

Adobe AIR 2 Boot Camp

Adobe AIR 2 Boot Camp.....	1
Forward.....	2
Audience Assumptions	2
Pre-Requisites	2
About the trainer.....	3
Project 1. Getting used with Flex Builder, design view, data binding, events and events listeners, MXML and ActionScript 3.....	4
Project 2. Chromeless applications	7
Project 3. Exporting, Signing, Distributing and installing AIR applications.....	11
Project 4. HTML capabilities in AIR.....	14
Project 5. Reading and Writing Files to local hardisk.....	16
Project 6. Local database in AIR, SQLite	19
Project 7. Serialize ActionScripts objects to the local hardisk	23
Project 8. Using CSS	26
Project 10. Talking to a server: REST style.....	29
Project 11. Window API's.....	35
Project 12. Volume Storage Detection.....	37
Project 13. Open a file with Default Application	39
Project 14. Native Processes.....	40
Project 15. Drag and Drop.....	42

Forward

Welcome to the 2010 Adobe AIR Boot Camp. This course has been put together in hopes to provide developers a boot camp to learn all the basics of Adobe AIR development in one 3 hour session. 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 AIR 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.

This course is written in such a manner that you should be able to take this alone as a self paced tutorial, although during live session, we will be there to lead you through it and help in the event you encounter any problems.

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.

IMPORTANT: There is a high probability we will not get through the entire course during the time allotted at this lab. 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.

Audience Assumptions

- This is a 101 Boot camp to introduce people to AIR who have never used it before or want to go past Hello World.
- Attendees have Adobe's AIR 2 runtime installed
- Attendees are roughly familiar with XML syntax rules
- Attendees have Flash Builder 4 installed

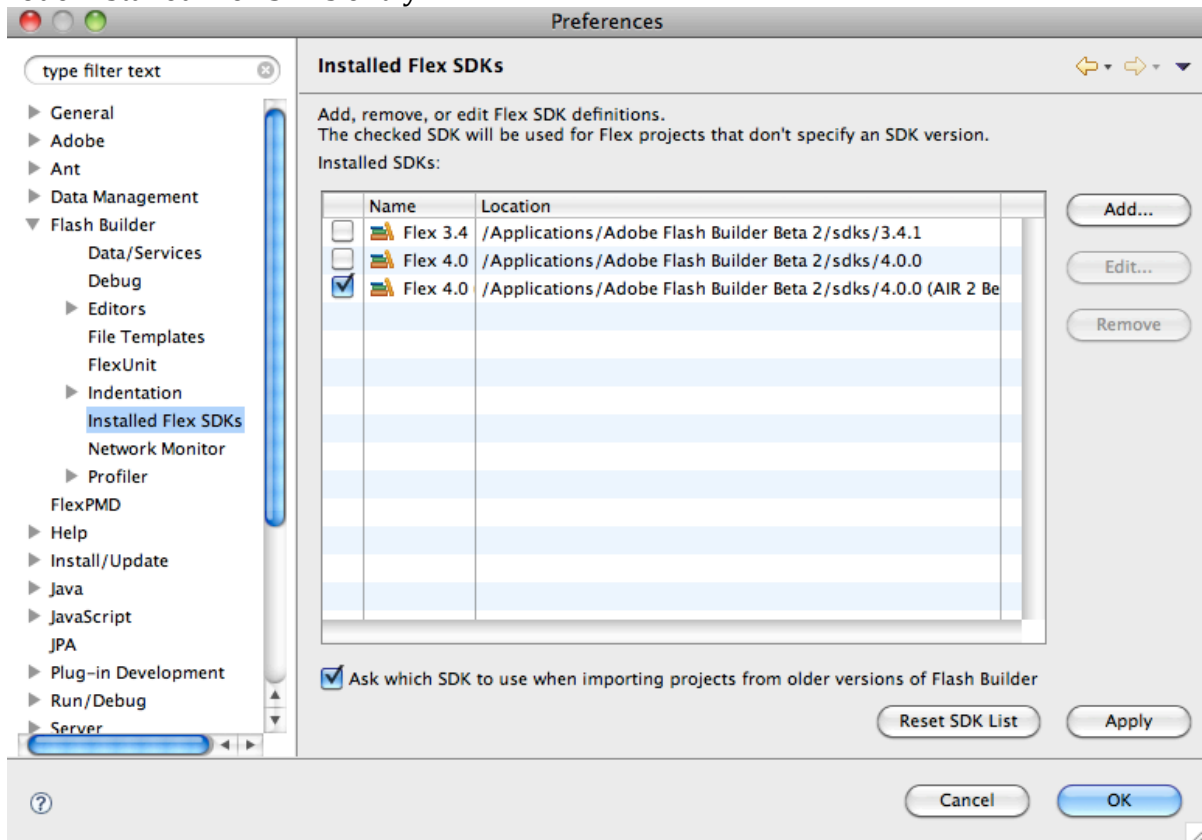
Pre-Requisites

Before you can take this course, you must have the following software installed on your computer:

- Adobe Flash Builder 4. You can download it from here: <http://labs.adobe.com/technologies/flashbuilder4/>
- Adobe AIR 2 runtime: <http://labs.adobe.com/technologies/air2/>
- AIR 2 SDK: <http://corlan.org/downloads/air2/>
- Additionally, you must download the projects archive (AttendeeProjects and CompletedProjects), AIR 2 SDK for your operating system, and a soft copy of this handout: <http://corlan.org/downloads/air2/>
- Download Tour de Flex application (it is AIR application that can help you learn Flex and AIR): <http://www.adobe.com/devnet/flex/tourdeflex/>

Adding the AIR SDK to your Flash Builder

1. Unzip the AIR SDK and copy into the installation folder of Flash Builder/sdks/
2. Open Flash Builder, go to Window > Preferences and select from the Flash Builder node Installed Flex SDKs entry:



3. Click on Add button and navigate to the AIR SDK folder you copied into the sdks. Select the newly added SDK to make it the default one and then click Apply and OK.

About the trainer

Mihai Corlan is a Platform Evangelist with Adobe Systems, part of the European team of evangelism. When he is not presenting at conferences/events across Europe, he is writing articles or code. You can follow him on Twitter @mcorlan or read his blog <http://corlan.org>.

