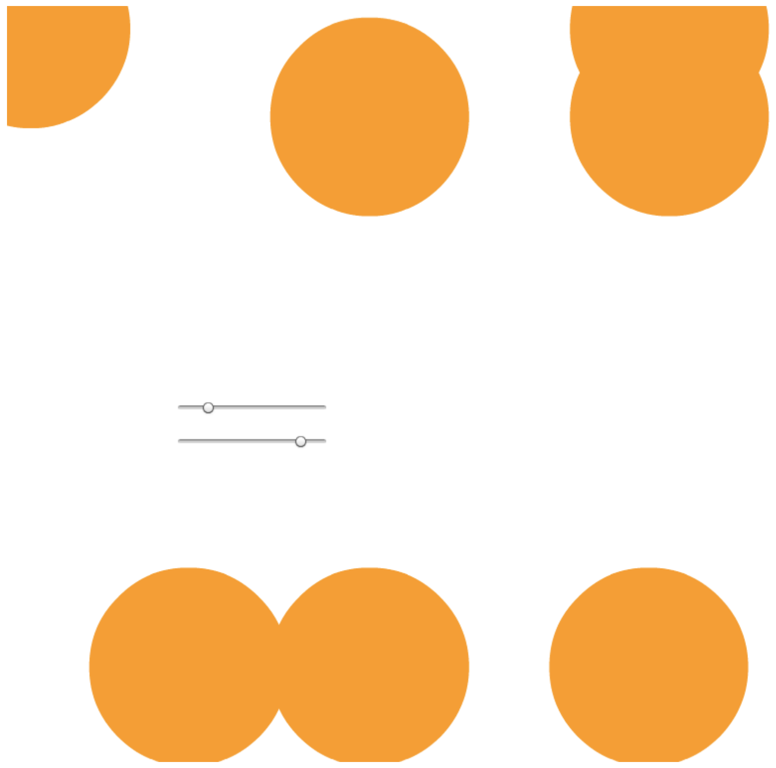
Project 19: Test Questions

Q: In the code above, what are the funny character at the end of these lines and why are they there?

```
text="Scotches we want to drink:&#xd;"
text="Scotches we will drink tonight:&#xd;"
```

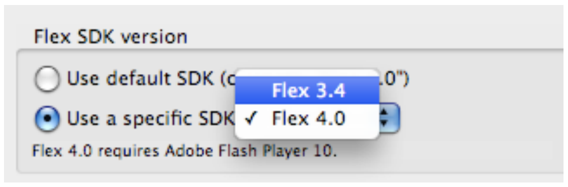
Q: Could you use an ArrayCollection in this project instead of an Array?

Project 20: Circles



This project will show how to use other Flex SDK's within Flash Builder 4 as well as how to control the creation of graphics using ActionScript.

1. Create new project in Flash Builder 4. Name the project ABC20-Circles. Check that the project is a browser based project and on the main dialog box, deselect the Flex 4 SDK and select the Flex 3.4 SDK as shown below.



Flex SDK version

Use default SDK (c .0")

Use a specific SDK: Flex 3.4 / ✓ Flex 4.0

Flex 4.0 requires Adobe Flash Player 10.

2. Create a Script block and add two variables of type number named xPosition and yPosition. Set their initial values to 200.

```
        private var xPosition:Number = 200;
        private var yPosition:Number = 200;
```

3. Add two horizontal Sliders to your project and lay them out so they each are located at the bottom center of your application. One must have the ID of yPosition and the second xPosition.

4. Add a button just below the sliders. Change the button label to "Make Circle" and add a click event handler to call a function (not yet written) called "makeRoundThing()"

5. Add the function makeRoundThing() to your code accepting no parameters and returning void.

6. Create a member variable called circle of type "Shape".

7. Add the following code to the function.

```
                    private function makeRoundThing():void
                    {
                          var circle:Shape = new Shape();
                          xPosition = xControl.value;
                          yPosition = yControl.value;
                          var xPos:Number = xPosition;
                          var yPos:Number = yPosition;
                          var radius:Number = 100;

                    }
```

8. Last, we have to add three more lines of code to construct and add the circle to the caller.

```
              private function makeRoundThing():void
                    {
                          …
                          circle.graphics.beginFill(0xFF8800);
                          circle.graphics.drawCircle(xPos, yPos, radius);
                          this.rawChildren.addChild(circle);
```

9. Run your project.

Project 20: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                          layout="absolute">
      <mx:Script>
```

```
			<![CDATA[
				private var xPosition:Number = 200;
				private var yPosition:Number = 200;

				private function makeRoundThing():void
				{
					var circle:Shape = new Shape();
					xPosition = xControl.value;
					yPosition = yControl.value;
					var xPos:Number = xPosition;
					var yPos:Number = yPosition;
					var radius:Number = 100;

					circle.graphics.beginFill(0xFF8800);
					circle.graphics.drawCircle(xPos, yPos, radius);
					this.rawChildren.addChild(circle);

				}
			]]>
		</mx:Script>

		<mx:HSlider  x="168" y="366" id="yControl" minimum="25"
		                   maximum="800" enabled="true"/>
		<mx:HSlider x="168" y="325" id="xControl" minimum="25"
		                  maximum="800" enabled="true"/>
		<mx:Button x="215" y="386" label="Make Circle" click="makeRoundThing()"/>
</mx:Application>
```

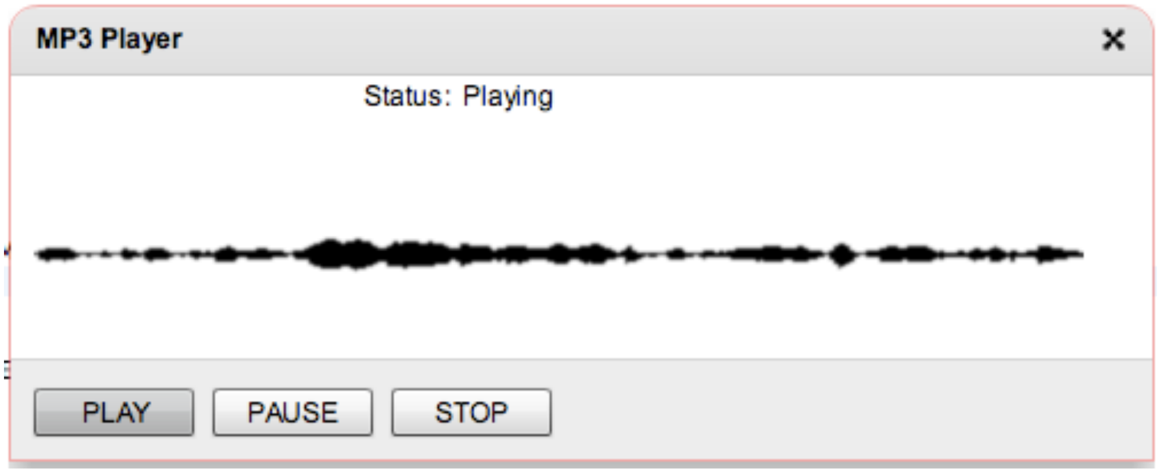
Project 20: Test Questions

1. Why is the [Bindable] keyword not necessary for the variable values?

2. What modifications would have to be made to this to render faster as a Web Application?

Q: What does the keyword “this” represent in the code above?

Project 21: Working with Audio



This project will demonstrate working with Audio, including a custom library component in an AIR application, namespace qualification of components, soundChannel and play, pause and stop.

It combines several aspects of the previous lessons including building chromeless apps, events, opening local assets and how to bundle (using [embed]) audio files vs streaming.

1. Open up the project in your ~/attendeeProjects folder called ABC21-MP3Player.

2. Note the structure of the folders ~/com/fusiox/ui/Visualization.as

3. Add it to the project by adding this line to the root element:

```
xmlns:fx="com.fusiox.ui.*"
```

4. Add it into the project (inside the mx:TitleWindow) by adding the following line to the source code (in source view):

```
<fx:Visualization id="vis" width="460" height="70"  x="10" y="43"/>
```

5. In order to control sound, you need a way to track where you are in the soundAsset. Declare a variable as shown (find the comment and add the line under it)

```
// TODO: add the number variable here called myPos
 private var myPos:Number = 0;
```

6. Load up the media asset by locating the comment and adding the three lines below it.

```
<!--TODO: add theURLRequest, Sound and SoundChannel-->
  <net:URLRequest id="myURLReq" url="assets/22ndCentury_509.mp3" />
  <media:SoundChannel id="mySoundChannel" />
  <media:Sound id="mySound" />
```

7. Now wire up the URLRequest by adding the second function call triggered from the creationComplete event.

```
creationComplete="init(), mySound.load(myURLReq)"
```

8. Inside the control bar, add the following buttons:

```
<!-- Add  three buttons-->
<mx:Button id="play" label="PLAY" click="mySoundChannel = mySound.play(myPos,0,null);
prog.text='Playing'" />

<mx:Button id="pause" label="PAUSE" click="myPos = mySoundChannel.position;
mySoundChannel.stop(); prog.text='Paused'" />

<mx:Button id="stop" label="STOP"   click="mySoundChannel.stop(); myPos=0;
prog.text='Stopped'" />
```

9. Run the damn program and rock out to the fine tunes from 22nd Century! (http://www.22ndcenturyofficial.com)

Project 21: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                    xmlns:s="library://ns.adobe.com/flex/spark"
                    xmlns:mx="library://ns.adobe.com/flex/halo"
                    showFlexChrome="false" xmlns:fu="com.fusiox.ui.*"
                    xmlns:net="flash.net.*" xmlns:media="flash.media.*"
                    layout="absolute" creationComplete="init(),
                    mySound.load(myURLReq)" >

<fx:Script><![CDATA[
      import mx.events.CloseEvent;
      import flash.display.Bitmap;

      private var myPos:Number = 0;

      private function init():void
      {
      // register listeners
      mainPanel.addEventListener(MouseEvent.MOUSE_DOWN, startMove);
      }

      // This makes the close button work.
      public function closeEvent(event:CloseEvent):void
      {
      stage.nativeWindow.close();
      }

      // Move the app when the panelis dragged
      public function startMove(event:MouseEvent):void
      {
      nativeWindow.startMove();
      }

]]></fx:Script>

<fx:Declarations>
      <net:URLRequest id="myURLReq" url="assets/22ndCentury_509.mp3" />
      <media:SoundChannel id="mySoundChannel" />
      <media:Sound id="mySound" />
</fx:Declarations>

<mx:TitleWindow mouseMove="mainPanel.addEventListener(MouseEvent.MOUSE_DOWN,
startMove)" id="mainPanel" title="MP3 Player" showCloseButton="true"
layout="absolute" close="closeEvent(event)" borderColor="#F02323"
themeColor="#FE0E02" cornerRadius="12" alpha="1.0" width="500" height="199">

      <mx:Text id="prog"  width="125" x="195"/>
      <mx:Label text="Status:" x="152"/>

      <fu:Visualization id="vis" width="460" height="70"  x="10" y="43"/>
      <mx:ControlBar>
      <mx:Button id="play" label="PLAY"  click="mySoundChannel =
                  mySound.play(myPos,0,null); prog.text='Playing'"/>
      <mx:Button id="pause" label="PAUSE"  click="myPos =
      mySoundChannel.position; mySoundChannel.stop(); prog.text='Stopped'" />
      <mx:Button id="stop" label="STOP"
         click="mySoundChannel.stop(); myPos=0; prog.text='Stopped'" />
      </mx:ControlBar>

</mx:TitleWindow>
</mx:WindowedApplication>
```

Project 21: Test Questions

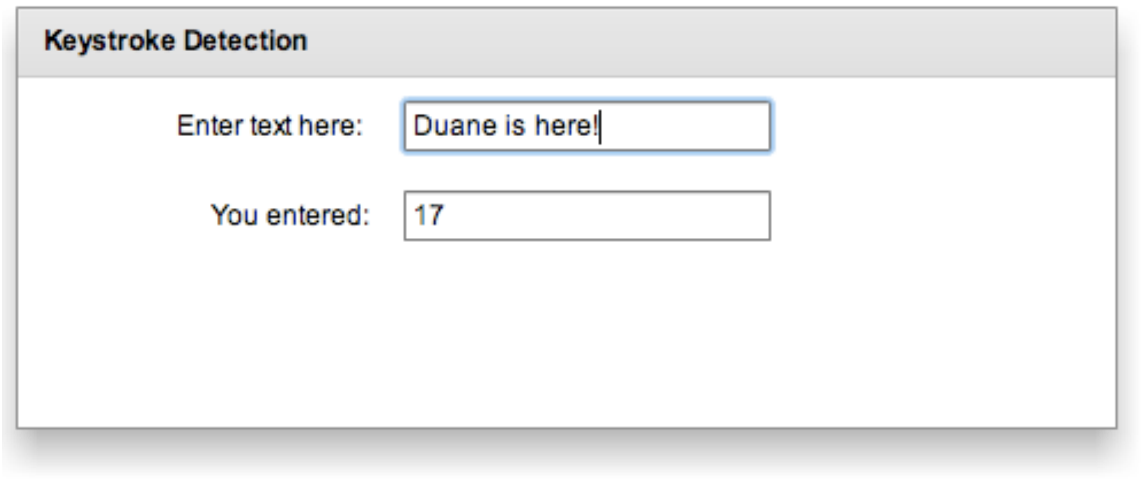
Q: What will the statement xmlns:fu="com.fusiox.ui.*" do?

Q: Why must the imports be namespace qualified?

Q: Why is it possible in this project to use the `showFlexChrome="false"` **syntax in the root element of this application?**

Q: What is the difference between a SoundChannel and a Sound?

Project 22: Keyboard Events



In some cases, you want to trap keys globally, meaning no matter where the user is in the application, their keystrokes are recognized by the application and the action is performed. Flex recognizes global keyboard events whether the user is hovering over a button or the focus is inside a TextInput control.

A common way to handle global key presses is to create a listener for the KeyboardEvent.KEY_DOWN or KeyboardEvent.KEY_UP event on the application. Listeners on the application container are triggered every time a key is pressed, regardless of where the focus is (as long as the focus is in the application on not in the browser controls or outside of the browser). Inside the handler, you can examine the key code or the character code using the charCode and keyCode properties of the KeyboardEvent.

1. Start a new project and call it ABC22-KeyEvents.

2. Add two textInputs and two Text objects to your project so that it looks roughly the same as the graphic at the start of this project.

3. Give one of the textInputs an ID of “keyCode” and set the value of the text beside it to “You Entered: ”

4. Start a Script block and declare a public variable of type String called keyString. This variable will hold the keyCode value when detected by an event listener.

```
public var keyString:String;
```

5. Set up a function called "init()" and call it from the creationComplete event.

6. Using the "this" keyword, add an event listener to trap the KEY_DOWN event (it is a KeyboardEvent) and register trapkeys() as the event handler.

```
            private function init():void
            {
                    this.addEventListener(KeyboardEvent.KEY_DOWN, trapKeys);
            }
```

7. Write the trapKeys function to accept the event and set the value of the text component to the key value.

```
            private function trapKeys(e:KeyboardEvent):void
            {
                    keyCode.text = String(e.keyCode);
            }
```

8. Run your program.

Project 22: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                       xmlns:s="library://ns.adobe.com/flex/spark"
                       xmlns:mx="library://ns.adobe.com/flex/halo"
                       minWidth="1024" minHeight="768"
                       creationComplete="init()">

<fx:Script><![CDATA[
            import flash.events.KeyboardEvent;

            public var keyString:String;
```

```
			private function init():void
			{
				this.addEventListener(KeyboardEvent.KEY_DOWN, trapKeys);
			}

			private function trapKeys(e:KeyboardEvent):void
			{
				keyCode.text = String(e.keyCode);
			}
]]></fx:Script>

<mx:Panel x="10" y="10" width="480" height="184" layout="absolute"
		title="Keystroke Detection">
			<mx:Text x="67" y="12" text="Enter text here:"/>
			<mx:TextInput x="168" y="10" />
			<mx:TextInput id="keyCode" x="168" y="49" />
			<mx:Text x="82" y="51" text="You entered:" textAlign="right"/>
	</mx:Panel>
</s:Application>
```

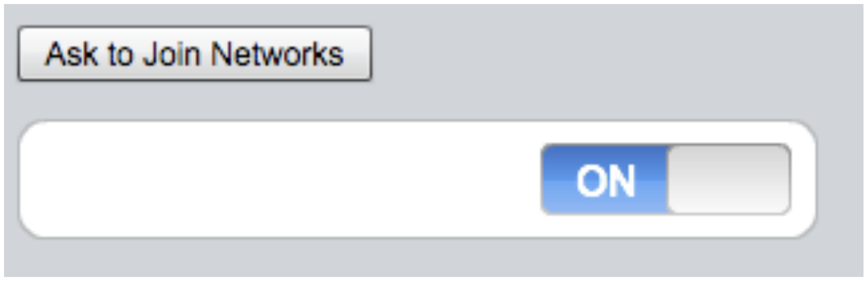
Project 22: Test Questions

Q: IN the following syntax, what does the "e" represent and why is it named "e"?

Q: What does the error say and why?

Q: Can it be named something else?

Project 23: Skinning Spark Components



Project 23: Solution Code

Main.mxml

```
<?xml version="1.0" encoding="utf-8"?>
      <s:Application
            pageTitle="Phone Toggle Button"
            width="100%"
            height="100%"
            backgroundColor="0xC6CBD1"
            xmlns:fx="http://ns.adobe.com/mxml/2009"
            xmlns:s="library://ns.adobe.com/flex/spark"
            xmlns:mx="library://ns.adobe.com/flex/halo">

            <s:ToggleButton x="10" y="10" label="Ask to Join Networks"/>

            <s:ToggleButton
                  x="10"
                  y="45"
                  label="Ask to Join Networks"
                  skinClass="skins.PhoneToggleSkin"/>