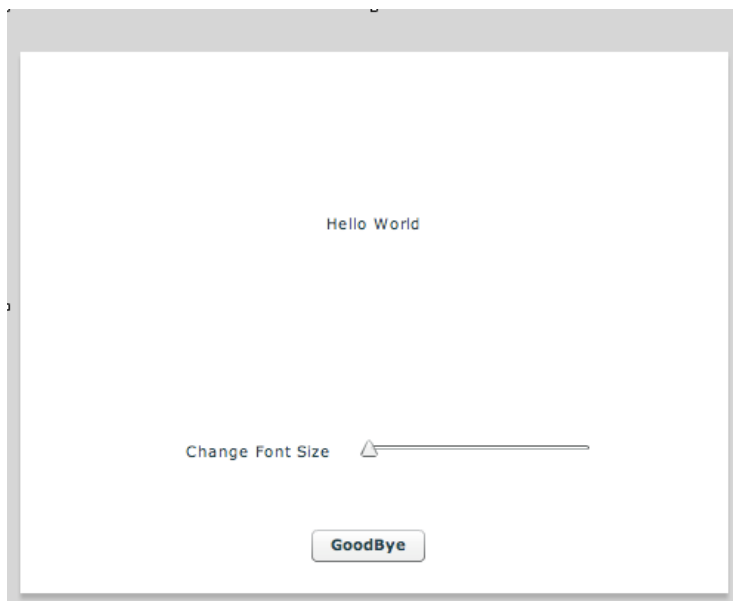
Project 1. Getting used with Flex Builder, design view, data binding, events and events listeners, MXML and ActionScript 3

We will have a look at:

- Component Exploration
- Working with Visual Controller Components (Slider, Button, etc)
- Binding data from a slider to a label
- Working with Layout Components (Panel, Window, Canvas etc)
- writing an app with both MXML and AS3
- writing a function
- Data binding – curly braces and the [Bindable] keyword
- Events – calling the function based on an event

Instructions:

1. Start a new AIR project and name it “AIRBootCamp1-HelloWorld”
2. In Design View, Drag a Panel into your project and resize it to around 500W * 400H
3. Add two labels (one saying “Hello World; the other “Change Font Size” and position them approximately as shown below. Add a button and label it “Goodbye” and a horizontal slider.



4. Switch to code view and give the following components names:

```
<s:Panel id="myPanel" x="0" y="0" width="500" height="400">  
  <s:Label id="myLabel" x="219" y="57" text="Hello World&#xd;" />  
  <s:HSlider id="mySlider" x="235" y="266" />  
</s:Panel>
```
5. Make the button work by adding the following line then run the application and hit the button.

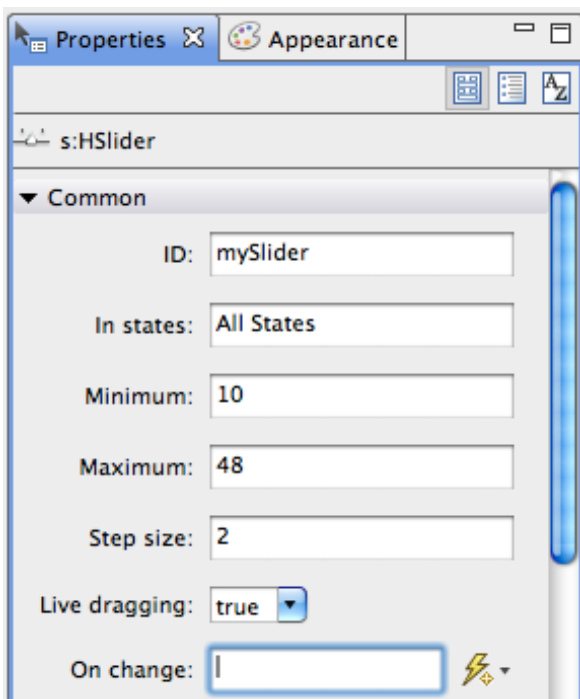
```
click="myPanel.visible=(false)"/>
```

6. Make a `<mx:Script>` block as shown below:

```
<fx:Script>
  <![CDATA[
    [Bindable]
    public var fontSize:Number = 12;

    public function changeText():void
    {
        fontSize = mySlider.value;
    }
  ]]>
</fx:Script>
```

7. In Design mode, select the horizontal slider and set the following properties:



8. Switch to Code view and add the following attribute within the Label that says Hello World. (Explain the curly brace syntax):

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

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

```
change="changeText()"
```

10. Run the application.

Solution:

```
<?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]
    public var fontSize:Number = 12;

    public function changeText():void
    {
        fontSize = mySlider.value;
    }
  ]]>
</fx:Script>

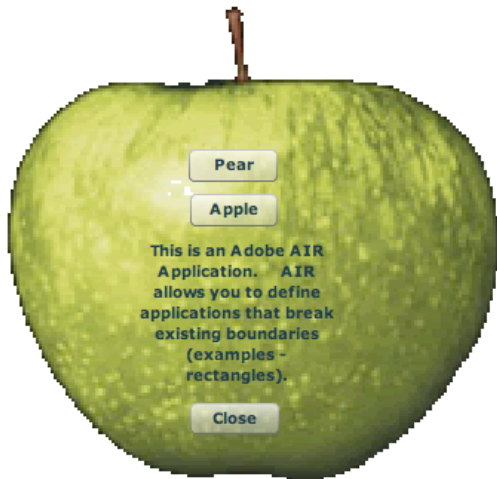
<fx:Declarations>
  <!-- Place non-visual elements (e.g., services, value objects) here -->
</fx:Declarations>
<s:Panel id="myPanel" x="0" y="0" width="500" height="400">
  <s:Label id="myLabel" x="219" y="57" text="Hello World&#xd;" fontSize="{fontSize}"/>
  <s:Label x="67" y="266" text="Change Font Size&#xd;"/>
  <s:HSlider id="mySlider" x="235" y="266" minimum="10" maximum="48" stepSize="2"
liveDragging="true" change="changeText()"/>
  <s:Button x="215" y="329" label="Goodbye" click="myPanel.visible=(false)"/>
</s:Panel>
</s:WindowedApplication>

```

Project 2. Chromeless applications

A thorough discussion on chromeless apps including how to make one from scratch. 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.

Discuss how you have to feed the “event” object to the event handler and why this is done. What values might people want to read from the event object?

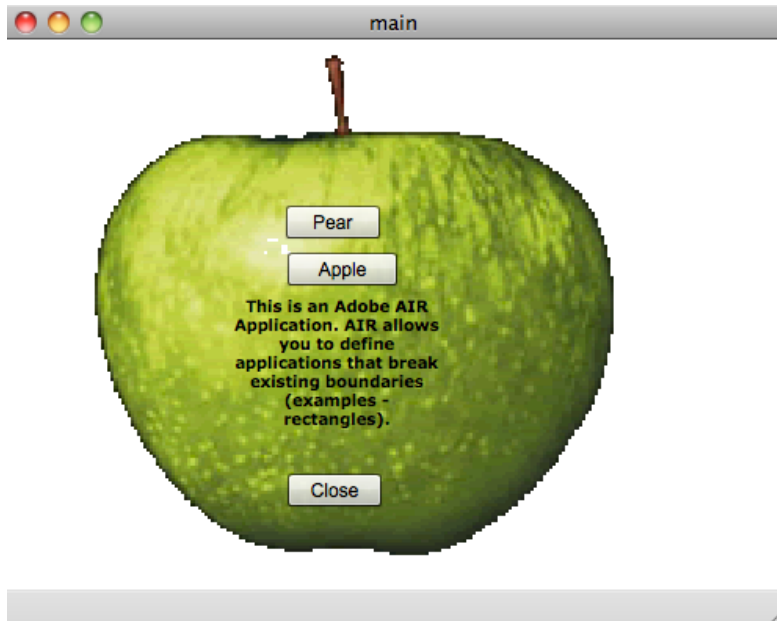


For this, attendees will start with a new project with two graphics – an apple and a pear with transparent backgrounds. Demo will first build the project, then take attendees through all the steps to making it chromeless. These include:

1. Modify the App Descriptor file (transparency=true, chrome=none) (describe what this does, what the file is for)
2. Adding backgroundAlpha="0"
3. How to add event handlers so you can move your app
4. How to replace the system chrome “close()” function

Instruction:

1. Import the project to Flex Builder from your attendee folder ~/attendees. The project name is AIRBootCamp2-TransparentApp.
2. Open the project and open both the main.mxml and main-app.mxml files.
3. Run the project to see what happens. Note that we will get rid of the chrome in the next steps. It should look like this:



4. In the main-app.mxml application descriptor file, find and modify these two lines of code by un-commenting them. Note: They are usually around line 45 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>
```

5. Save and close the application descriptor file. Then try running the application. Note the ugly side effect.
6. Add the following attributes to the Root Element of WindowedApplication

```
backgroundAlpha="0" showStatusBar="false"
```

7. Run the project. It should look good but we still cannot move it and the close button does not work. Discussion: Why?
8. 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:

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

and add this line to the <s:Button> with the Close label 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.

9. inside the `init()` function, add the following line of code:

```
mainPanel.addEventListener(MouseEvent.MOUSE_DOWN, startMove);
```

10. add a listener for the `creationComplete` event on `WindowedApplication` tag:
`creationComplete="init()"`

11. then add this function inside the `<fx:Script>` element:

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

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

Solution

```
<?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"
backgroundAlpha="0" showStatusBar="false" creationComplete="init()">
    <fx:Script>
        <![CDATA[

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

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

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

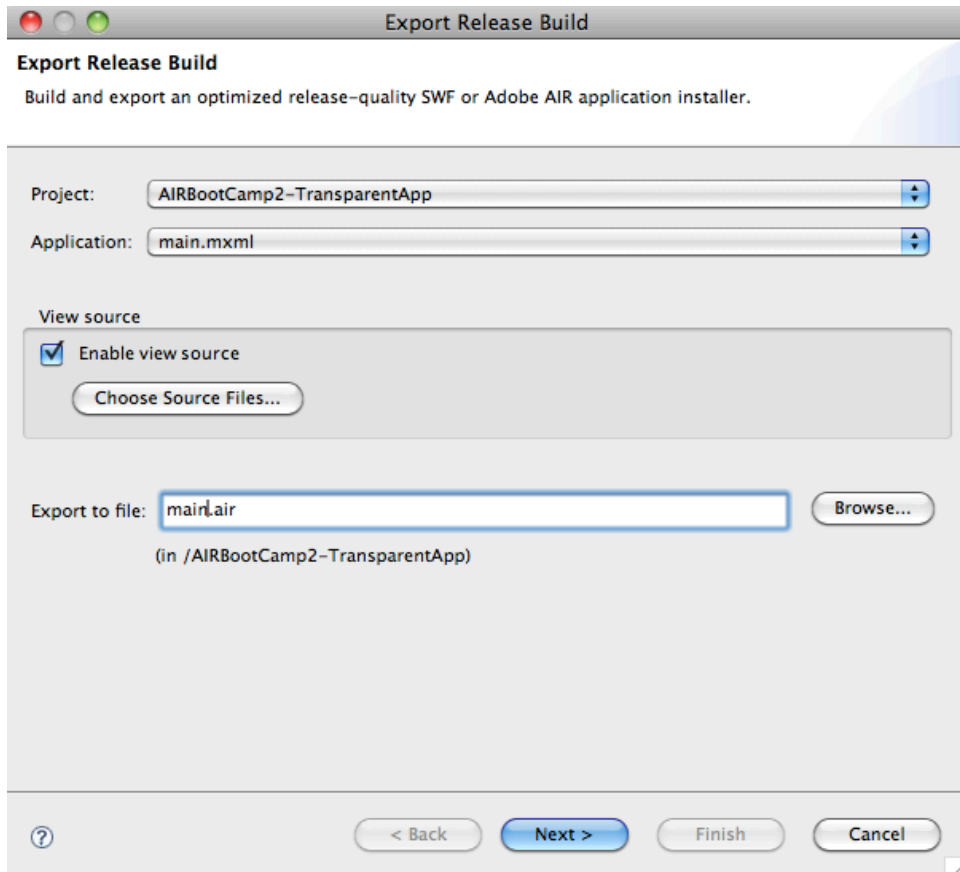
        ]]>
    </fx:Script>
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <mx:Image id="mainPanel" x="46" y="0" width="363" height="343" horizontalAlign="center"
        source="assets/apple.gif" scaleContent="true" autoLoad="true"/>