</s:Application>
```

PhoneToggleSkin.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Skin
      width="300"
      height="45"
      xmlns:fx="http://ns.adobe.com/mxml/2009"
      xmlns:s="library://ns.adobe.com/flex/spark"
      xmlns:mx="library://ns.adobe.com/flex/halo">
```

```
	<!-- Metadata (host component) -->
	<fx:Metadata>
		[HostComponent( "spark.components.ToggleButton" )]
	</fx:Metadata>

	<!-- States -->
	<s:states>
		<mx:State name="up"/>
		<mx:State name="over"/>
		<mx:State name="down"/>
		<mx:State name="upAndSelected"/>
		<mx:State name="overAndSelected"/>
		<mx:State name="downAndSelected"/>
		<mx:State name="disabled"/>
		<mx:State name="disabledAndSelected"/>
	</s:states>

	<!-- Transition to play at selection state change -->
	<s:transitions>
		<mx:Transition>
			<s:Move
				target="{slider}"
				duration="250"/>
		</mx:Transition>
	</s:transitions>

	<!-- Control background area -->
	<s:Rect bottom="0" left="0" right="0" top="0" radiusX="10" radiusY="10">
		<s:fill>
			<mx:SolidColor color="0xA7ABAC"/>
		</s:fill>
	</s:Rect>

	<s:Rect bottom="1" left="1" right="1" top="1" radiusX="10" radiusY="10">
		<s:fill>
			<mx:SolidColor color="0xFFFFFF"/>
		</s:fill>
	</s:Rect>

	<!-- Label -->
	<!-- Picks up on label element that toggle expects -->
	<mx:Text id="labelElement" fontSize="16" color="0x000000"
fontWeight="bold" verticalCenter="2" left="10"/>

	<!-- Toggle portion -->
	<s:Group id="toggle" width="94" height="27" right="10"
verticalCenter="0">

	<!-- Outer border -->
	<s:Rect bottom="0" left="0" right="0" top="0" radiusX="4" radiusY="4">
	  <s:fill>
		<mx:LinearGradient rotation="90">
```

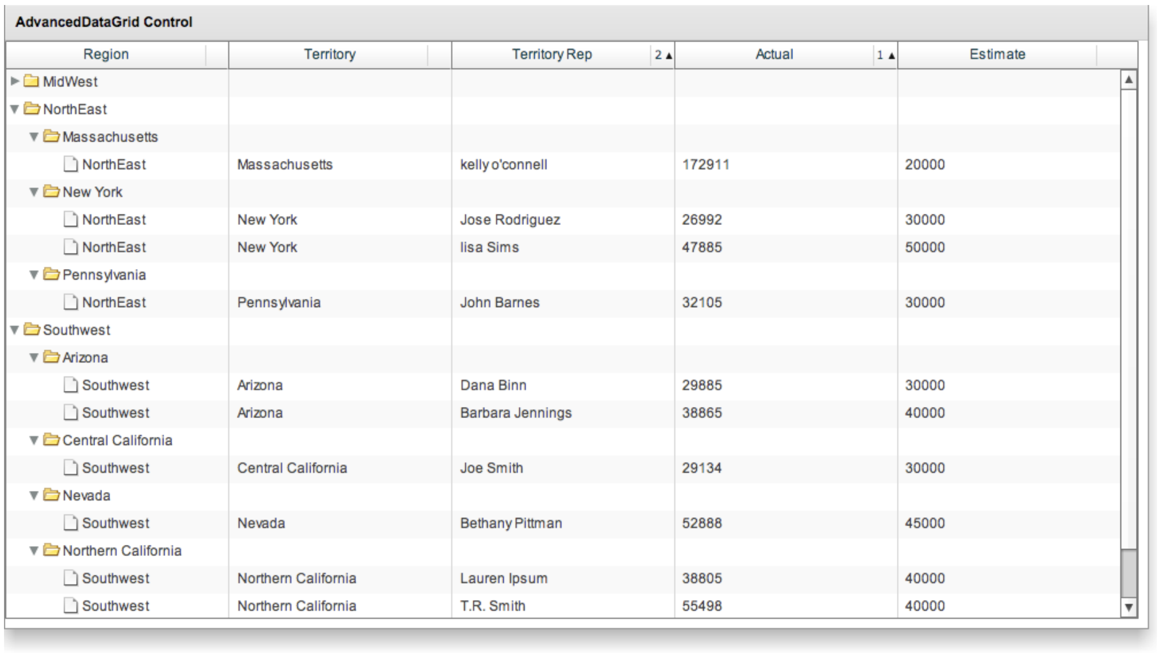
```
                  <mx:entries>
                     <mx:GradientEntry color="0x7D7D7D" ratio="0"/>
                     <mx:GradientEntry color="0xC0C0C0" ratio="1"/>
                  </mx:entries>
               </mx:LinearGradient>
            </s:fill>
         </s:Rect>

         <!-- Slider element -->
         <s:Group id="slider" x.up="{0 - ( toggle.width / 2 )}" x.over="{0 - (
toggle.width / 2 )}" x.overAndSelected="0" x.upAndSelected="0"
width="{toggle.width * 1.5}" height="100%" mask="{slidermask}">

         <!-- Blue (on) side -->
         <s:Rect width="{slider.width / 2}" height="100%">
            <s:fill>
               <mx:LinearGradient rotation="90">
                  <mx:entries>
                     <mx:GradientEntry color="0x2E59B7" ratio="0"/>
                     <mx:GradientEntry color="0x4185EA" ratio="0.5"/>
                     <mx:GradientEntry color="0x5294F2" ratio="0.5"/>
                     <mx:GradientEntry color="0x7DACF4" ratio="1"/>
                  </mx:entries>
               </mx:LinearGradient>
            </s:fill>
         </s:Rect>

         <!-- Gray (off) side -->
         <s:Rect width="{slider.width / 2}" height="100%" right="0">
            <s:fill>
               <mx:LinearGradient rotation="90">
                  <mx:entries>
                     <mx:GradientEntry color="0xE7E7E7" ratio="0"/>
                      <mx:GradientEntry color="0xEEEEEEE" ratio="0.5"/>
                      <mx:GradientEntry color="0xFAFAFA" ratio="0.5"/>
                      <mx:GradientEntry color="0xFEFEFE" ratio="1"/>
                  </mx:entries>
               </mx:LinearGradient>
            </s:fill>
         </s:Rect>

         <!-- Labels -->
         <mx:Text text="ON" left="11" verticalCenter="2" fontSize="16"
color="0xFFFFFF" fontWeight="bold"/>
         <mx:Text text="OFF" right="8" verticalCenter="2" fontSize="16"
color="0x7D7D7D" fontWeight="bold"/>

         <!-- Button -->
         <s:Group id="button" x="{( slider.width - button.width ) / 2}"
width="{toggle.width / 2}" height="100%">

         <!-- Outer button border -->
         <s:Rect width="100%" height="100%" radiusX="4" radiusY="4">
```

```
          <s:fill>
              <mx:LinearGradient rotation="90">
                 <mx:entries>
                      <mx:GradientEntry color="0x7D7D7D" ratio="0"/>
                      <mx:GradientEntry color="0xC0C0C0" ratio="1"/>
                 </mx:entries>
              </mx:LinearGradient>
          </s:fill>
      </s:Rect>

      <!-- Inner button fill -->
      <s:Rect x="1" y="1" width="{button.width - 2}" height="{button.height -
2}" radiusX="4" radiusY="4">
          <s:fill>
              <mx:LinearGradient rotation="90">
                 <mx:entries>
                     <mx:GradientEntry color="0xCCCCCC" ratio="0"/>
                     <mx:GradientEntry color="0xFBFBFB" ratio="1"/>
                 </mx:entries>
              </mx:LinearGradient>
          </s:fill>
      </s:Rect>

     </s:Group>

   </s:Group>

      <!-- Mask for the slider -->
      <s:Group id="slidermask" bottom="1" left="1" right="1" top="1">

      <s:Rect width="100%" height="100%" radiusX="4" radiusY="4">
              <s:fill>
                     <mx:SolidColor color="0x00FF00"/>
              </s:fill>
      </s:Rect>

      </s:Group>

   </s:Group>

</s:Skin>
```

Project 23: Test Questions

Q: How many states does a Spark Button have?

Project 24: Advanced Data Grid

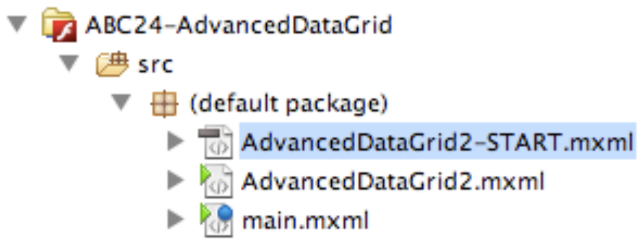


AdvancedDataGrid Control

Region	Territory	Territory Rep 2 ▲	Actual 1 ▲	Estimate
▶ MidWest				
▼ NorthEast				
▼ Massachusetts				
NorthEast	Massachusetts	kelly o'connell	172911	20000
▼ New York				
NorthEast	New York	Jose Rodriguez	26992	30000
NorthEast	New York	lisa Sims	47885	50000
▼ Pennsylvania				
NorthEast	Pennsylvania	John Barnes	32105	30000
▼ Southwest				
▼ Arizona				
Southwest	Arizona	Dana Binn	29885	30000
Southwest	Arizona	Barbara Jennings	38865	40000
▼ Central California				
Southwest	Central California	Joe Smith	29134	30000
▼ Nevada				
Southwest	Nevada	Bethany Pittman	52888	45000
▼ Northern California				
Southwest	Northern California	Lauren Ipsum	38805	40000
Southwest	Northern California	T.R. Smith	55498	40000

1. Import the project in the attendeeProjects folder named ABC24-AdvancedDataGrid into your workspace.

2. Open up the file AdvancedDataGrod2-START.mxml.



```
▼ ABC24-AdvancedDataGrid
    ▼ src
        ▼ (default package)
            ▶ AdvancedDataGrid2-START.mxml
            ▶ AdvancedDataGrid2.mxml
            ▶ main.mxml
```

3. In source code view you can see a large arrayCollection of data. The task at hand is to display that data on an advanced data grid. Inside the Panel declaration, add an advanced data grid and give it an ID of myADG as shown below.

```
<s:Panel title="AdvancedDataGrid Control"  width="996" height="544">
<mx:AdvancedDataGrid  id="myADG" color="0x323232" width="986" height="503">


</mx:AdvancedDataGrid>
</s:Panel>
```

4. Rather than supply a simple dataProvider attribute, we will manually bind different parts of the dataset to the datagrid. Within the Advanced Data grid component declaration, add the dataprovider element and then add a GroupingCollection and set the id to gc and the source to be bound to dpFlat (the name of the ArrayCollection) as shown below.

```
<mx:dataProvider>
   <mx:GroupingCollection id="gc" source="{dpFlat}">

    </mx:GroupingCollection>
</mx:dataProvider>
```

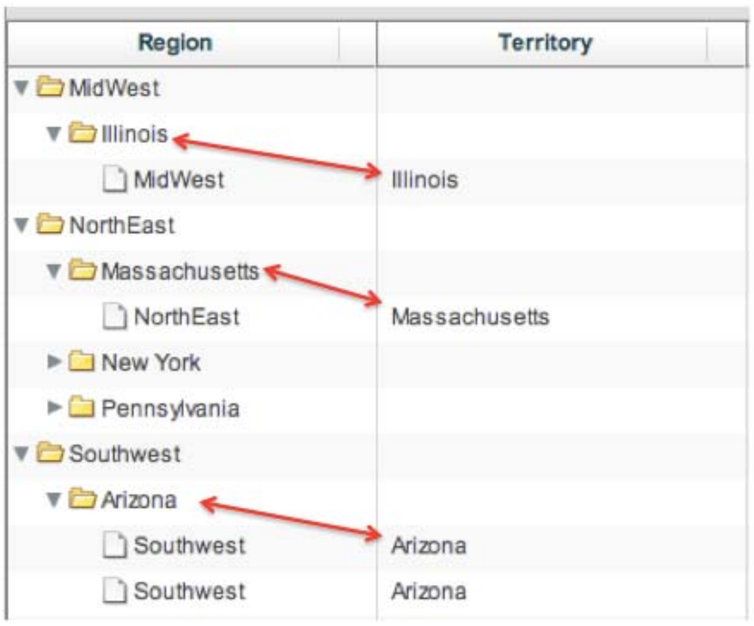
5. Within the GroupingCollection declaration, add the following code

```
<mx:dataProvider>
   <mx:GroupingCollection id="gc" source="{dpFlat}">
       <mx:grouping>
          <mx:Grouping>
              <mx:GroupingField name="Region"/>
              <mx:GroupingField name="Territory"/>
          </mx:Grouping>
       </mx:grouping>
  </mx:GroupingCollection>
</mx:dataProvider>
```

<mx:grouping> Specifies the Grouping instance applied to the source data. Setting the grouping property does not automatically refresh the view, so you must call the refresh() method after setting this property.

<mx:Grouping> **mx.collections.Grouping** The Grouping class defines the fields in the data provider of the AdvancedDataGrid control used to group data. You use this class to create groups when the input data to the AdvancedDataGrid control has a flat structure.

<mx:GroupingField> declares the flat data fields that will be grouped. In this case, Region and Territory will be grouped which creates a hierarchy in the first column.



Region	Territory
▼ MidWest	
▼ Illinois	
MidWest	Illinois
▼ NorthEast	
▼ Massachusetts	
NorthEast	Massachusetts
▶ New York	
▶ Pennsylvania	
▼ Southwest	
▼ Arizona	
Southwest	Arizona
Southwest	Arizona

6. Now add the columns to the advanced data grid that you wish to display.

```
<mx:columns>
        <mx:AdvancedDataGridColumn dataField="Region"/>
        <mx:AdvancedDataGridColumn dataField="Territory"/>
        <mx:AdvancedDataGridColumn dataField="Territory_Rep"
                                  headerText="Territory Rep"/>
        <mx:AdvancedDataGridColumn dataField="Actual"/>
        <mx:AdvancedDataGridColumn dataField="Estimate"/>
</mx:columns>
```

7. The IGroupingCollection does not detect changes to a group automatically, so you must call the refresh() method to update the view after setting the group property. The refresh method must be called manually based on the initialize event for the advanced data grid. Add the following attribute

```
initialize="gc.refresh();"
```

8. Run your project. To test out the grouping functions, try changing the values of the grouped fields and run it again to see what happens like the code shown below.

```
        <mx:Grouping>
                <mx:GroupingField name="Region"/>
                <mx:GroupingField name="Actual"/>
        </mx:Grouping>
```

Project 24: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                             xmlns:s="library://ns.adobe.com/flex/spark"
                             xmlns:mx="library://ns.adobe.com/flex/halo"
                             minWidth="1024" minHeight="768">
        <fx:Script>
                <![CDATA[
                import mx.collections.ArrayCollection;

                [Bindable]
                private var dpFlat:ArrayCollection = new ArrayCollection([
                {Region:"Southwest", Territory:"Arizona",
                Territory_Rep:"Barbara Jennings", Actual:38865, Estimate:40000},
                {Region:"Southwest", Territory:"Arizona",
                Territory_Rep:"Dana Binn", Actual:29885, Estimate:30000},
                {Region:"Southwest", Territory:"Central California",
                Territory_Rep:"Joe Smith", Actual:29134, Estimate:30000},
                {Region:"Southwest", Territory:"Nevada",
                Territory_Rep:"Bethany Pittman", Actual:52888, Estimate:45000},
                {Region:"Southwest", Territory:"Northern California",
                Territory_Rep:"Lauren Ipsum", Actual:38805, Estimate:40000},
                {Region:"Southwest", Territory:"Northern California",
                Territory_Rep:"T.R. Smith", Actual:55498, Estimate:40000},
                {Region:"Southwest", Territory:"Southern California",
                Territory_Rep:"Alice Treu", Actual:44985, Estimate:45000},
                {Region:"Southwest", Territory:"Southern California",
                Territory_Rep:"Jane Grove", Actual:44913, Estimate:45000},
                {Region:"NorthEast", Territory:"New York",
                Territory_Rep:"Jose Rodriguez", Actual:26992, Estimate:30000},
                {Region:"NorthEast", Territory:"New York",
                Territory_Rep:"lisa Sims", Actual:47885, Estimate:50000},
                {Region:"NorthEast", Territory:"Massachusetts",
                Territory_Rep:"kelly o'connell", Actual:172911, Estimate:20000},
                {Region:"NorthEast", Territory:"Pennsylvania",
                Territory_Rep:"John Barnes", Actual:32105, Estimate:30000},
                {Region:"MidWest", Territory:"Illinois",
                Territory_Rep:"Seth Brown", Actual:42511, Estimate:40000}
                ]);
```

```
             ]]>
      </fx:Script>
      <s:Panel title="AdvancedDataGrid Control"  width="996" height="544">

      <mx:AdvancedDataGrid id="myADG" color="0x323232"
              initialize="gc.refresh();" width="986" height="503">
         <mx:dataProvider>
             <mx:GroupingCollection id="gc" source="{dpFlat}">
                <mx:grouping>
                    <mx:Grouping>
                       <mx:GroupingField name="Region"/>
                       <mx:GroupingField name="Territory"/>
                    </mx:Grouping>
                </mx:grouping>
             </mx:GroupingCollection>
         </mx:dataProvider>

         <mx:columns>
             <mx:AdvancedDataGridColumn dataField="Region"/>
             <mx:AdvancedDataGridColumn dataField="Territory"/>
             <mx:AdvancedDataGridColumn dataField="Territory_Rep"
                                        headerText="Territory Rep"/>
             <mx:AdvancedDataGridColumn dataField="Actual"/>
             <mx:AdvancedDataGridColumn dataField="Estimate"/>
         </mx:columns>
      </mx:AdvancedDataGrid>
   </s:Panel>
</s:Application>
```

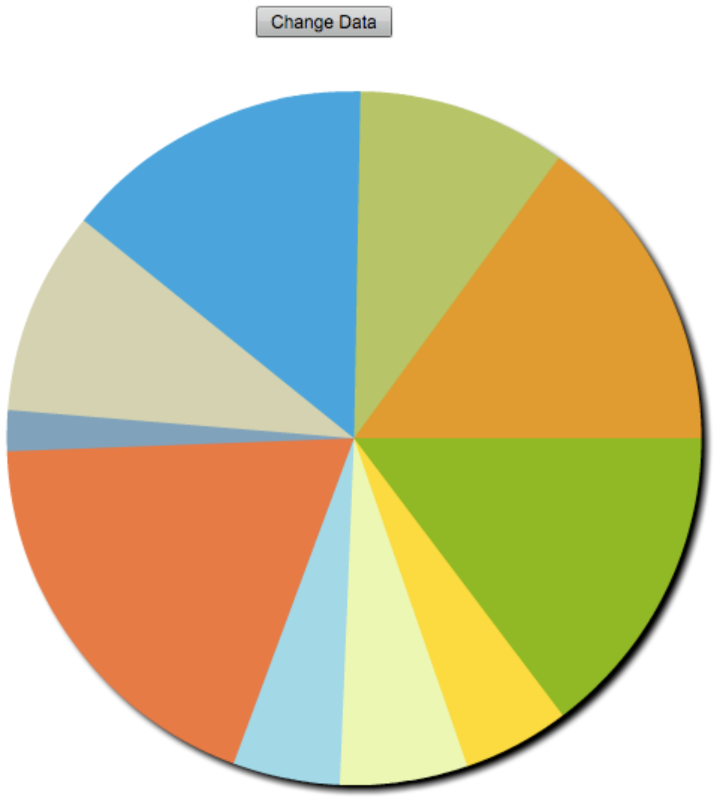
Project 24: Test Questions

Q: When should refresh() be called to update the view?

Q: Can you group more than 2 fields for hierarchic display in an advanced data grid?

Q: What features make the Advanced Data Grid class different than the regular data grid?

Project 25: Dynamic Pie Chart



This project will demonstrate a cool transition effect for rendering a pie chart as well as some advanced math functions in ActionScript.

1. Start a new project and call it ABC25-DynamicPieChart
2. In the Design view, add a PieChart and a Button. Change the button's text label to read "Change Data"
3. Switch to code view and give the button a click event handler that calls initDP() (not yet written).
4. Within the <Declarations> element, define an <fx:Array with an id of 'dp'.