    <s:Button x="182" y="277" label="Close" width="60" height="21" click="closeApp()"/>
    <s:RichText x="145" y="166" text="This is an Adobe AIR Application. AIR allows you to define
applications that break existing boundaries (examples - rectangles)." width="137" height="121"
        fontFamily="Verdana" fontWeight="bold" fontSize="10" textAlign="center"/>
    <s:Button x="181" 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'"/>
</s:WindowedApplication>
```


Project 3. Exporting, Signing, Distributing and installing AIR applications

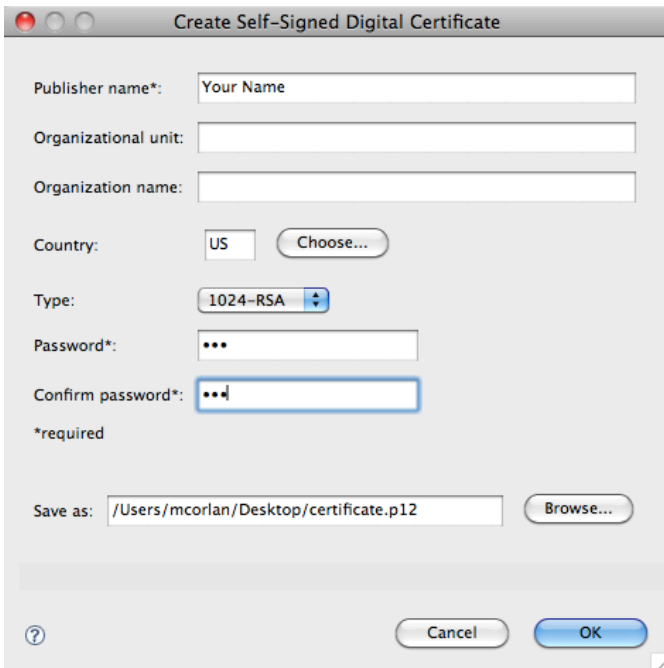
This exercise is to demonstrate how to export, sign and distribute AIR applications. Use the preceding project and take the following steps.

1. Click “File -> Export -> Release Build...” The following dialog will appear. Select your project and click “Next”.

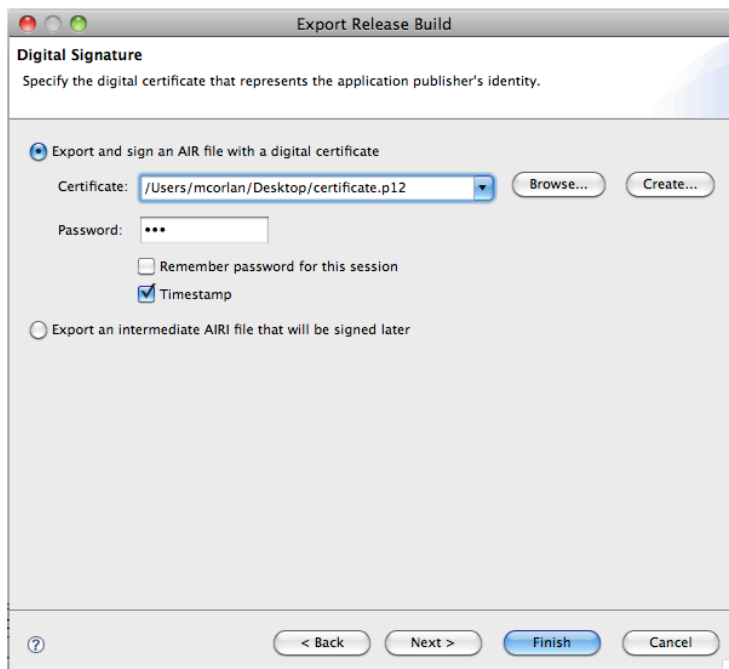


2. You can now sign your application. For production, you will want a real certificate registered with a CA working with the AIR runtime. For now, select “Create”. Fill out the following dialog box: Publisher name, password, and choose a location where the certificate will be created.

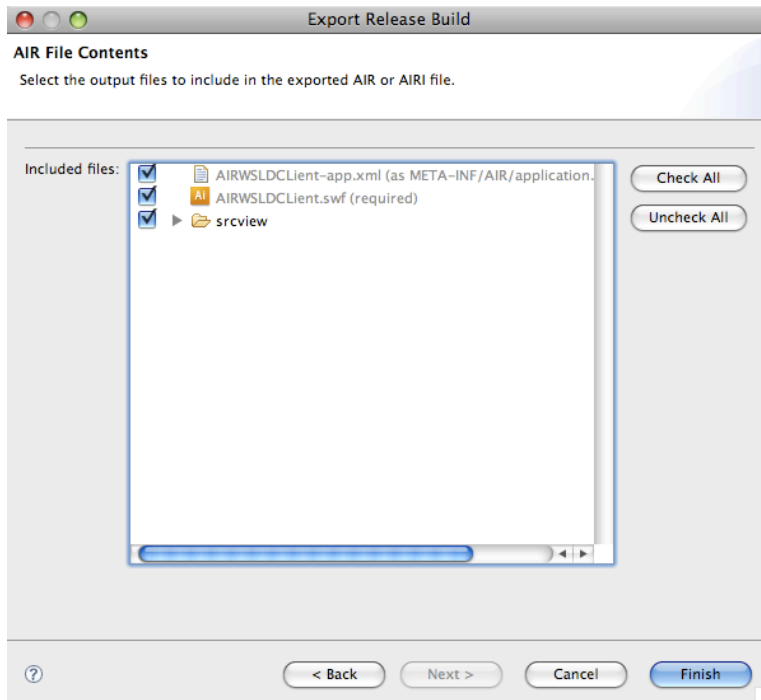
NOTE: Self signed certificates are basically worthless.



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



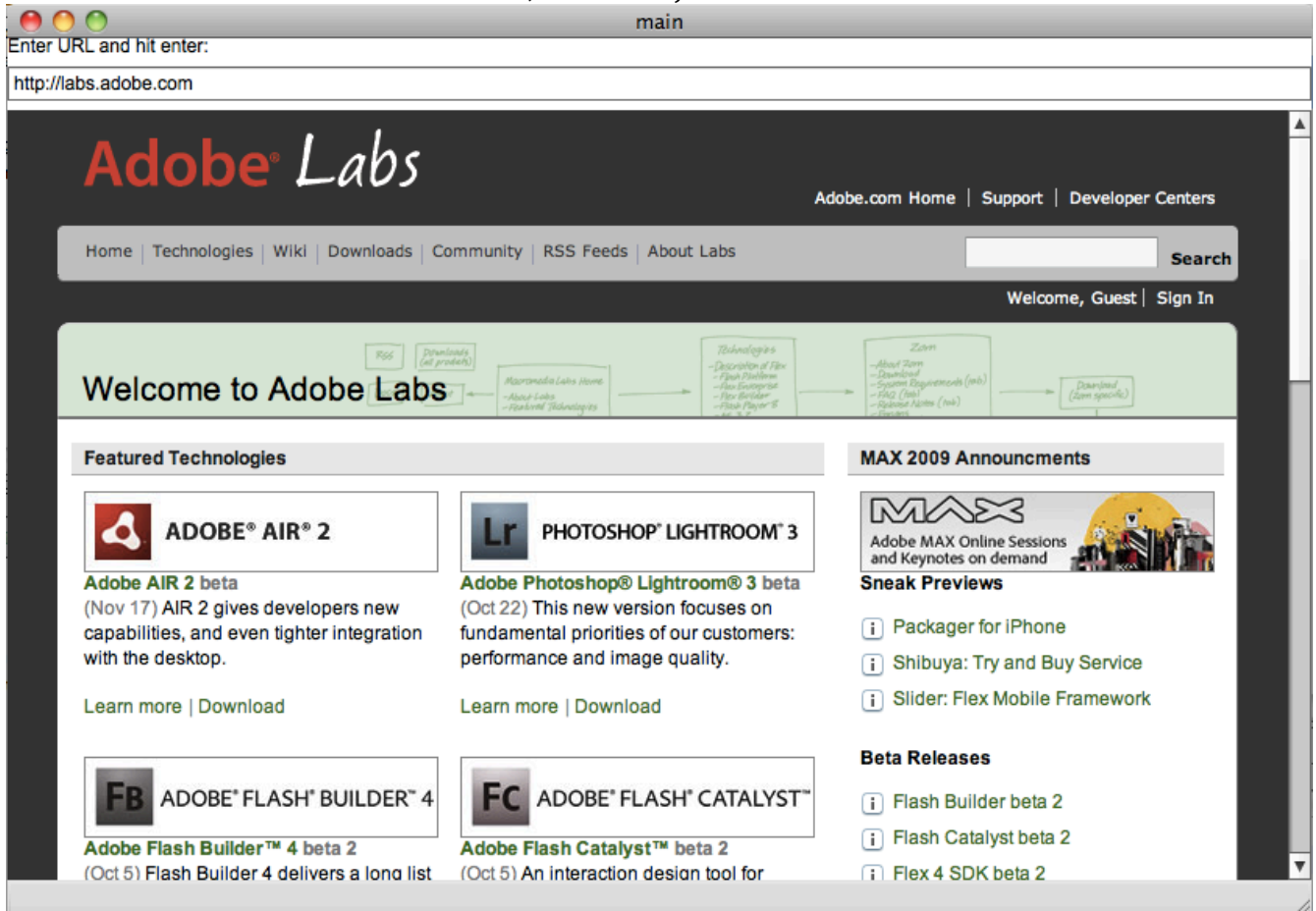
4. Select the parts you want to export and click "Finish"



5. The *.AIR file is now ready to be distributed. By default, it is create inside your project.

Project 4. HTML capabilities in AIR

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.



Instructions:

1. Create a new AIR project named AIRBootCamp4-HTML
2. Set the layout of the application vertical:

```
<s:layout>
    <s:VerticalLayout/>
</s:layout>
```
3. Add a label :

```
<s:Label text="Enter URL and hit enter:"/>
```
4. Add a TextInput control. Set the width 100%, and add this code for enter event:

```
html.location = urlTxt.text. Next add an URL for the text property.
<s:TextInput id="urlTxt" width="100%" enter="html.location = urlTxt.text"
text="http://labs.adobe.com" />
```
5. Add a HTML component and set html as id and 100% for width and height. Add a valid URL for the location attribute (this will be loaded by default when you run the app)

```
<mx:HTML id="html" width="100%" height="100%"
location="http://labs.adobe.com" />
```

Solution

```
<?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:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <s:Label text="Enter URL and hit enter:"/>
    <s:TextInput id="urlTxt" width="100%" enter="html.location = urlTxt.text"
text="http://labs.adobe.com" />
    <mx:HTML id="html" width="100%" height="100%" location="http://labs.adobe.com" />
</s:WindowedApplication>
```

Project 5. Reading and Writing Files to local hardisk

This exercise shows you how to write and read files to the local disk. When you want to read and write files you use an instance of File class to get access to an existen file or to create a new one. In order to read from the file or to write to file you use a FileStream object.

You will write to disk the content you type into a RichTextEditor component, and you will be able to open any file from your computer and display the content using the same RichTextEditr component.

For your convinience the File class offer quick access to some locations from the user machine by using the following static members:

- File.desktopDirectory – return a File object that points to the user desktop directory
- File.applicationDirctory – return a File object that points to the application installation folder
- File.applicationStorageDirectory - return a File object that points to the application storage folder. This folder is outside of the application, typically inside of the user documents folder.

There are also a number of out-of-the-box Flex components for browsing the file system such as FileSystemComboBox, FileSystemDataGrid, FileSystemList, and FileSystemTree. You can easily use these components in a Flex application such as:

```
<?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/haIo">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <mx:FileSystemDataGrid directory="{File.desktopDirectory}"/>
    <mx:FileSystemTree directory="{File.desktopDirectory}"/>
</s:WindowedApplication>
```

Instructions:

1. Import the project called AIRBootCamp5-ReadWriteFiles from AttendeeProjects
2. In the script block you will add the functions that handles the writing and reading of files to the local hardisk. First add the function that handles the Save button click event:

```
private function clickSave():void
{
    var f:File = File.desktopDirectory;
    f.browseForSave("Save As");
    f.addEventListener(Event.SELECT, saveData);
}
```

3. Create the saveData function. This function will be called once the user select the location where the file will be saved. Inside this function you will write the file to disk:

```
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();
    }

```

4. Now let's create the functions for loading a file from disk. First create the clickLoad() function. This will let you select the file.