```
    <fx:Declarations>
            <fx:Array id="dp" />
     </fx:Declarations>
```

5. Add a script block and define a function called initDP(): void.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                xmlns:s="library://ns.adobe.com/flex/spark"
                xmlns:mx="library://ns.adobe.com/flex/halo"
                 minWidth="1024" minHeight="768">
       <fx:Declarations>
              <fx:Array id="dp" />
       </fx:Declarations>
       <fx:Script>
              <![CDATA[
                     private function initDP():void
                     {

                     }
              ]]>
       </fx:Script>
       <mx:PieChart x="125" y="89" id="chart" width="333" height="328">
              <mx:series>
                     <mx:PieSeries displayName="Series 1" field=""/>
              </mx:series>
       </mx:PieChart>
       <mx:Legend dataProvider="{chart}"/>
       <s:Button x="252" y="30" click="initDP()" label="Button"/>
</s:Application>
```

6. Declare a private constant called MAX_ITEMS, typed as uint with an initialized value of 10.

```
        private const MAX_ITEMS:uint = 10;
```

7. Within the function define the following items
- a member variable called "i" type = uint.
- an untyped array called 'dp'. This will be the data provider for the pie chart

8. We will now iterate through an the array as shown in the code block below and create a random value and push (assign) it to the array.

```
       for (i = 0; i < MAX_ITEMS; i++)
        {
              dp.push({data:getRandomUint(100), label:"item " + i});
        }
```

9. At this point you will still get an error in your project since the getRandomUint() has not been defined. Define that function as shown below:

```
       private function getRandomUint(max:uint):uint
       {
              return Math.round(Math.random() * max);
        }
```

10. Create the function getRandomUint() as shown below.

```
private function getRandomUint(max:uint):uint
{
       return Math.round(Math.random() * max);
}
```

11. Set the dataProvider property of your Pie Chart to bind to ‘dp’.

12. Change the PieSeries declarations as shown below:

```
       <mx:PieSeries field="data">
           <mx:showDataEffect>
              <mx:SeriesInterpolate duration="1000" />
           </mx:showDataEffect>
       </mx:PieSeries>
```

13. Finally – trigger initDP() on the ApplicationComplete event and run your program.

Project 25: Solution Code

```

<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                       xmlns:s="library://ns.adobe.com/flex/spark"
                       xmlns:mx="library://ns.adobe.com/flex/halo"
                       applicationComplete="initDP()" >
<fx:Script><![CDATA[

private const MAX_ITEMS:uint = 10;
private function initDP():void
{
	var i:uint;
	dp = [];
	for (i = 0; i < MAX_ITEMS; i++)
	{
		dp.push({data:getRandomUint(100), label:"item " + i});
	}
}

private function getRandomUint(max:uint):uint
{
	return Math.round(Math.random() * max);
}
]]></fx:Script>

<fx:Declarations>
	<fx:Array id="dp" />
</fx:Declarations>

<s:Button label="Change Data" click="initDP();"  x="349" y="10"/>

<mx:PieChart id="chart" height="492" width="719"
		dataProvider="{dp}" x="55" y="53">
	<mx:series>
		<mx:PieSeries field="data">
			<mx:showDataEffect>
				<mx:SeriesInterpolate duration="1000" />
			</mx:showDataEffect>
		</mx:PieSeries>
	</mx:series>
</mx:PieChart>

</s:WindowedApplication>
```

Project 25: Test Questions

Q: What does the 1000 mean in the following code?

```
<mx:SeriesInterpolate duration="1000" />
```

Q: What does the const keyword do to a variable?

Q: Write some code to show the random number values in an Alert box.

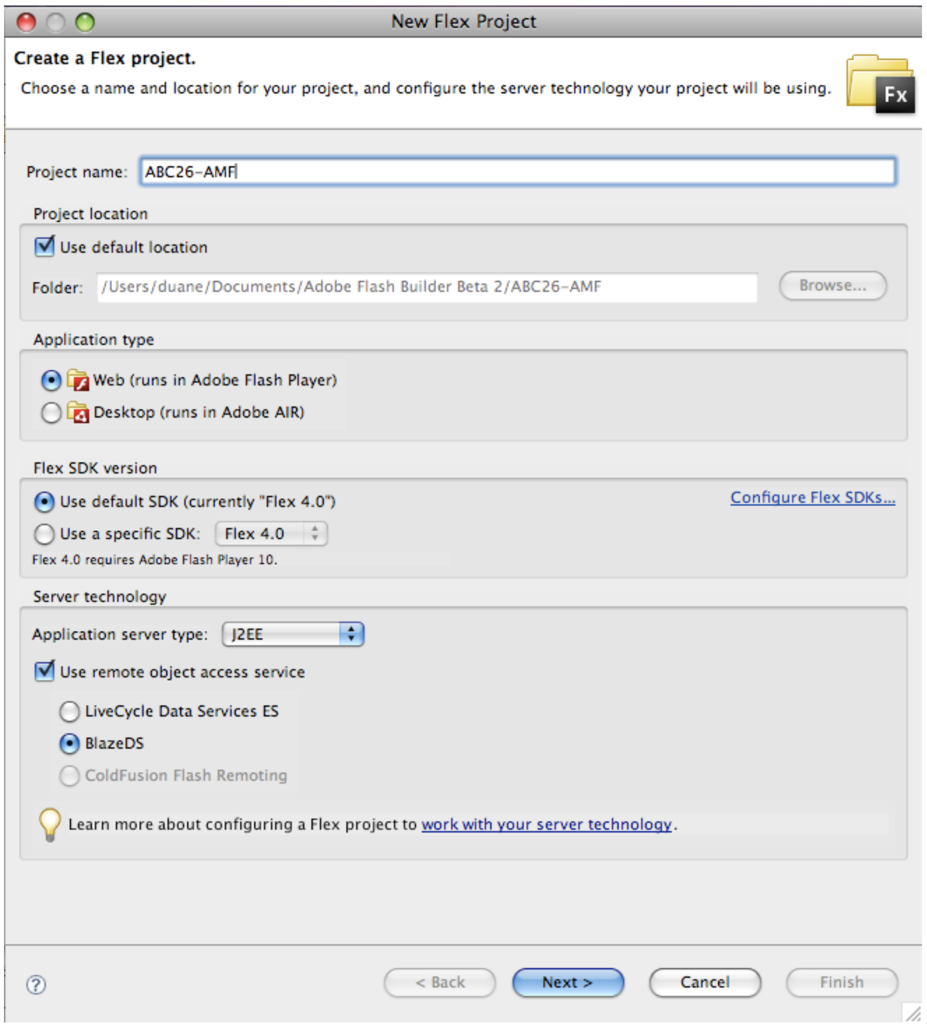
Project 26: AMF (Flex Remoting)

Note: Make sure BlazeDS is still up and running for this lab. This can be done by checking http://localhost:8400/samples/

1. Start a new project and name it MAX-BlazeDS. Make this a Flex application and select the box for server technologies as shown below. Call your application ABC26-AMF as shown below.

   ```
   Application Server Type -> J2EE
   Use Remote Object Access Service -> Checked
   LiveCycle Data Services -> Checked
   ```

   Click "Next"


New Flex Project

Create a Flex project.

Choose a name and location for your project, and configure the server technology your project will be using.

Project name: ABC26-AMF

Project location

[✓] Use default location

Folder: /Users/duane/Documents/Adobe Flash Builder Beta 2/ABC26-AMF

Application type

(●) Web (runs in Adobe Flash Player)

( ) Desktop (runs in Adobe AIR)

Flex SDK version

(●) Use default SDK (currently "Flex 4.0")

( ) Use a specific SDK: Flex 4.0

Flex 4.0 requires Adobe Flash Player 10.

Server technology

Application server type: J2EE

[✓] Use remote object access service

( ) LiveCycle Data Services ES

(●) BlazeDS

( ) ColdFusion Flash Remoting

Learn more about configuring a Flex project to work with your server technology.

2. As shown below, on the next screen you will need to validate your server configuration. You will need to fill in the proper values for Root Folder, Root URL and Context Root as shown. Use the “Browse” button to locate the

`<BlazeDS_Root>\BlazeDS\tomcat\webapps\samples`

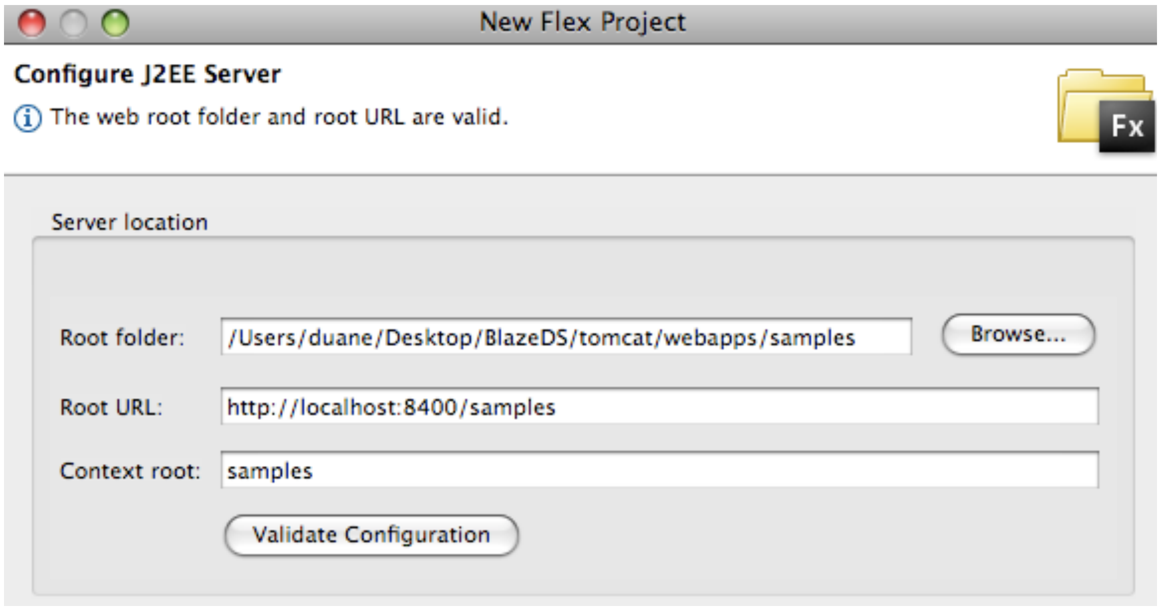
directory.

Set the other values as shown:

Root URL -> http://localhost:8400/samples
Context Root -> /samples

The output folder should be filled out by default.

NOTE: you must have permissions to write to this folder for the lab to work.


New Flex Project

Configure J2EE Server

The web root folder and root URL are valid.

Server location

Root folder: /Users/duane/Desktop/BlazeDS/tomcat/webapps/samples

Root URL: http://localhost:8400/samples

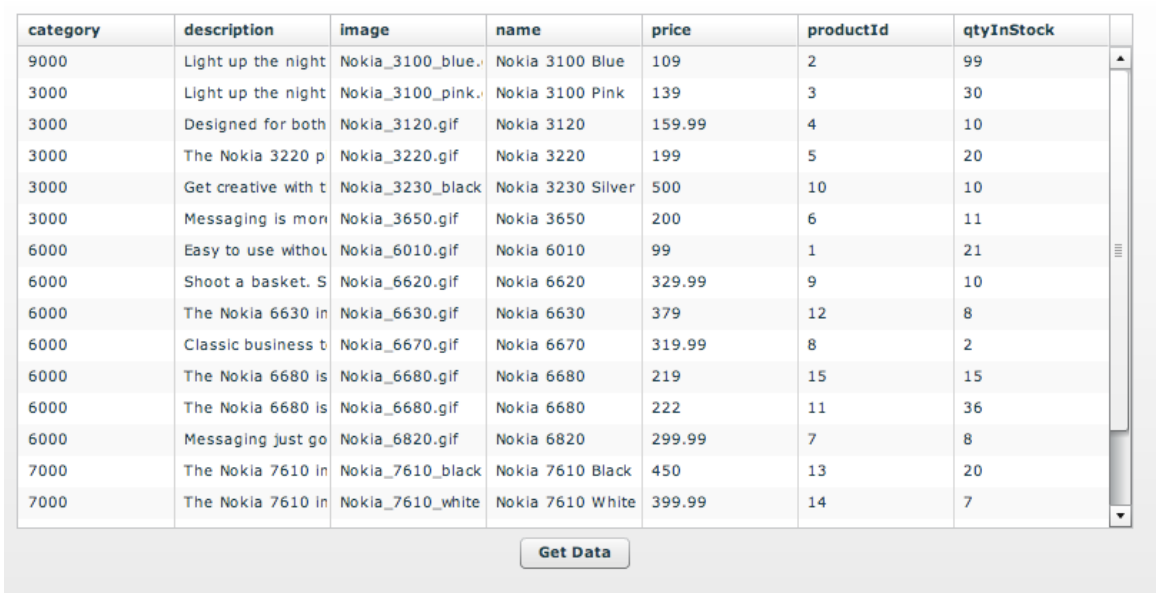
Context root: samples

3. Click Finished and you will have a new project done.

4. In your Flex project, enter the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                  xmlns:s="library://ns.adobe.com/flex/spark"
                  xmlns:mx="library://ns.adobe.com/flex/halo"
minWidth="1024" minHeight="768">
      <s:layout>
          <s:BasicLayout/>
      </s:layout>
      <fx:Declarations>
            <mx:RemoteObject id="srv" destination="product"/>
      </fx:Declarations>
      <mx:DataGrid dataProvider="{srv.getProducts.lastResult}"
width="836" height="362" y="6" x="5"/>
      <s:Button label="Get Data" click="srv.getProducts()"  x="349"
y="423"/>

</s:Application>
```

5. Click “run” and your project should be able to connect to the locahost:8400 remote object and return the screen as follows:



category	description	image	name	price	productId	qtyInStock
9000	Light up the night	Nokia_3100_blue.	Nokia 3100 Blue	109	2	99
3000	Light up the night	Nokia_3100_pink.	Nokia 3100 Pink	139	3	30
3000	Designed for both	Nokia_3120.gif	Nokia 3120	159.99	4	10
3000	The Nokia 3220 p	Nokia_3220.gif	Nokia 3220	199	5	20
3000	Get creative with t	Nokia_3230_black	Nokia 3230 Silver	500	10	10
3000	Messaging is mor	Nokia_3650.gif	Nokia 3650	200	6	11
6000	Easy to use withou	Nokia_6010.gif	Nokia 6010	99	1	21
6000	Shoot a basket. S	Nokia_6620.gif	Nokia 6620	329.99	9	10
6000	The Nokia 6630 in	Nokia_6630.gif	Nokia 6630	379	12	8
6000	Classic business t	Nokia_6670.gif	Nokia 6670	319.99	8	2
6000	The Nokia 6680 is	Nokia_6680.gif	Nokia 6680	219	15	15
6000	The Nokia 6680 is	Nokia_6680.gif	Nokia 6680	222	11	36
6000	Messaging just go	Nokia_6820.gif	Nokia 6820	299.99	7	8
7000	The Nokia 7610 in	Nokia_7610_black	Nokia 7610 Black	450	13	20
7000	The Nokia 7610 in	Nokia_7610_white	Nokia 7610 White	399.99	14	7

6. Now that we have done this, let's play with the server side code a bit. Navigate (use whatever software is typical for your OS) to the

   <BlazeDS_Root>/tomcat/webapps/samples/WEB-INF/src/flex/samples/products/

   directory and open up the file ProductService.java.

7. In the previous Flex sample, we wrote the following line of code:

```
<s:Button label="Get Data" click="srv.getProducts()"  x="349" y="423"/>
```

8. The srv.getProducts() corresponds to the Java method of the same name. These are case sensitive. Note line 13 of the java source says:

```
public List getProducts() throws DAOException {..}
```

9. If you have configured a Java SDK (JDK) on your system and have the JAVA_HOME environmental variable set properly, you will be able to modify the java code. First, in the file

   <BlazeDS_Root>/blazeds/tomcat/webapps/samples/WEB-INF/src/flex/samples/product/Product.java

   Modify the line of code that reads:

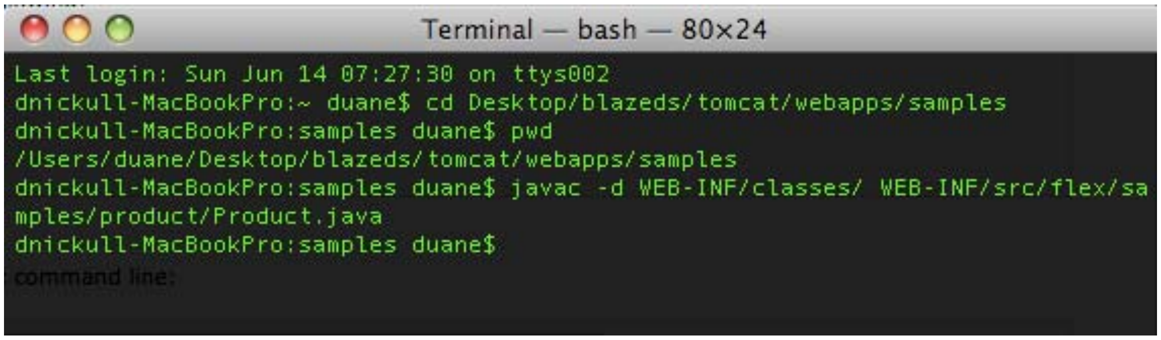
```
public String getDescription() {
   return description;
}
```

   So that it reads

```
public String getDescription() {
   return "BlazeDS Vancouver Build Rocks!";
}
```

   And save the file

10. Grab a terminal (Command Window) and change directory to **<BLazeDS_Root>/blazeds/tomcat/webapps/samples**



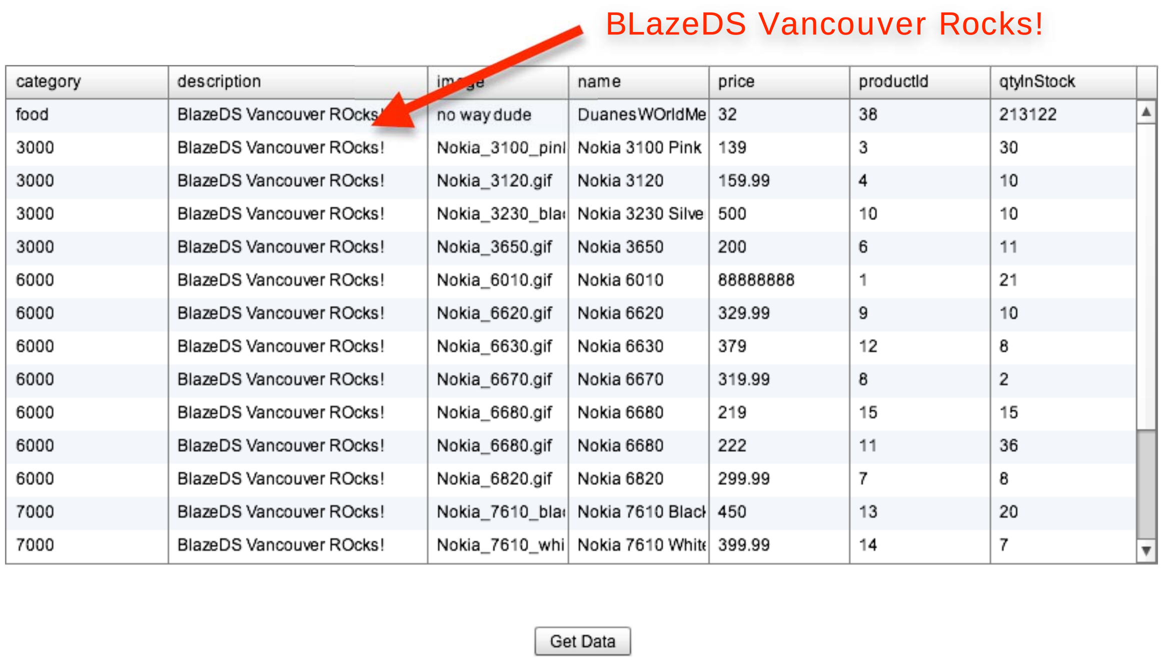
```
Last login: Sun Jun 14 07:27:30 on ttys002
dnickull-MacBookPro:~ duane$ cd Desktop/blazeds/tomcat/webapps/samples
dnickull-MacBookPro:samples duane$ pwd
/Users/duane/Desktop/blazeds/tomcat/webapps/samples
dnickull-MacBookPro:samples duane$ javac -d WEB-INF/classes/ WEB-INF/src/flex/sa
mples/product/Product.java
dnickull-MacBookPro:samples duane$
```

11. Recompile the Product.java class and place the resulting class file in the destination folder by typing in:

```
javac -d WEB-INF/classes/ WEB-
INF/src/flex/samples/product/Product.java
```

This command creates the file Product.class, and deploys it to the WEB-INF\classes\flex\samples\product directory.

Re-run the sample and you will see a different result in the column for products. Note it might take a few seconds for the server to pick up the new class.



category	description	image	name	price	productId	qtyInStock
food	BlazeDS Vancouver ROcks!	no way dude	DuanesWOrldMe	32	38	213122
3000	BlazeDS Vancouver ROcks!	Nokia_3100_pinl	Nokia 3100 Pink	139	3	30
3000	BlazeDS Vancouver ROcks!	Nokia_3120.gif	Nokia 3120	159.99	4	10
3000	BlazeDS Vancouver ROcks!	Nokia_3230_bla	Nokia 3230 Silve	500	10	10
3000	BlazeDS Vancouver ROcks!	Nokia_3650.gif	Nokia 3650	200	6	11
6000	BlazeDS Vancouver ROcks!	Nokia_6010.gif	Nokia 6010	88888888	1	21
6000	BlazeDS Vancouver ROcks!	Nokia_6620.gif	Nokia 6620	329.99	9	10
6000	BlazeDS Vancouver ROcks!	Nokia_6630.gif	Nokia 6630	379	12	8
6000	BlazeDS Vancouver ROcks!	Nokia_6670.gif	Nokia 6670	319.99	8	2
6000	BlazeDS Vancouver ROcks!	Nokia_6680.gif	Nokia 6680	219	15	15
6000	BlazeDS Vancouver ROcks!	Nokia_6680.gif	Nokia 6680	222	11	36
6000	BlazeDS Vancouver ROcks!	Nokia_6820.gif	Nokia 6820	299.99	7	8
7000	BlazeDS Vancouver ROcks!	Nokia_7610_bla	Nokia 7610 Black	450	13	20
7000	BlazeDS Vancouver ROcks!	Nokia_7610_whi	Nokia 7610 White	399.99	14	7

Project 26: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                  xmlns:s="library://ns.adobe.com/flex/spark"
                  xmlns:mx="library://ns.adobe.com/flex/halo"
minWidth="1024" minHeight="768">

      <fx:Declarations>
            <mx:RemoteObject id="srv" destination="product"/>
      </fx:Declarations>
      <mx:DataGrid dataProvider="{srv.getProducts.lastResult}"
width="836" height="362"/>
      <s:Button label="Get Data" click="srv.getProducts()"  x="384"
```

```
y="407"/>

</s:Application>
```

Project 26: Test Questions

Q: What is AMF?

Q: Given the following line of code, where does "product" come from or what does it reference?

```
        <fx:Declarations>
                <mx:RemoteObject id="srv" destination="product"/>
         </fx:Declarations>
```

Project 27: Data Paging

Flash Builder 4 lets you enable data paging with just few clicks for any server technology.It can be enabled for any service type (Including HTTP Service).

In this sample, we will enable data paging for a PHP class. For data paging to work you just need to implement 2 functions on the server and leave it to the generated Flex code by Flash Builder to invoke these functions when data has to be fetched, for example when user scrolls the DataGrid scroll bar.

Here is a snippet of the PHP code:

```
<?php


//The full file is at the end of this tutorial in Appendix A
```

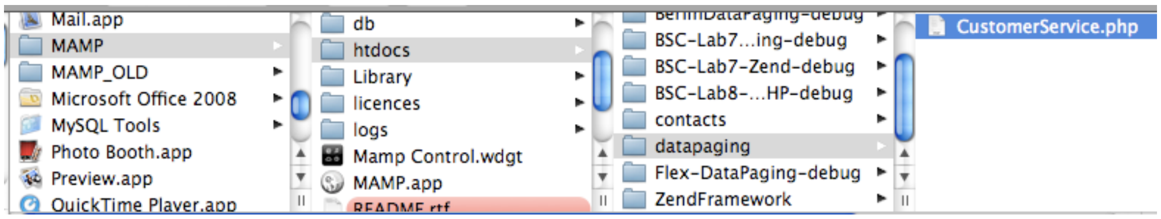
```
public function getItems_paged($startIndex, $numItems) {

                                $this->connect();
                                $startIndex = mysqli_real_escape_string($this->connection,
$startIndex);
                                $numItems = mysqli_real_escape_string($this->connection,
$numItems);
                                $sql = "SELECT * FROM customers LIMIT $startIndex,
$numItems";

                                $result = mysqli_query($this->connection, $sql) or die('Query
failed: ' . mysqli_error($this->connection));

                                $rows = array();
               while ($row = mysqli_fetch_object($result)) {
                               $rows[] = $row;
               }

               mysqli_free_result($result);
                                mysqli_close($this->connection);

                  return $rows;

          }

}

?>
```

Two functions/methods/operations to implement on server

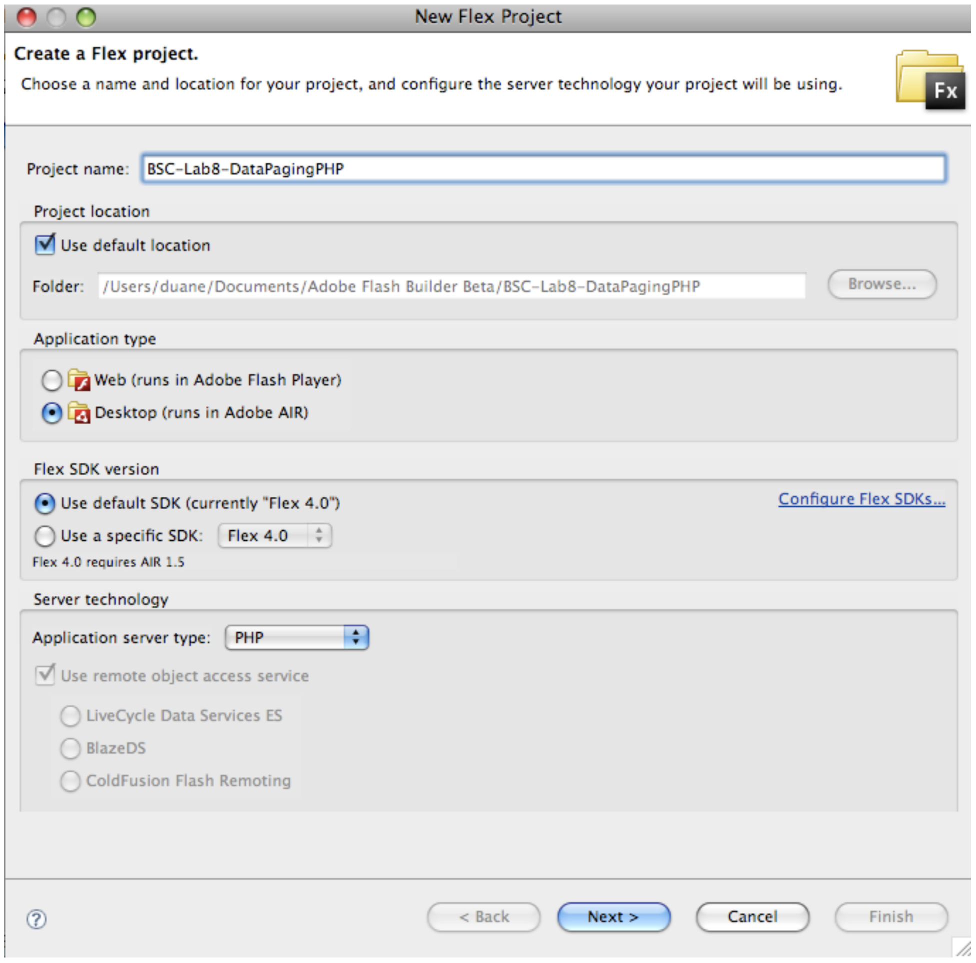
1. Returns number of rows in the database for example “count()”

2. Takes two input arguments and returns the data collection. The two input arguments are start index and number of items to return for example “getItems_paged($startIndex, $numItems)”

Before you begin, set up the CustomerService.php file under your MAMP htdocs root. Below you can see that I used the path htdocs/datapaging/CustomerService.php.



1. Create a new Flex project



New Flex Project

Create a Flex project.

Choose a name and location for your project, and configure the server technology your project will be using.

Project name: BSC-Lab8-DataPagingPHP

Project location

☑ Use default location

Folder: /Users/duane/Documents/Adobe Flash Builder Beta/BSC-Lab8-DataPagingPHP

Application type

○ Web (runs in Adobe Flash Player)

● Desktop (runs in Adobe AIR)

Flex SDK version

● Use default SDK (currently "Flex 4.0")

○ Use a specific SDK: Flex 4.0

Flex 4.0 requires AIR 1.5

Server technology

Application server type: PHP

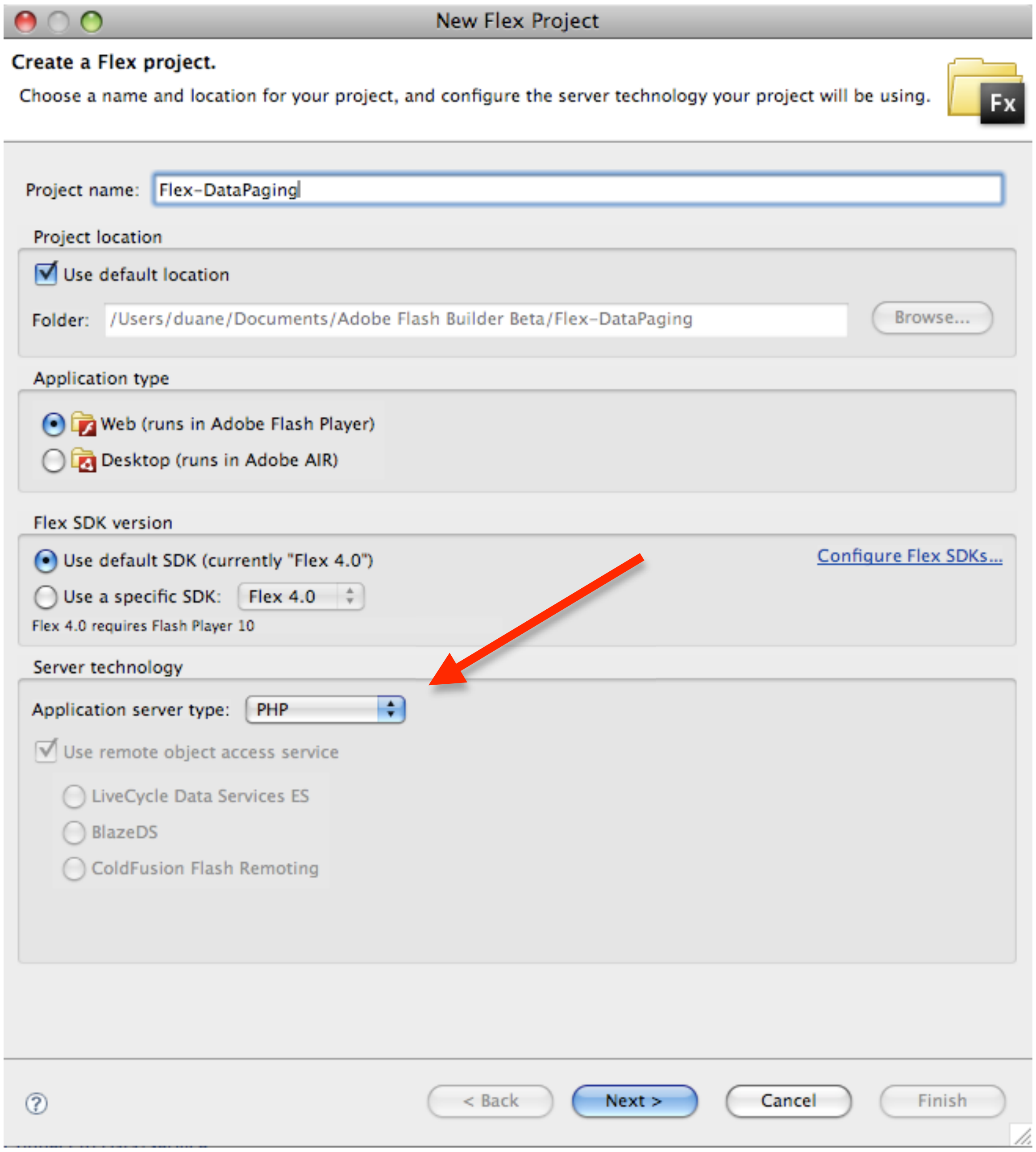
☑ Use remote object access service

○ LiveCycle Data Services ES

○ BlazeDS

○ ColdFusion Flash Remoting

2. Create a new Flex project and enter project details as shown in the image above. Click on next to enter the PHP server details, you will see a window launched as shown in the image below.