```

private function clickLoad():void
{
    var f:File = File.desktopDirectory;
    f.browseForOpen("Select the file");
    f.addEventListener(Event.SELECT, fileSelected);
}

```

5. Create the fileSelected() function that is called once the user selects a file:

```

private function fileSelected(event:Event):void
{
    var stream:FileStream = new FileStream();
    stream.open(event.target as File, FileMode.READ);
    var fileData:String = stream.readUTFBytes(stream.bytesAvailable);
    rte.htmlText = fileData;
}

```

Solution:

```

<?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/haio">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <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();
            }

            private function clickSave():void
            {
                var f:File = File.desktopDirectory;
                f.browseForSave("Save As");
                f.addEventListener(Event.SELECT, saveData);
            }

            private function clickLoad():void
            {
                var f:File = File.desktopDirectory;
                f.browseForOpen("Select the file");
                f.addEventListener(Event.SELECT, fileSelected);
            }

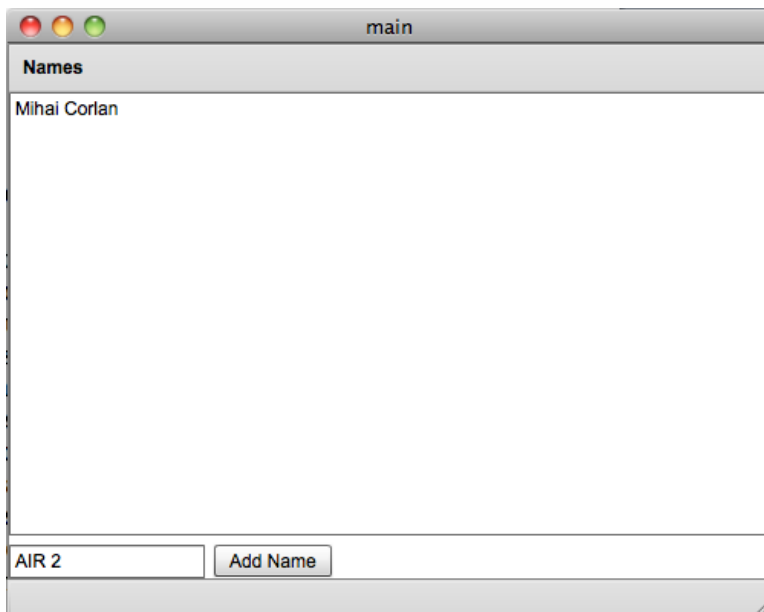
            private function fileSelected(event:Event):void
            {
                var stream:FileStream = new FileStream();
                stream.open(event.target as File, FileMode.READ);
                var fileData:String = stream.readUTFBytes(stream.bytesAvailable);

```

```
        rte.htmlText = fileData;
    }

    ]]>
</fx:Script>
<mx:RichTextEditor id="rte" width="100%" height="100%" title="Text Editor"/>
<s:Button label="Save" click="clickSave()" />
<s:Button label="Load file" click="clickLoad()" />
</s:WindowedApplication>
```

Project 6. Local database in AIR, 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.

Keep in mind that you can have the database encrypted. Here is the signature for SqlConnection open method:

```
public function open(reference:Object = null, openMode:String = "create",
    autoCompact:Boolean = false, pageSize:int = 1024, encryptionKey:ByteArray =
null):void
```

The encryption key is 16 bytes long. The database file is encrypted using AES.

Instructions:

1. Create a project called AIRBootCamp6-SQLite
2. First create the UI of the application. Add a Panel with VerticalLayout, inside the Panel add a List and a Group that holds a TextInput and a Button:

```
<s:Panel title="Names" width="100%" height="100%">
<s:layout>
    <s:VerticalLayout/>
</s:layout>
<s>List dataProvider="{names}" labelField="name" width="100%"
height="100%">/>
<s:Group>
    <s:layout>
        <s:HorizontalLayout/>
    </s:layout>
    <s:TextInput id="newName"/>
    <s:Button label="Add Name" click="addName()"/>
</s:Group>
</s:Panel>
```

3. Add a script block and inside of it declare two variables. sqlConn will hold the connection to the local SQLite database and names will hold the data retrieved from the database (an ArrayCollection of strings):

```
import mx.collections.ArrayCollection;
import flash.data.*;
```

```
private var sqlConn:SqlConnection;
[Bindable]
private var names:ArrayCollection;
```

4. Create the initApp() function that will be called once the application finished to load. In this function you initialize the connection to the database and create the table if it isn't already created:

```
private function initApp():void
{
    sqlConn = new SqlConnection();
    sqlConn.open(File.applicationStorageDirectory.resolvePath("my.db"));
    var stmt:SQLStatement = new SQLStatement();
    stmt.sqlConnection = sqlConn;
    stmt.text = "create table if not exists names (name string)";
    stmt.execute();
    queryNames();
}
```

- Register the `initApp` function on the `creationComplete` event of the application; add this to the `WindowedApplication` tag:
`creationComplete="initApp()"`
- Define the `queryNames()` function that will retrieve all the records from the table and add them to the `names` variable:

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

- Finally, add the function `addName()` that is called once the `Add Name` button is clicked:

```
private function addName():void
{
    var stmt:SQLStatement = new SQLStatement();
    stmt.sqlConnection = sqlConn;
    stmt.text = "insert into names (name) values (:name)";
    stmt.parameters[":name"] = newName.text;
    stmt.execute();
    queryNames();
}
```

Solution:

```
<?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"
    creationComplete="initApp()">
    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import flash.data.*;

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

            private function initApp():void
            {
                sqlConn = new SQLConnection();
                sqlConn.open(File.applicationStorageDirectory.resolvePath("my.db"));
                var stmt:SQLStatement = new SQLStatement();
                stmt.sqlConnection = sqlConn;
                stmt.text = "create table if not exists names (name string)";
                stmt.execute();
                queryNames();
            }

            private function queryNames():void
            {
                var stmt:SQLStatement = new SQLStatement();
```

```

        stmt.sqlConnection = sqlConn;
        stmt.text = "select * from names";
        stmt.execute();
        var result:SQLResult = stmt.getResult();
        names = new ArrayCollection(result.data);
    }

    private function addName():void {
        var stmt:SQLStatement = new SQLStatement();
        stmt.sqlConnection = sqlConn;
        stmt.text = "insert into names (name) values (:name)";
        stmt.parameters[":name"] = newName.text;
        stmt.execute();
        queryNames();
    }
    ]]>
</fx:Script>
<s:Panel title="Names" width="100%" height="100%">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <s>List dataProvider="{names}" labelField="name" width="100%" height="100%"/>
    <s:Group>
        <s:layout>
            <s:HorizontalLayout/>
        </s:layout>
        <s:TextInput id="newName"/>
        <s:Button label="Add Name" click="addName()"/>
    </s:Group>
</s:Panel>
</s:WindowedApplication>

```

Project 7. Serialize ActionScript objects to the local hardisk

Sometimes you want to save entire ActionScript objects to the local hardisk as part of your offline strategy, or just to speed up the loading type by using a local cache until you get the changes from the server.

Adobe AIR enables you to serialize ActionScript objects on the disk, and to read them back to your application.

You use a FileStream object, and its methods writeObject and readObject.

Instructions:

1. Create a project named AIRBootCamp7-Serialize
2. Create the UI of the application. You need a Panel to hold all the other components, next a Form to help you align a label, text input, and two buttons. The buttons itselfs will be hold by a Group with HorizontalAlign layout, thus the two buttons will be rendered on the same line.