New Flex Project

Create a Flex project.

Choose a name and location for your project, and configure the server technology your project will be using.

Project name: Flex-DataPaging

Project location

[✓] Use default location

Folder: /Users/duane/Documents/Adobe Flash Builder Beta/Flex-DataPaging

Application type

[● Web (runs in Adobe Flash Player)]

○ Desktop (runs in Adobe AIR)

Flex SDK version

[● Use default SDK (currently "Flex 4.0")]

○ Use a specific SDK: Flex 4.0

Flex 4.0 requires Flash Player 10

Server technology

Application server type: PHP

[✓] Use remote object access service

○ LiveCycle Data Services ES

○ BlazeDS

○ ColdFusion Flash Remoting

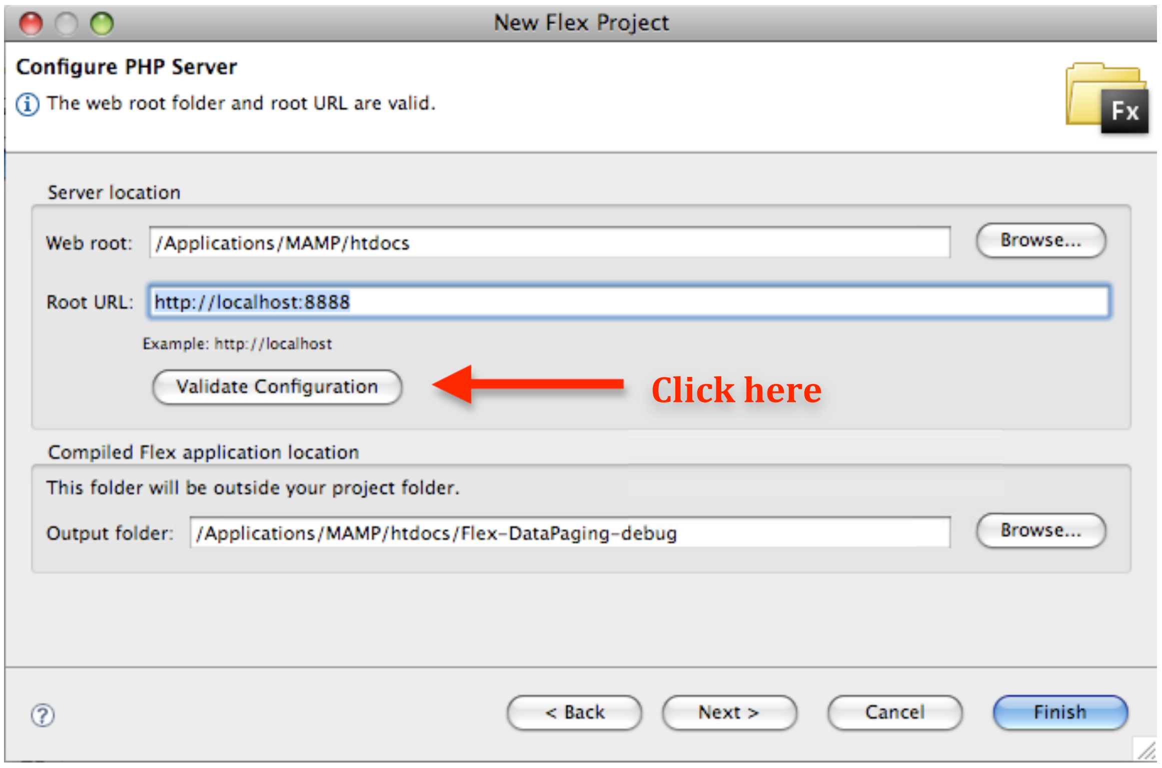
3. In this next screen, validate your projects configuration. Set the Web root to the root folder of your PHP server. In the sample below, it is **`/Applications/MAMP/htdocs`**.

4. Set the root URL to the root URL of your PHP server. Its **http://localhost:8888** in this case

5. Just leave the output folder to default

6. Click the “Validate Configuration” to validate the server details entered

5. Click on finish to continue


New Flex Project

Configure PHP Server

The web root folder and root URL are valid.

Server location

Web root: /Applications/MAMP/htdocs

Root URL: http://localhost:8888

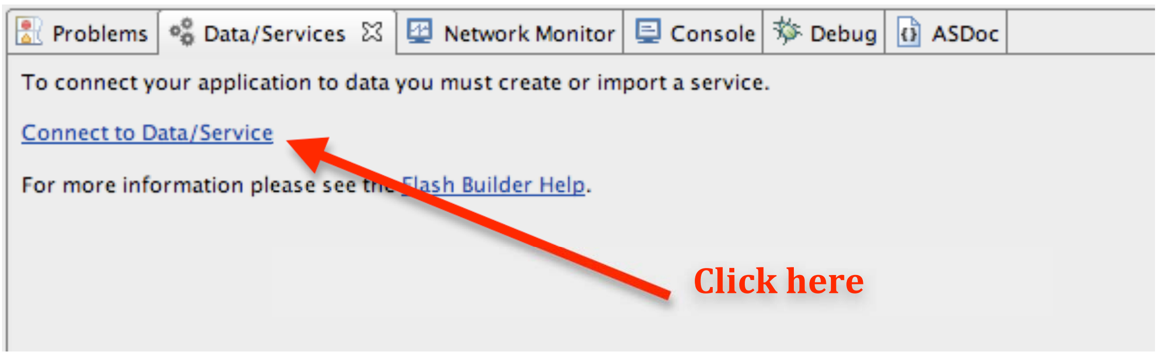
Example: http://localhost

Compiled Flex application location

This folder will be outside your project folder.

Output folder: /Applications/MAMP/htdocs/Flex-DataPaging-debug

6. Create a service using the service Wizard as shown below.



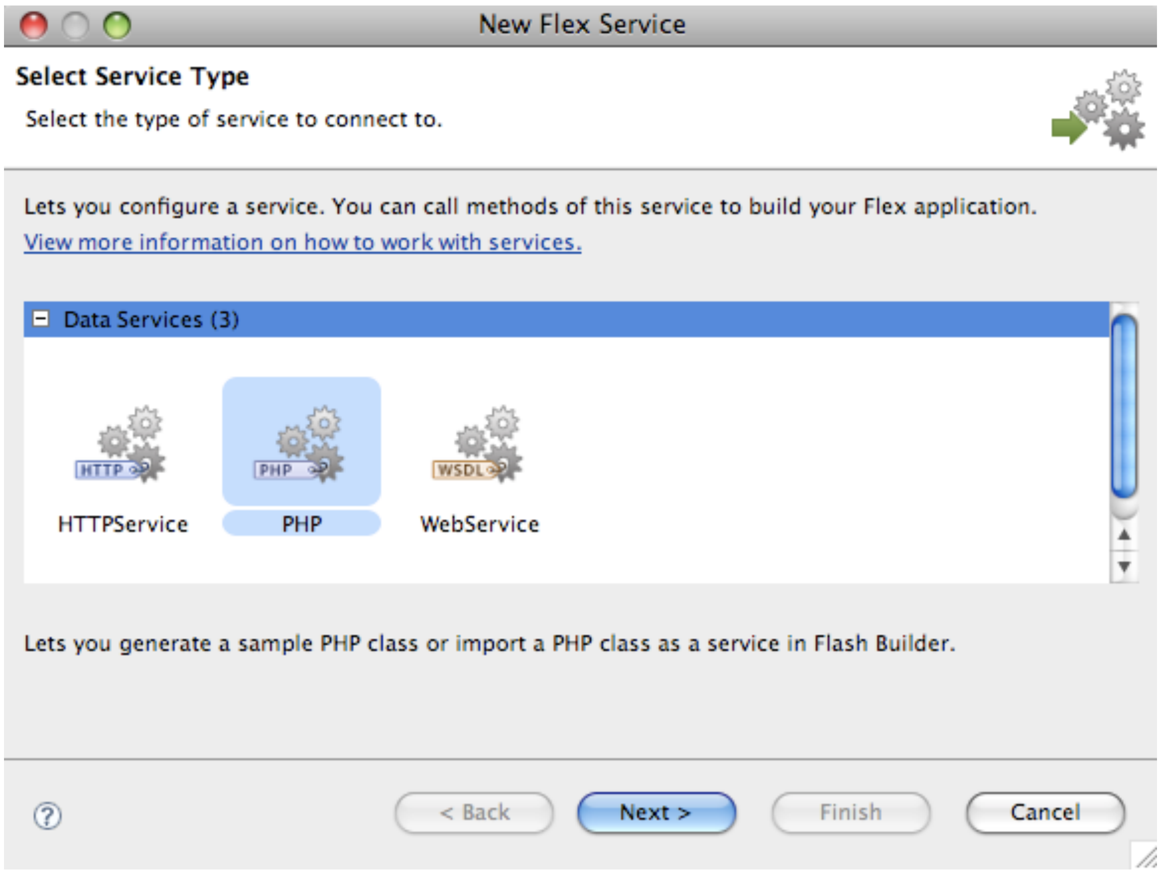
To connect your application to data you must create or import a service.

Connect to Data/Service

For more information please see the Flash Builder Help.

7. You can see Data/Service window, which is called services explorer. Click on the "***Connect to Data/Service***" link to create a new service, you can see a window launched as shown in the image below.

8. Set the service type to PHP.



Select Service Type

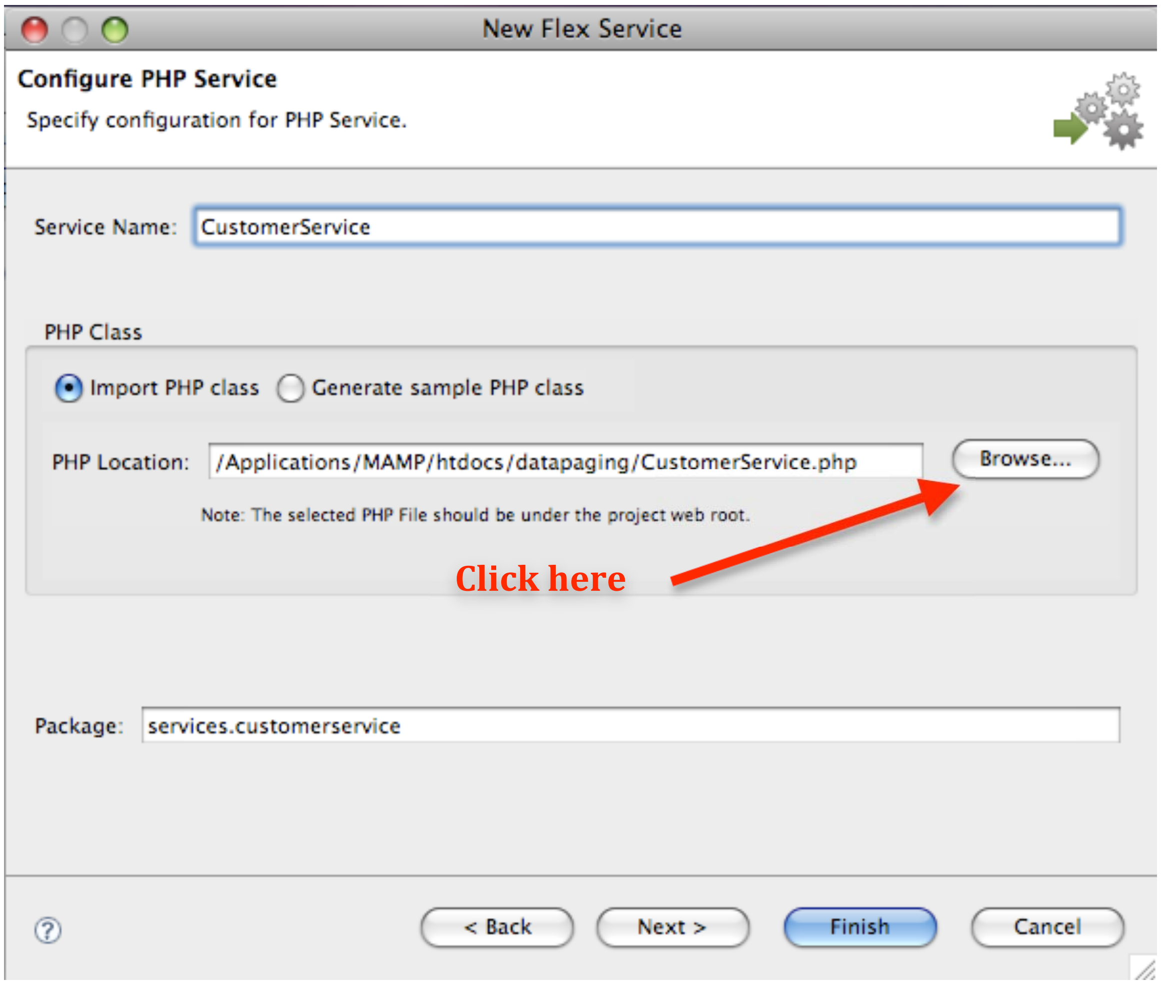
Select the type of service to connect to.

Lets you configure a service. You can call methods of this service to build your Flex application.

View more information on how to work with services.

Lets you generate a sample PHP class or import a PHP class as a service in Flash Builder.

9. In this window we can select the type of service we want to create. Since we want to communicate with the PHP we created, select PHP and click on ***next*** button.


New Flex Service

Configure PHP Service

Specify configuration for PHP Service.

Service Name: CustomerService

PHP Class

[✓ Import PHP class] Generate sample PHP class

PHP Location: /Applications/MAMP/htdocs/datapaging/CustomerService.php

Note: The selected PHP File should be under the project web root.

Package: services.customerservice

10. Name the service as **CustomerService** (note that this happens automatically if you first browse for the file

`<MAMP_HTDOCS_ROOT>/path/CustomerService.php.`

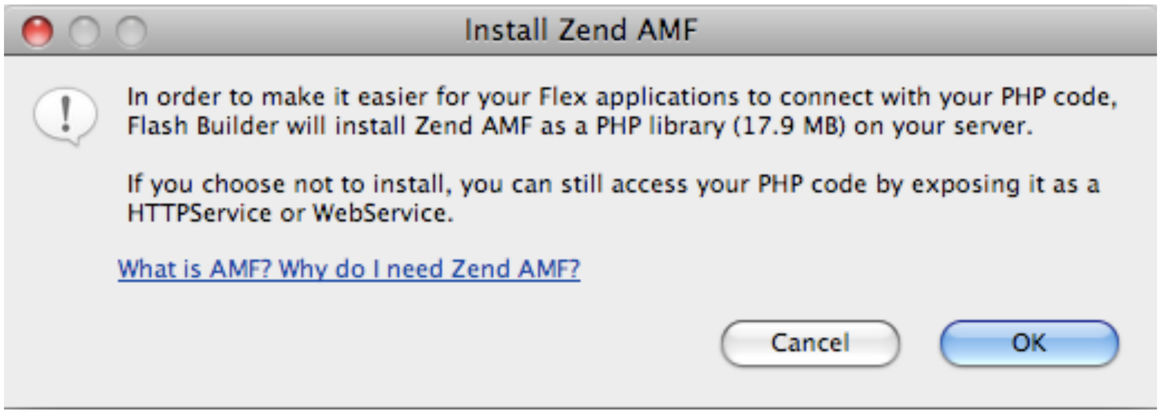
11. Make sure "Import PHP class" radio button is selected, because we are trying reuse the PHP class created.

12. Click on ***next*** button to continue.

Optional:

NOTE: You can copy the CustomerService.php from the end of this courseware in Appendix A.

You might see a window displayed, saying Zend AMF library will be installed. Just click on OK and continue will the set up.



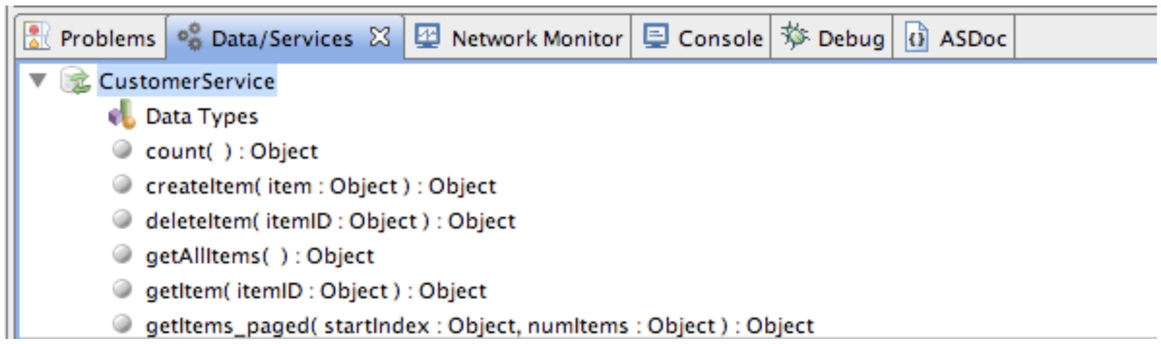
Install Zend AMF

In order to make it easier for your Flex applications to connect with your PHP code, Flash Builder will install Zend AMF as a PHP library (17.9 MB) on your server.

If you choose not to install, you can still access your PHP code by exposing it as a HTTPService or WebService.

What is AMF? Why do I need Zend AMF?

13. Flash Builder will deploy the Zend AMF on your PHP server. Once the installation completes, Flash Builder will display a message. Clicking OK will display the functions in the PHP class. Functions are referred as “operation”. Image below shows the window with operations exposed.

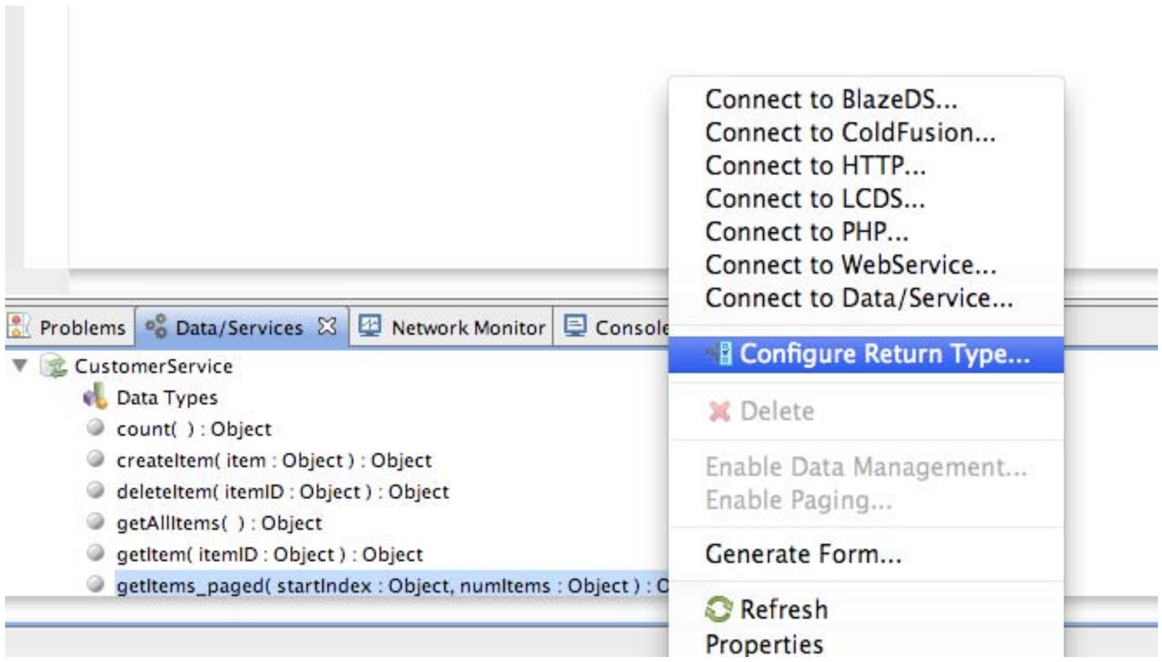


CustomerService
Data Types
count( ) : Object
createItem(item : Object) : Object
deleteItem(itemID : Object) : Object
getAllItems( ) : Object
getItem(itemID : Object) : Object
getItems_paged(startIndex : Object, numItems : Object) : Object

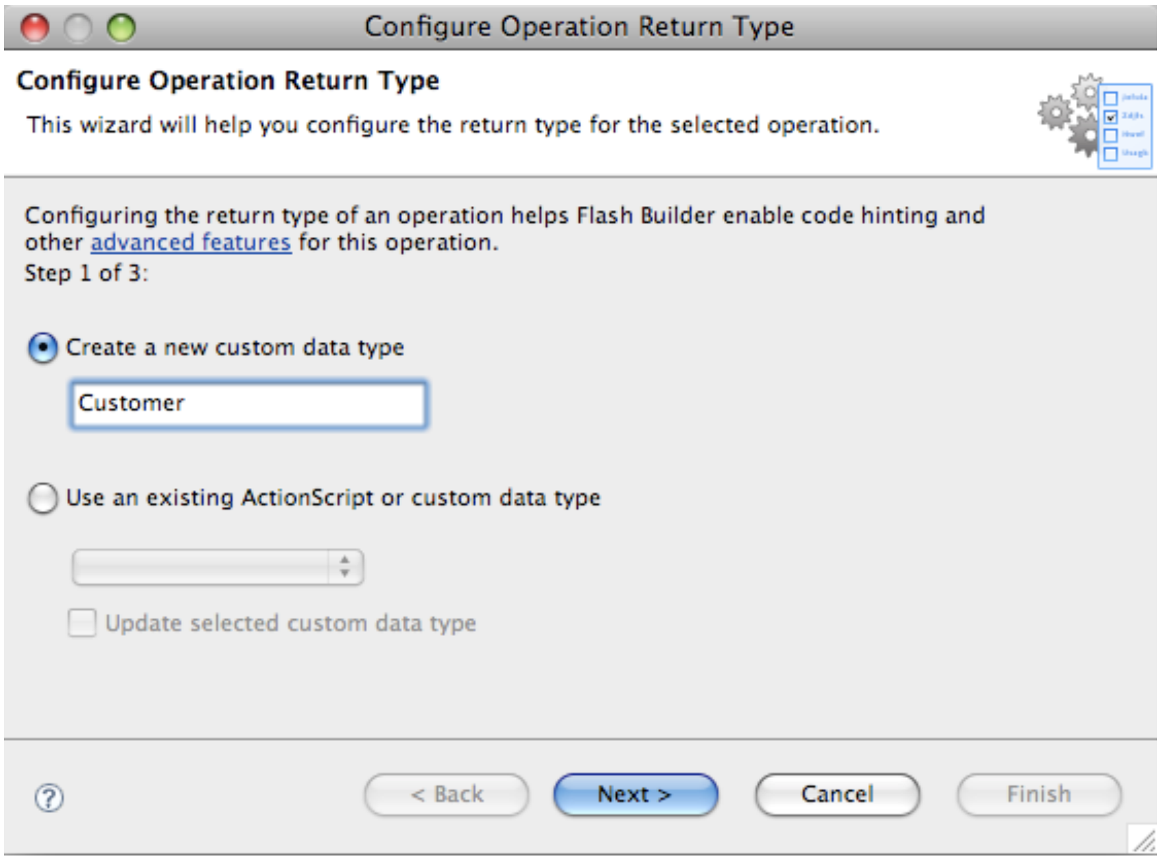
14. Click on finish to continue. A message window displaying next logical steps will be displayed as shown in the image below. You can also see the service created above listed in the services explorer. Just click on the ok button in the message window.

15. In this step we will configure the return type and test the operation and configure the return type on the client i.e. we will specify what type of object to create with the response from the server. This will make it easy for us to develop, since it is easier to deal with strong typed objects.

Right click (Control Click on Mac) on the "getItems_paged()" operation in the services explorer and select "Configure return type" as shown in the image below.



16. A window as shown in the image below will be launched with options to configure the return type. We can chose to create a new data type based on the response from the server. We will configure to create a class named "Customer" with the response.


Configure Operation Return Type

Configure Operation Return Type

This wizard will help you configure the return type for the selected operation.

Configuring the return type of an operation helps Flash Builder enable code hinting and other advanced features for this operation.

Step 1 of 3:

(●) Create a new custom data type

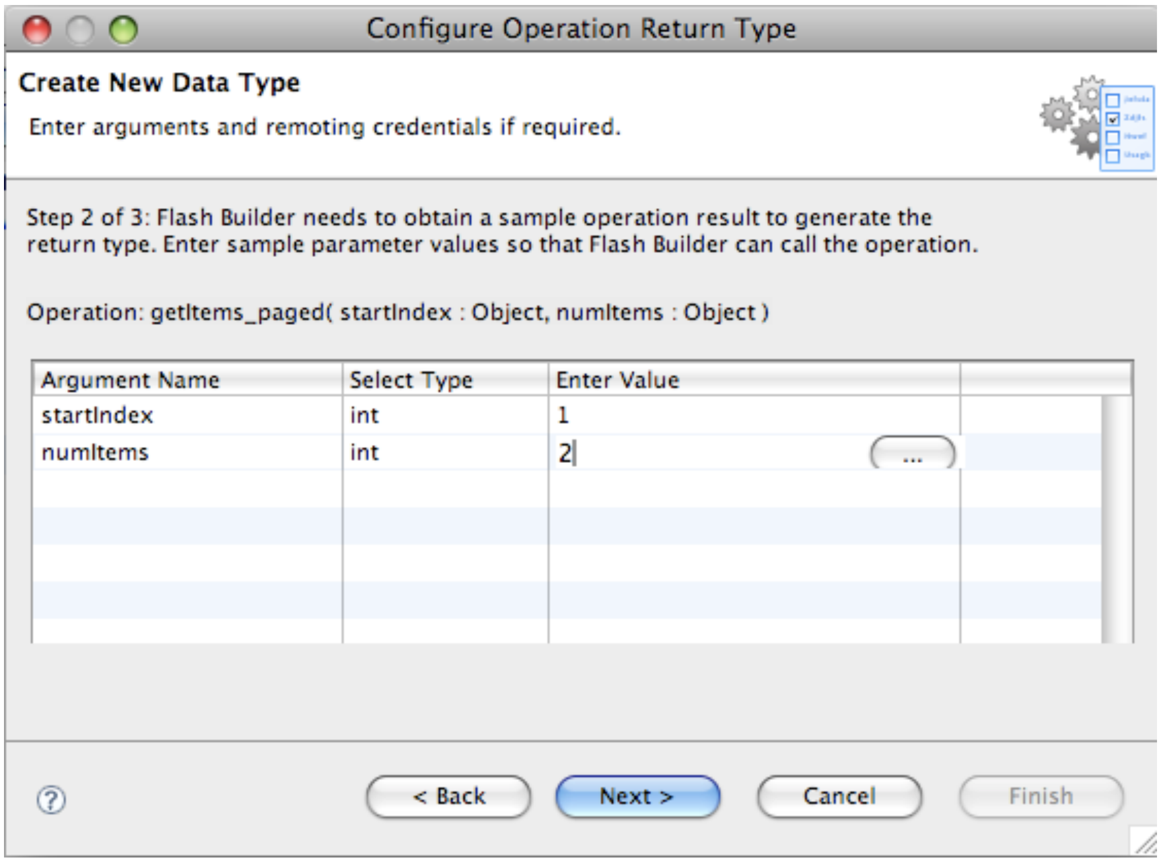
Customer

( ) Use an existing ActionScript or custom data type

☐ Update selected custom data type

17. Enter the name of the class as shown in the image below and click on the next. Next invoke the operation and introspect the response. We can do this by sending a request to the server and see the response to configure the "Customer" class type.

18. Operation requires two integer values as input arguments. Enter those two values as shown in the image above. Since our service doesn't require authentication, select "No" and click on next to continue. You can see the response from the server in the window as shown in the image below.



Configure Operation Return Type

Create New Data Type

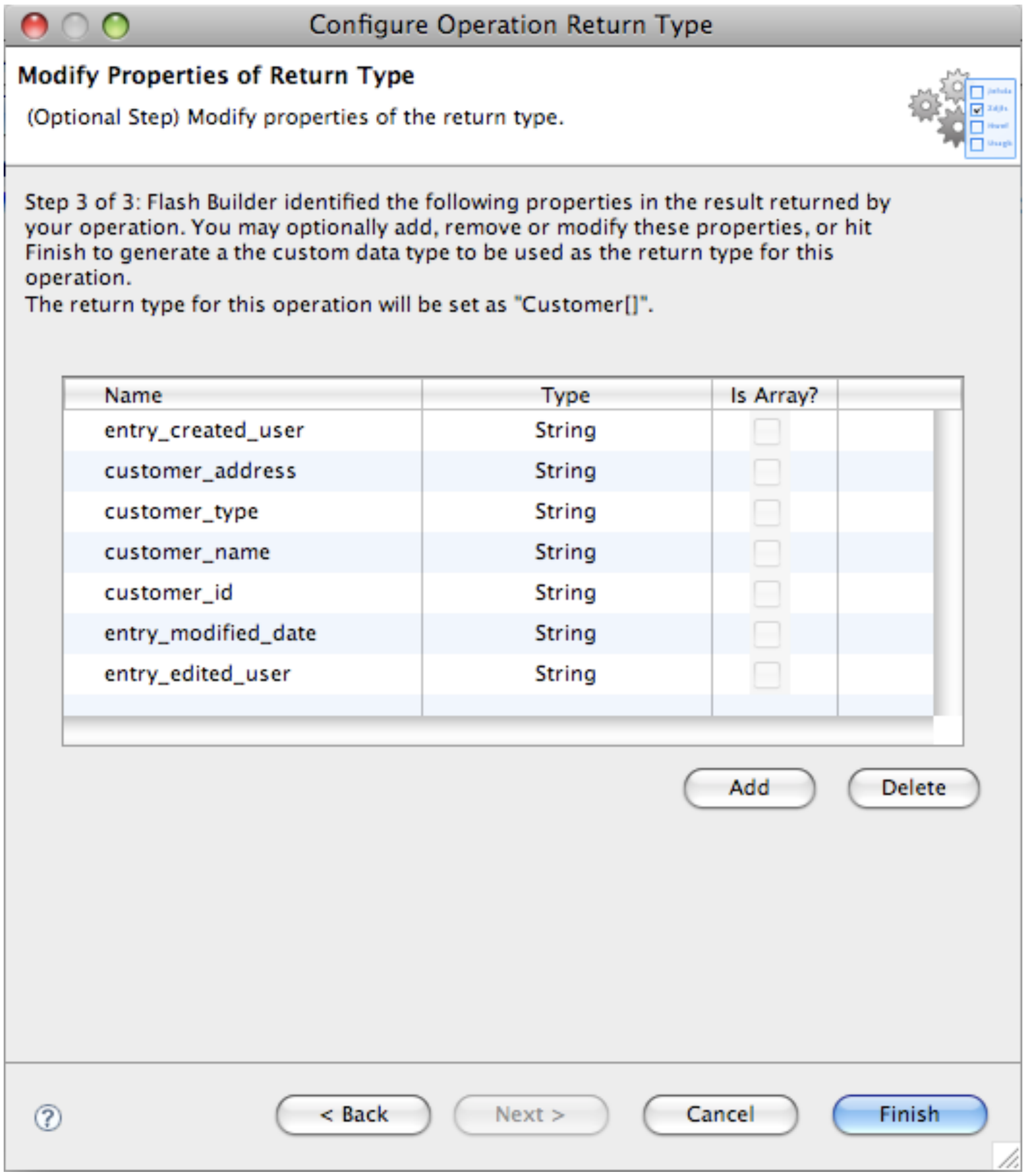
Enter arguments and remoting credentials if required.

Step 2 of 3: Flash Builder needs to obtain a sample operation result to generate the return type. Enter sample parameter values so that Flash Builder can call the operation.

Operation: getItems_paged(startIndex : Object, numItems : Object)

Argument Name	Select Type	Enter Value
startIndex	int	1
numItems	int	2

19. Just click on “Next”. You can see the return type of the operation changed as shown in the image below. When we invoke the “getItems_paged()” operation, response will be object of the type “Customer[]” (note that this is different than “Customer”), in our case ArrayCollection containing Customer objects.



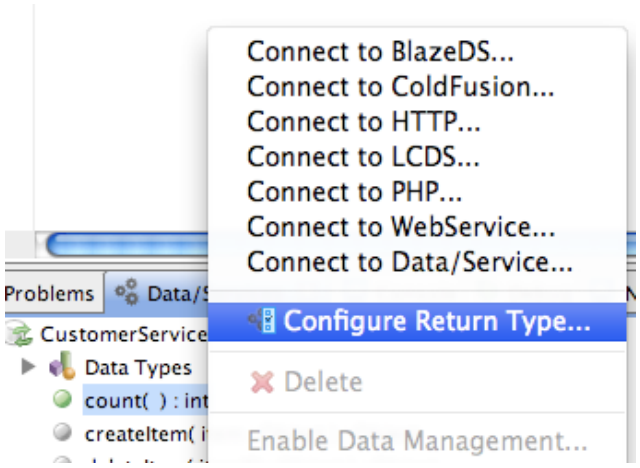
Modify Properties of Return Type

(Optional Step) Modify properties of the return type.

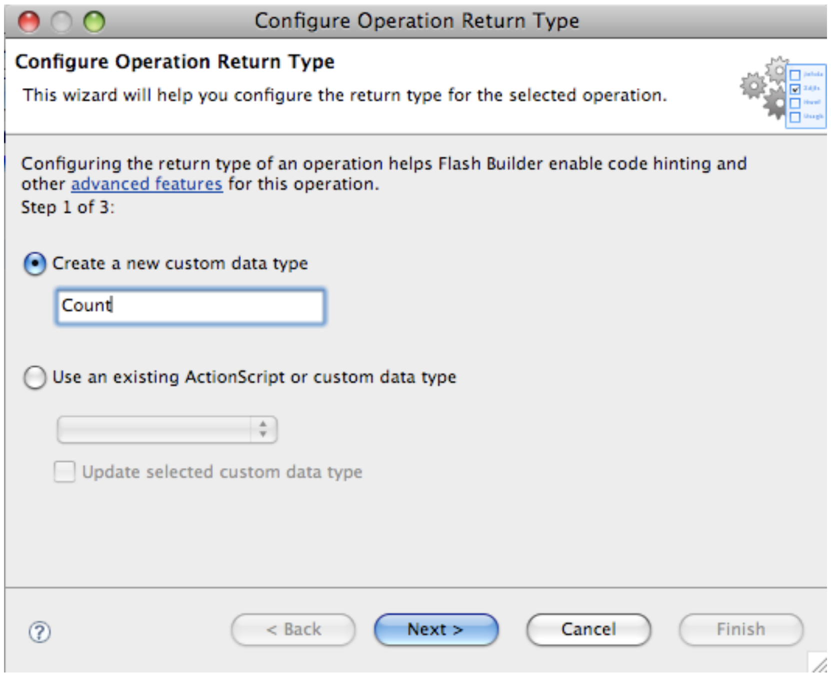
Step 3 of 3: Flash Builder identified the following properties in the result returned by your operation. You may optionally add, remove or modify these properties, or hit Finish to generate a the custom data type to be used as the return type for this operation.
The return type for this operation will be set as "Customer[]".

Name	Type	Is Array?
entry_created_user	String	☐
customer_address	String	☐
customer_type	String	☐
customer_name	String	☐
customer_id	String	☐
entry_modified_date	String	☐
entry_edited_user	String	☐

20. Similarly configure return type for “count()” operation to “int”.



then enter “Count”



Configure Operation Return Type

This wizard will help you configure the return type for the selected operation.

Configuring the return type of an operation helps Flash Builder enable code hinting and other advanced features for this operation.

Step 1 of 3:

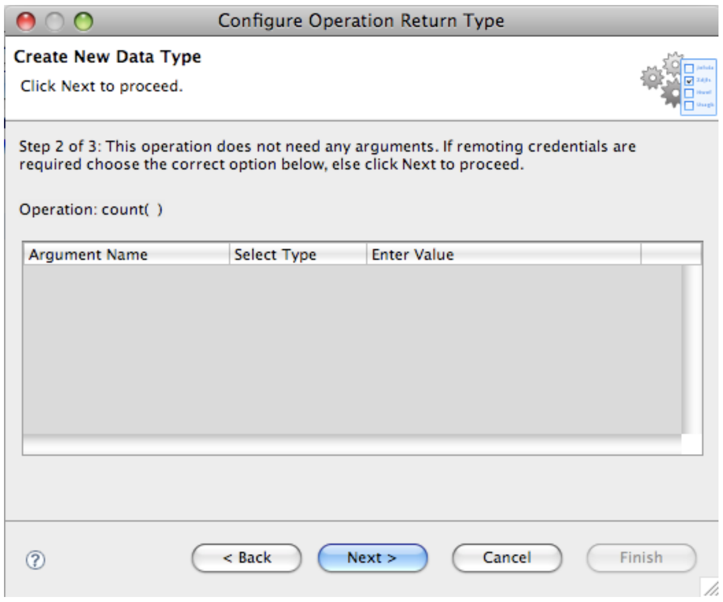
(●) Create a new custom data type

Count

( ) Use an existing ActionScript or custom data type

[ ] Update selected custom data type

Note that the Count return configuration does not require any arguments as the previous example did:

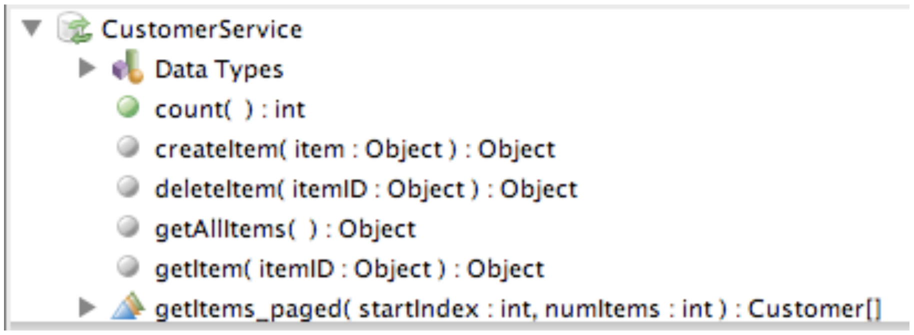


Step 2 of 3: This operation does not need any arguments. If remoting credentials are required choose the correct option below, else click Next to proceed.

Operation: count( )

Argument Name	Select Type	Enter Value

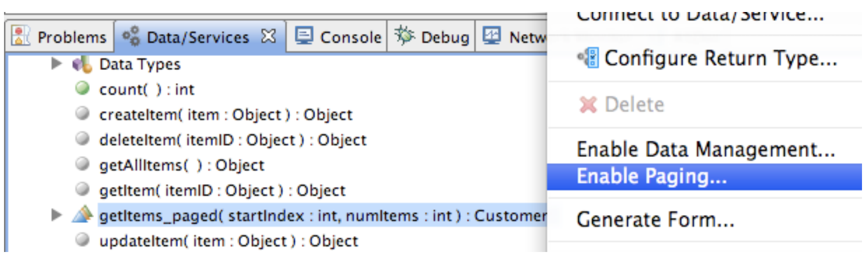
Voila~



```
CustomerService
    Data Types
    count( ) : int
    createItem( item : Object ) : Object
    deleteItem( itemID : Object ) : Object
    getAllItems( ) : Object
    getItem( itemID : Object ) : Object
    getItems_paged( startIndex : int, numItems : int ) : Customer[]
```

Note that the lower “getItems_paged() operation has two parameters in it’s signature. These correspond to the start index and the number of items returned.

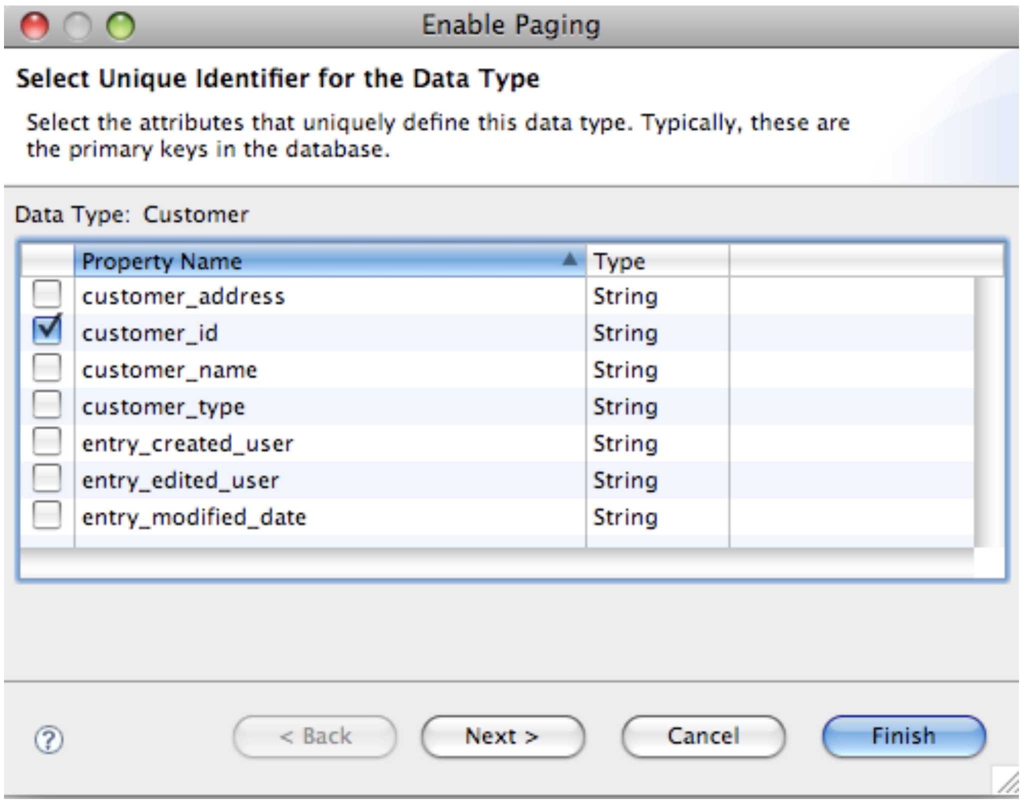
22. To enable paging right click on “getItems_paged()” operation and select “Enable paging” as shown in the image below.


Problems
Data/Services
Console
Debug
Netw
Data Types
count( ) : int
createItem(item : Object) : Object
deleteItem(itemID : Object) : Object
getAllItems( ) : Object
getItem(itemID : Object) : Object
getItems_paged(startIndex : int, numItems : int) : Customer
updateItem(item : Object) : Object
Configure Return Type...
Delete
Enable Data Management...
Enable Paging...
Generate Form...

23. You will get a window as shown in the image below, where you need to select the primary key for the Customer class. If you recall the SQL used to create the database as shown below (Bold red font), the primary key was customer_id.

```
DROP TABLE IF EXISTS `customers`;
CREATE TABLE `customers` (
 `customer_id` int(11) NOT NULL auto_increment,
 `customer_name` varchar(225) default NULL,
 `customer_address` blob,
 `customer_type` varchar(100) default NULL,
 `entry_created_user` int(11) default NULL,
 `entry_edited_user` int(11) default NULL,
 `entry_modified_date` datetime default NULL,

 PRIMARY KEY  (`customer_id`),

 UNIQUE KEY `customer_name` (`customer_name`)

) ENGINE=InnoDB AUTO_INCREMENT=82 DEFAULT CHARSET=utf8;

...
```

24. Use the primary key and check off the box beside it as shown below and hit “Next” (not “Finish”).


Enable Paging

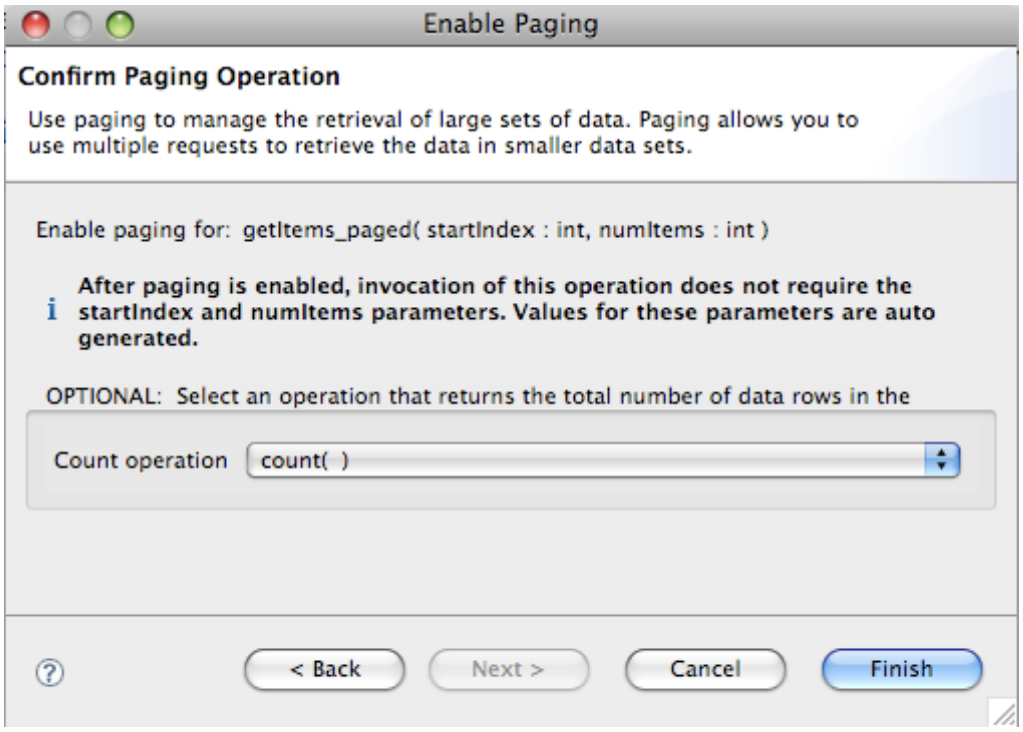
Select Unique Identifier for the Data Type

Select the attributes that uniquely define this data type. Typically, these are the primary keys in the database.

Data Type: Customer

	Property Name	Type
☐	customer_address	String
✓	customer_id	String
☐	customer_name	String
☐	customer_type	String
☐	entry_created_user	String
☐	entry_edited_user	String
☐	entry_modified_date	String

25. The dialog screen will come up and prompt you to select the operation to use for getting total count of records. Flex 4 can page small sets of data and does not require parameters be sent with subsequent paging requests. Count() is used to track how many records are left to use count() for this setting as shown below.



Use paging to manage the retrieval of large sets of data. Paging allows you to use multiple requests to retrieve the data in smaller data sets.

Enable paging for: getItems_paged(startIndex : int, numItems : int)

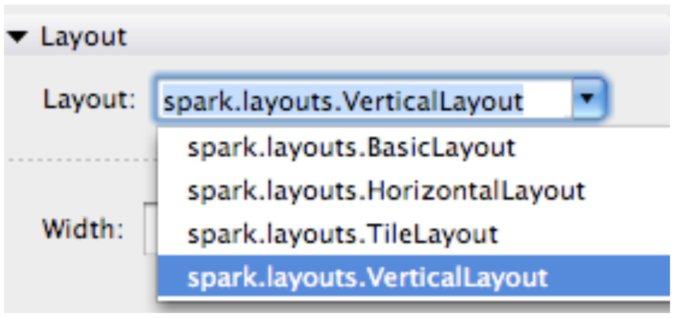
After paging is enabled, invocation of this operation does not require the startIndex and numItems parameters. Values for these parameters are auto generated.

OPTIONAL: Select an operation that returns the total number of data rows in the

Count operation: count( )

26. All that is left to do it to build a graphical user interface to control how we fetch data. We have enabled data paging and can demonstrate how data is fetched on demand automatically

27. Switch to design view as shown in the image below and change the Application layout to “vertical”.

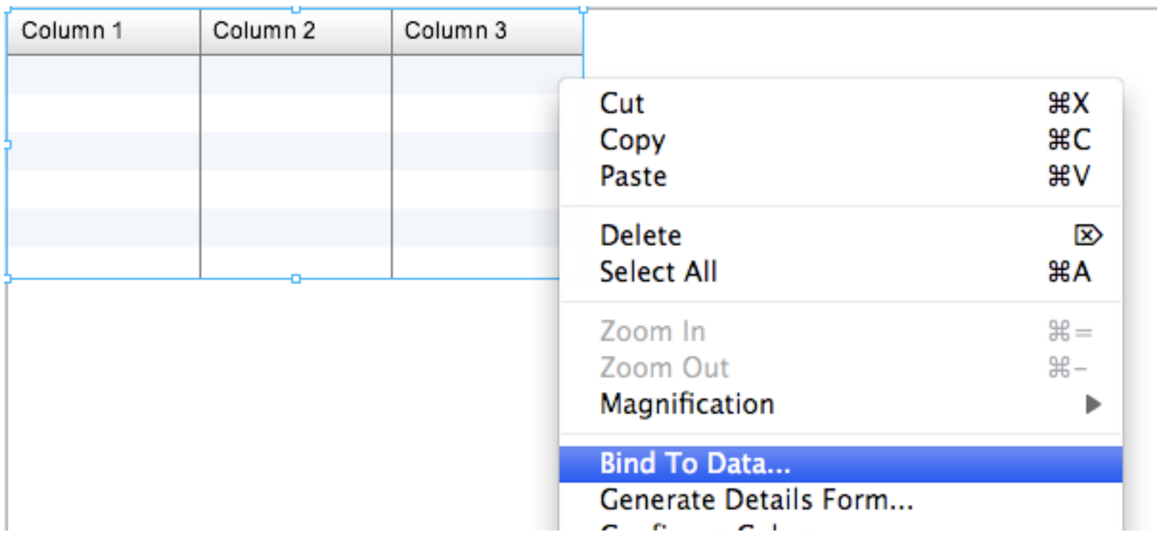


Layout: spark.layouts.VerticalLayout

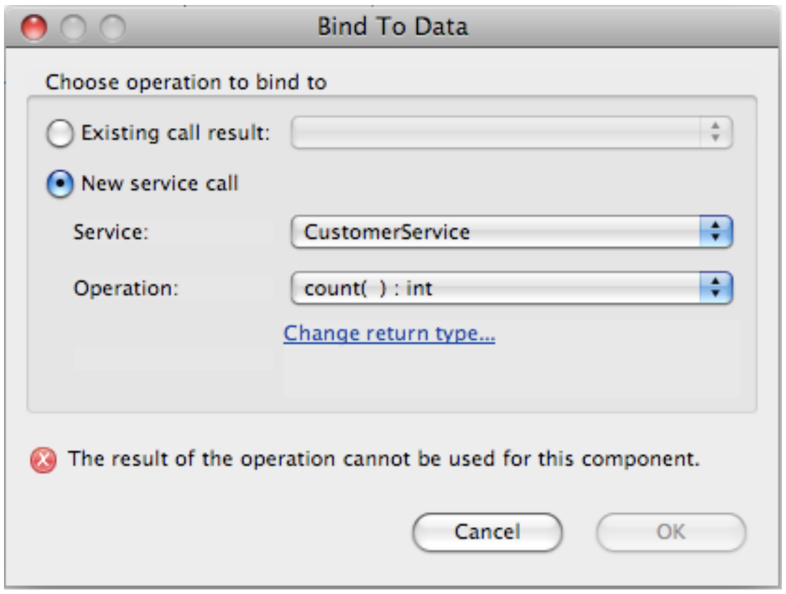
spark.layouts.BasicLayout
spark.layouts.HorizontalLayout
spark.layouts.TileLayout
spark.layouts.VerticalLayout

28. Drag and drop a DataGrid from the components panel to the design view and make sure it is selected (the outer lines turn blue).

29. Right click on the DataGrid and select “Bind To Data”



30. Note the red X at the bottom warning you that the result of the operation cannot be used for this component. This is because count() is chosen by default. Change the Operation to getItems_pages():Customer[].


Bind To Data

Choose operation to bind to

Existing call result:

New service call

Service: CustomerService

Operation: count( ) : int

Change return type...

The result of the operation cannot be used for this component.

In this screen, select “Customer Service” and “getItems_paged()” operation and click ok and continue.

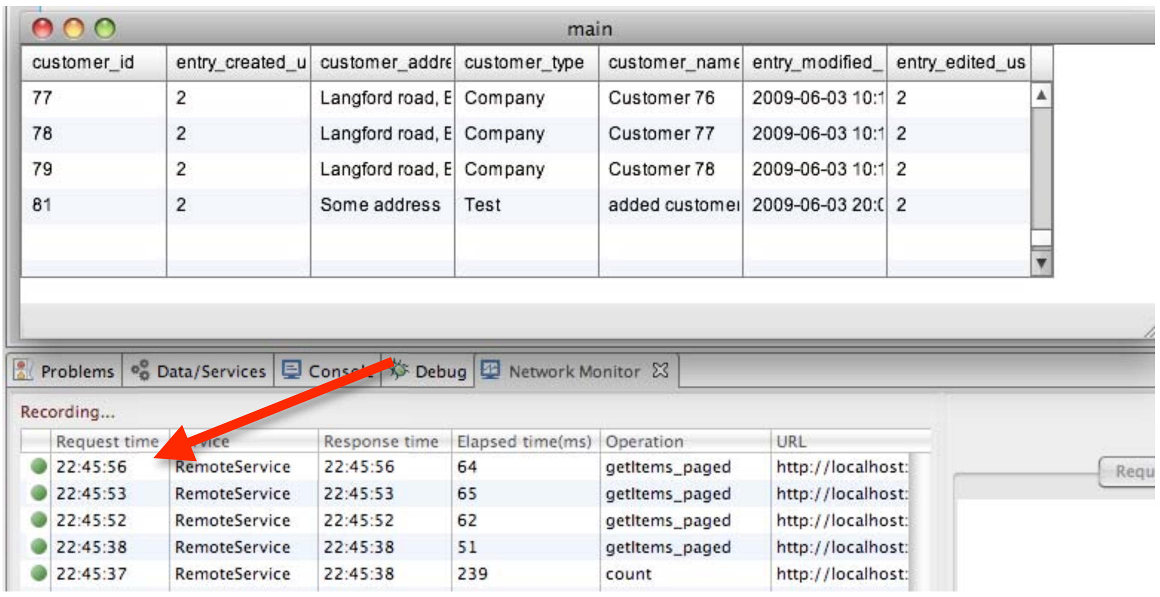
Optional:

Save the application and run. Paging is enabled and the data will be retrieved only when it is required, for example when you scroll the DataGrid to a row whose data is not retrieved, the data gets loaded from server automatically and displayed. You can change the paging size by adding following lines of code as shown in the image below.

```
var dm:DataManager = customerService.getDataManager(
customerService.DATA_MANAGER_CUSTOMER);
dm.pageSize = 25;
```

WHEN YOU RUN:

To demonstrate the data paging, make sure the network monitor is viewable so you can see additional calls as the datagrid is scrolled down:



customer_id	entry_created_u	customer_addre	customer_type	customer_name	entry_modified_	entry_edited_us
77	2	Langford road, E	Company	Customer 76	2009-06-03 10:1	2
78	2	Langford road, E	Company	Customer 77	2009-06-03 10:1	2
79	2	Langford road, E	Company	Customer 78	2009-06-03 10:1	2
81	2	Some address	Test	added custome	2009-06-03 20:(	2

Recording...

Request time	...vice	Response time	Elapsed time(ms)	Operation	URL
22:45:56	RemoteService	22:45:56	64	getItems_paged	http://localhost:
22:45:53	RemoteService	22:45:53	65	getItems_paged	http://localhost:
22:45:52	RemoteService	22:45:52	62	getItems_paged	http://localhost:
22:45:38	RemoteService	22:45:38	51	getItems_paged	http://localhost:
22:45:37	RemoteService	22:45:38	239	count	http://localhost:

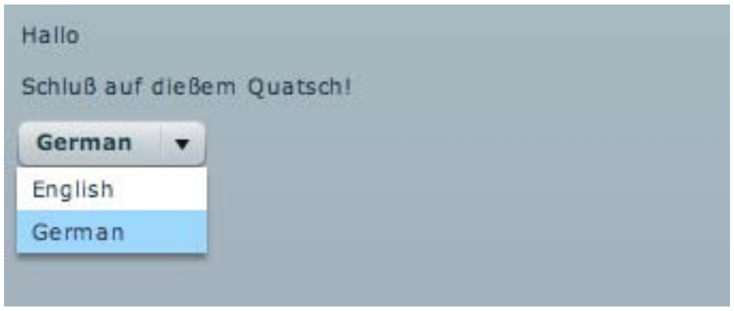
Project 27: Solution Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/halo"
xmlns:customerservice="services.customerservice.*">
	<fx:Script>
		<![CDATA[
			import services.customerservice.Customer;
			import mx.rpc.AsyncToken;
			import mx.data.DataManager;
			import mx.events.FlexEvent;
			import mx.controls.Alert;


			protected function
dataGrid_creationCompleteHandler(event:FlexEvent):void
			{
				getItems_pagedResult.token =
customerService.getItems_paged();
			}
		]]>
	</fx:Script>
	<mx:DataGrid id="dataGrid"
creationComplete="dataGrid_creationCompleteHandler(event)"
dataProvider="{getItems_pagedResult.lastResult}" editable="true" height="157"
width="696">
		<mx:columns>
			<mx:DataGridColumn headerText="customer_id"
dataField="customer_id" editable="false"/>
			<mx:DataGridColumn headerText="entry_created_user"
dataField="entry_created_user"/>
			<mx:DataGridColumn headerText="customer_address"
dataField="customer_address"/>
			<mx:DataGridColumn headerText="customer_type"
dataField="customer_type"/>
			<mx:DataGridColumn headerText="customer_name"
dataField="customer_name"/>
			<mx:DataGridColumn headerText="entry_modified_date"
dataField="entry_modified_date"/>
			<mx:DataGridColumn headerText="entry_edited_user"
dataField="entry_edited_user"/>
		</mx:columns>
	</mx:DataGrid>

	<s:layout>
		<s:VerticalLayout/>
	</s:layout>
	<fx:Declarations>
		<s:CallResponder id="getItems_pagedResult"/>
		<customerservice:CustomerService id="customerService"
```

```
destination="CustomerService" endpoint="http://localhost:8888/BSC-Lab8-
DataPagingPHP-debug/gateway.php" fault="Alert.show(event.fault.faultString)"
showBusyCursor="true" source="CustomerService"/>
	</fx:Declarations>

</s:WindowedApplication>
```

Project 27: Test Questions

Project 28: Internationalization & Resource Bundles



Major thanks and credit to Nick Bilyk for this Project!

The fl.lang.Locale class allows you to control how multilanguage text is displayed in a SWF file. Within the Flex Builder and Flash Builder development environments, the Flash Strings panel allows you to use string IDs instead of string literals in dynamic text fields. This allows you to create a SWF file that displays text loaded from a language-specific XML file (part of the resource bundles). The XML file must use the XML Localization Interchange File Format (XLIFF).

XLIFF is an XML-based format that enables translators to concentrate on the text to be translated. Likewise, since it's a standard, manipulating XLIFF files makes localization engineering easier: once you have converters written for your source file formats, you can simply write new tools to deal with XLIFF and not worry about the original file format. It also supports a full localization process by providing tags and attributes for review comments, the translation status of individual strings, and metrics such as word counts of the source sentences. For more on XLIFF, check in here:

http://developers.sun.com/dev/gadc/technicalpublications/articles/xliff.html

There are three ways to display the language-specific strings contained in the XLIFF files:

* "automatically at runtime"—Flash Player replaces string IDs with strings from the XML file matching the default system language code returned by flash.system.capabilities.language.

* "manually using stage language"—String IDs are replaced by strings at compile time and cannot be changed by Flash Player.

* "via ActionScript at runtime"—String ID replacement is controlled using ActionScript at runtime. This option gives you control over both the timing and language of string ID replacement.

You can use the properties and methods of this class when you want to replace the string IDs "via ActionScript at runtime."

All of the properties and methods available are static, which means that they are accessed through the fl.lang.Locale class itself rather than through an instance of the class.

Note: The Locale class is installed in the Flash Authoring classpath and is automatically compiled into your SWF files. Using the Locale class increases the SWF file size slightly since the class is compiled into the SWF.

1. Import the localization project into your Flex Builder 3. This should be in the attendee Projects Folder under the name ABC28-Internationalization.
2. Open the project and run it. It should look like the screenshot above and contain the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
  <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            public var labels:ArrayCollection = new ArrayCollection(
                [ {label:"English", data:1},
                  {label:"German", data:2},
                ]);
        ]]>
    </mx:Script>

      <mx:VBox x="10" y="10" height="100%" width="100%">
             <mx:Label text="Hello"/>
             <mx:Text text="Say something"/>
            <mx:ComboBox id="localesCB" dataProvider="{labels}"  />
      </mx:VBox>
</mx:Application>
```

3. The first thing we have to do is to create the model for the locales. Use the <mx:Model> tag to declare a data model in MXML. An <mx:Model> tag is compiled into a tree of ActionScript objects; the leaves of the tree are scalar values.

   There are two locales declared within the project folder named "locale". There are two resource bundles – one for English and one for German.   Add the following code to the application just below the Script tag.

```
      <mx:Model id="locales">
            <locales>
                  <locale label="English" data="en_US"/>
                  <locale label="German" data="de_DE"/>
            </locales>
      </mx:Model>
```

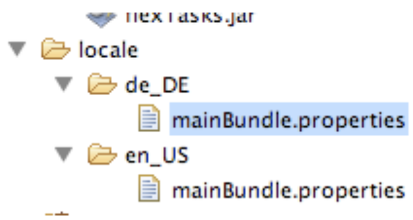
4. To use these bundles, we have to use the references in the projects. There is a file called resouceBundles.xml in the root folder of the project. Open up this file and change the path to match the path to your Flex SDK. This sets the {FLEX_HOME} directory to your own SDK.

```
<!-- CHANGE TO YOUR FLEX DIRECTORY //-->
<property name="FLEX_HOME" value="/Applications/Adobe Flex Builder 3 Plug-
in/sdks/3.2.0"/>
```

5. Lower down in this file you will see references for a resource bundle named "mainBundle". This sets the path and we will now use this as a way to build assets which can be dynamically loaded at runtime instead of static values in the application for labels.

```
<target name="en_US">
   <mxmlc>
       <locale>en_US</locale>
          <source-path>locale/{locale}</source-path>
          <include-resource-bundles>mainBundle</include-resource-bundles>
          <output>bin-debug/Resources_en_US.swf</output>
   </mxmlc>
</target>
```

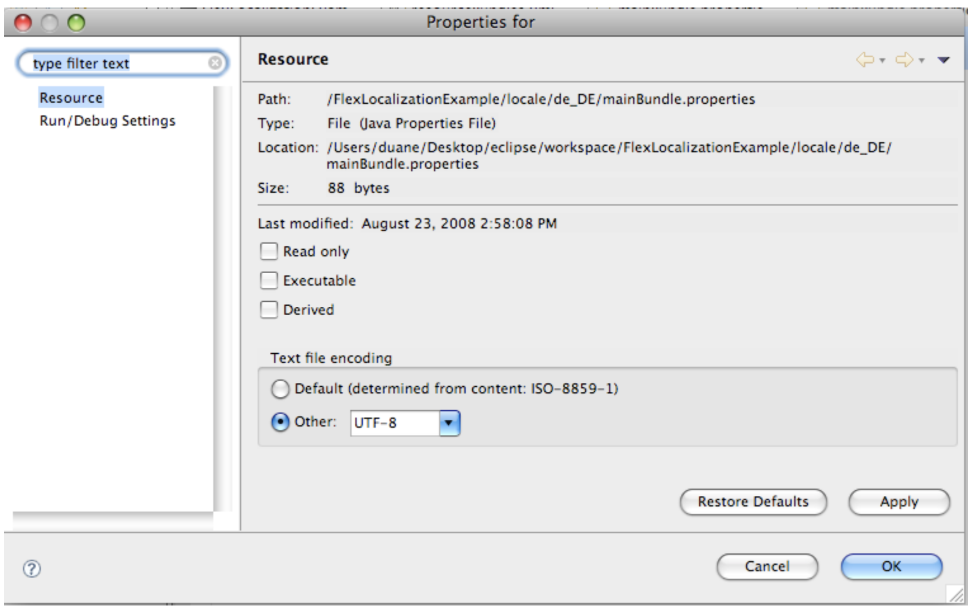
6. In the project you will find two files that contain definitions for the labels. These are located under the locale directory. It is very important that these files are in UTF-8 format. By default, when you create a new properties file, it will be in ISO-8859 format.



7. To ensure these are in the right format, open one up. It should look as below.

```
# locale/de_DE/myBundle.properties
greeting=Hallo
message=Schluß auf dießem Quatsch!
```

To check the format, with the file open, click "File" and "Properties". You should see a screen like the one below.



Path: /FlexLocalizationExample/locale/de_DE/mainBundle.properties
Type: File (Java Properties File)
Location: /Users/duane/Desktop/eclipse/workspace/FlexLocalizationExample/locale/de_DE/mainBundle.properties
Size: 88 bytes

Last modified: August 23, 2008 2:58:08 PM
☐ Read only
☐ Executable
☐ Derived

Text file encoding
○ Default (determined from content: ISO-8859-1)
● Other: UTF-8

8. Now the ComboBox in the Application itself has to be modified to call a new function and also use the resource bundles. Currently, the code hard binds the model to the view as written:

```
        <mx:VBox x="10" y="10" height="100%" width="100%">
               <mx:Label text="Hello"/>
               <mx:Text text="Say something"/>
               <mx:ComboBox id="localesCB" dataProvider="{labels}"  />
        </mx:VBox>
```

The values for both the Label and the Text are hard coded. In order to swap out labels on an application, you must use a reference to a value and treat each value as a variable bound ot the component. The correct syntax for this is as shown below. Also add a change event handler and call a new functi0on (which needs to be written) called changeLanguage() and passes the change event. Set the data provider to the ComboBox to locales.locale.

```
<mx:VBox x="10" y="10" height="100%" width="100%">
<mx:Label text="{resourceManager.getString('mainBundle', 'greeting')}"/>
<mx:Text text="{resourceManager.getString('mainBundle', 'message')}"/>
<mx:ComboBox id="localesCB" dataProvider="{locales.locale}" labelField="label"
                    change="changeLanguage(event);" />
</mx:VBox>
```

9. The function must now be written. Delete the variable that was previously used as a data provider (labels). Add the function for changeLanguage and remember to accept an event (you will get an error otherwise).

```
<mx:Script>
   <![CDATA[
       private function changeLanguage(event:Event):void
       {

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

10. Now add a member variable to the function that will hold the value of the referenced language from the ComboBox. The ID of the ComboBox is localesCB. Use the "selectedItem" accessor to grab the selected value. Note that this will only be called on a change event.

```
var newLocale:String = localesCB.selectedItem.data;
```

11. Next we have to create a new variable of type IEventDispatcher and call the Resource manager **mx.core.UIComponent.resourceManager():IResourceManager.** This is a read only reference to the object which manages all of the application's localized resources. This is a singleton instance which implements the IResourceManager interface and the property can be used as the source for data binding. After the program has been compiled, this will access the swf resource for the proper localized-language components.

```
var eventDispatcher:IEventDispatcher =
resourceManager.loadResourceModule("Resources_" + newLocale + ".swf");
```

12. We now have to build an array to pass as arguments to the resourceManager. When you call the ResourceManager methods getObject(), getString(), getStringArray(), getNumber(), getInt(), getUint(), getBoolean(), or getClass() to get the value of a resource, you specify a bundle name and a resource name, but not a locale. The ResourceManager starts with the first locale in the localeChain and looks for a ResourceBundle with the specified bundle name for that locale. If such a ResourceBundle exists, and the specified resource exists in it, then the value of that resource is returned. Otherwise, the ResourceManager proceeds on to the other locales in the localeChain.

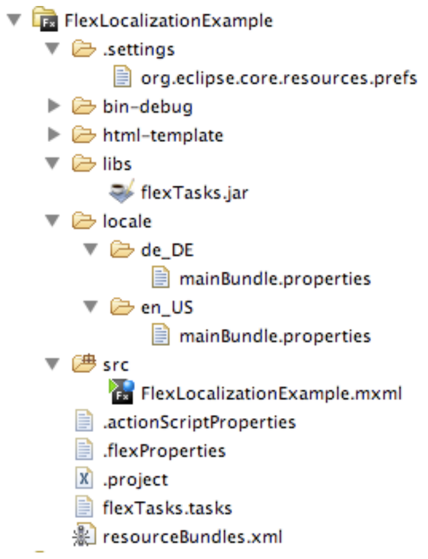
13. Add the following lines of code to your application. Note that by default, the language localeChain will default to the en_US. The second line sets the resourceManager's localeChain to the default value.

```
var localeChain:Array = [newLocale, "en_US"];
resourceManager.localeChain = localeChain;
```

14. Run Project.

Project 28: Solution Code

Here is the application structure:



```
FlexLocalizationExample
    .settings
        org.eclipse.core.resources.prefs
    bin-debug
    html-template
    libs
        flexTasks.jar
    locale
        de_DE
            mainBundle.properties
        en_US
            mainBundle.properties
    src
        FlexLocalizationExample.mxml
    .actionScriptProperties
    .flexProperties
    .project
    flexTasks.tasks
    resourceBundles.xml
```

Here is the code:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">

<mx:Script><![CDATA[

private function changeLanguage(event:Event):void
{
   var newLocale:String = localesCB.selectedItem.data;
   var eventDispatcher:IEventDispatcher =
       resourceManager.loadResourceModule("Resources_" + newLocale + ".swf");
       var localeChain:Array = [newLocale, "en_US"];
       resourceManager.localeChain = localeChain;
}
]]>
</mx:Script>

<mx:Model id="locales">
   <locales>
       <locale label="English" data="en_US"/>
       <locale label="German" data="de_DE"/>
   </locales>
</mx:Model>

<mx:VBox x="10" y="10" height="100%" width="100%">
 <mx:Label text="{resourceManager.getString('mainBundle', 'greeting')}"/>
 <mx:Text text="{resourceManager.getString('mainBundle', 'message')}"/>
 <mx:ComboBox id="localesCB" dataProvider="{locales.locale}" labelField="label"
                    change="changeLanguage(event);" />
</mx:VBox>
</mx:Application>
```

Project 28: Test questions

Q: Can you use the iResourceManager interface to set values?

Q: What standarde does Flex use for internationalization and resource bundle formats?

Q: What encoding should be used for resource bundles?

About the Authors:

This document is a combination of the efforts of all Adobe Evangelists. It is a living document and we hope you enjoyed it as much as we enjoyed building it.