```
<s:Panel title="Write/Read ActionScript Objects to/from files"
width="100%" height="100%">
    <mx:Form x="10" y="10">
        <mx:FormItem label="Subject:">
            <s:TextInput id="txt" width="257" maxChars="255"/>
        </mx:FormItem>
        <mx:FormItem>
            <s:Group>
                <s:layout>
                    <s:HorizontalLayout/>
                </s:layout>
                <s:Button label="Write" click="writeObject()"/>
                <s:Button label="Read" click="readObject()"/>
            </s:Group>
        </mx:FormItem>
    </mx:Form>
</s:Panel>
```

3. Add a script block and inside of it you will define the two functions called by the Write and Read buttons. The first function is writeObject(). Using a File object and a FileStream object, this function serialize the Object instace created using the text property of the text input component.

```
private function writeObject():void
{
    var object:Object = new Object();
    object.value = txt.text;
    var file:File =
    File.applicationStorageDirectory.resolvePath("myobject.file");
    if (file.exists)
        file.deleteFile();
    var fileStream:FileStream = new FileStream();
    fileStream.open(file, FileMode.WRITE);
    fileStream.writeObject(object);
}
```

```

        fileStream.close();
    }

```

4. Now, add the readObject() function. First it checks that a file exists (an object was serialized before), next it reads the content of the file into an object (you use readObject() method of the FileStream class):

```

private function readObject():void
{
    var file:File =
    File.applicationStorageDirectory.resolvePath("myobject.file");
    if (!file.exists) {
        Alert.show("There is no object saved!");
        return;
    }
    var fileStream:FileStream = new FileStream();
    fileStream.open(file, FileMode.READ);
    var object:Object = fileStream.readObject();
    Alert.show("The text member has this value: " + object.value);
}

```

Solution:

```

<?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;

            private function writeObject():void
            {
                var object:Object = new Object();
                object.value = txt.text;
                var file:File =
File.applicationStorageDirectory.resolvePath("myobject.file");
                if (file.exists)
                    file.deleteFile();
                var fileStream:FileStream = new FileStream();
                fileStream.open(file, FileMode.WRITE);
                fileStream.writeObject(object);
                fileStream.close();
            }

            private function readObject():void
            {
                var file:File =
File.applicationStorageDirectory.resolvePath("myobject.file");
                if (!file.exists) {
                    Alert.show("There is no object saved!");
                    return;
                }
                var fileStream:FileStream = new FileStream();
                fileStream.open(file, FileMode.READ);
                var object:Object =
                    Alert.show("The text member has this value: " +
object.value);
            }
        ]]>
    </fx:Script>
    <s:Panel title="Write/Read ActionScript Objects to/from files" width="100%" height="100%">
        <mx:Form x="10" y="10">
            <mx:FormItem label="Subject:">

```

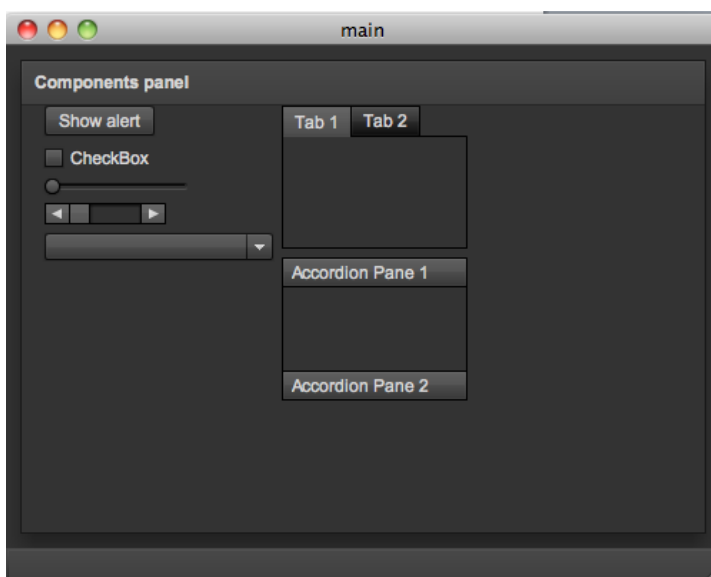
```
        <s:TextInput id="txt" width="257" maxChars="255"/>
    </mx:FormItem>
    <mx:FormItem>
        <s:Group>
            <s:layout>
                <s:HorizontalLayout/>
            </s:layout>
            <s:Button label="Write" click="writeObject()"/>
            <s:Button label="Read" click="readObject()"/>
        </s:Group>
    </mx:FormItem>
</mx:Form>
</s:Panel>
</s:WindowedApplication>
```

Project 8. Using CSS

You can customize the appearance of a an AIR application by using two methods: skinning (creating skins for the components using ActionScript and FXG) or using CSS (Cascading Style Sheet). In this exercise you will learn how to use CSS to change the default look.

Instructions:

1. Import AIRBootCamp8-CSS project from AttendeeProjects.
2. Run the project and look at the components style
3. Uncomment the `<fx:Style.../>` tag and run again the application. Note how the look has changed.



- **base-color** (baseColor) – The main color for a component
- **content-background-color** (contentBackgroundColor) – Color of the fill of an item renderer
- **symbol-color** (symbolColor) – Color of any symbol of a component. Examples include the check mark of a CheckBox or the arrow of a scroll button
- **selection-color** (selectionColor) – The color of text when the component is enabled and has focus
- **focus-color** (focusColor) – Color of focus ring when the component is in focus
- **roll-over-color** (rollOverColor) – Color of the highlights when the mouse is over the component
- **color** – Color of the text
- **text-roll-over-color** (textRollOverColor) – Color of the text highlights when the mouse is over the component, used in old MX components.
- **accent-color** (accentColor) – Additional color used for accent. This color is used by “emphasized” buttons (default button in Alert box), and the slider track highlight.

Project 9. Using Encrypted Local Store (ELS)

If you have sensitive data, such as user personal data, passwords that you want to store locally, then Encrypted Local Store is perfect for this job:

- uses DPAPI on Windows, and KeyChain
- uses AES 128-bit encryption
- one store per application and user account
- by default the ELS is bound to the application and user account; you can bind it to the application bits (if the bits are changed, the information are lost)
- it acts like a hashmap: you use a key and assign the value, you get a value by using the key:

```
EncryptedLocalStore.setItem("myEncryptedData", data);  
var bytes:ByteArray = EncryptedLocalStore.getItem("myEncryptedData");
```

Instructions:

1. Import the project called AIRBootCamp9-ELS from AttendeeProjects
2. You'll have to create the functions for encrypt (called when the user click on Encrypt button) and decrypt (called by the Decrypt button)
3. When you want to store something into ELS you use a ByteArray to write into the values you want to store, and then you store the ByteArray to ELS. Add this code to the encrypt function:

```
var data:ByteArray = new ByteArray();  
data.writeUTFBytes(encryptText.text);  
EncryptedLocalStore.setItem("myEncryptedData", data);
```

4. Next define the decrypt function:

```
var bytes:ByteArray = EncryptedLocalStore.getItem("myEncryptedData");  
if (bytes) {  
    Alert.show(bytes.readUTFBytes(bytes.bytesAvailable));  
} else {  
    Alert.show("There is no value set!");  
}
```

Solution:

```
<?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;  
  
            private function encrypt():void {  
                var data:ByteArray = new ByteArray();  
                data.writeUTFBytes(encryptText.text);  
                EncryptedLocalStore.setItem("myEncryptedData", data);  
            }  
  
            private function decrypt():void {  
                var bytes:ByteArray =
```

```

EncryptedLocalStore.getItem("myEncryptedData");
    if (bytes) {
        Alert.show(bytes.readUTFBytes(bytes.bytesAvailable));
    } else {
        Alert.show("There is no value set!");
    }
}
]]>
</fx:Script>

<s:Panel title="Store ActionScript objects in the encrypted local store" width="100%"
height="100%">
    <s:Group left="10" top="10">
        <s:layout>
            <s:VerticalLayout/>
        </s:layout>
        <s:TextInput id="encryptText" />
        <s:Button label="Encrypt" click="encrypt()" />
        <s:Button label="Decrypt" click="decrypt()" />
    </s:Group>
</s:Panel>

</s:WindowedApplication>

```

Project 10. Talking to a server: REST style

In this exercise you will use PHP as a server side technology. Flash Builder has wizards to assist you while connecting a frontend to data from a server.

You will use HTTPService option from Data/Services wizard, this option let you consume any REST style service you might have on your server. However, in order to be able to connect from your AIR application to a server script, you need to place cross-domain.xml policy file on the server. You can read more about this here:

http://livedocs.adobe.com/flash/8/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00001621.html

This file looks like this:

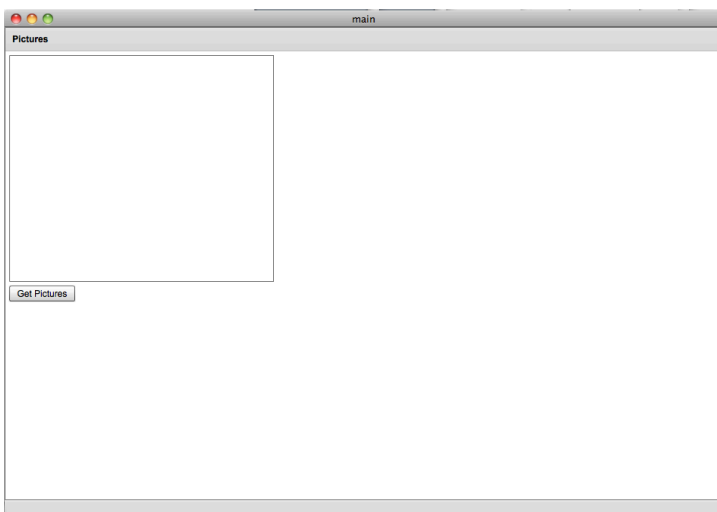
```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <allow-access-from domain="*" />
  <site-control permitted-cross-domain-policies="master-only"/>
</cross-domain-policy>
```

The PHP script you'll gonna use is at this URL

<http://corlan.org/downloads/rest/getpictures.php> and outputs an XML like this:

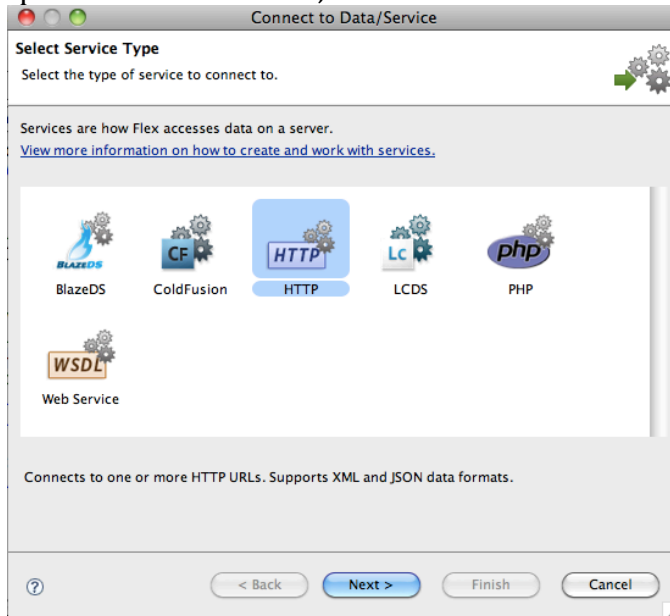
```
<?xml version="1.0" encoding="utf-8"?>
<data>
  <picture>
    <small>http://farm4.static.flickr.com/3260/3231547573_3fdafd9407_s.jpg</small>
    <big>http://farm4.static.flickr.com/3260/3231547573_3fdafd9407.jpg</big>
    <name>Greek Island</name>
  </picture>
  <picture>
    <small>http://farm4.static.flickr.com/3520/3936812128_ffc4202c7b_s.jpg</small>
    <big>http://farm4.static.flickr.com/3520/3936812128_ffc4202c7b.jpg</big>
    <name>China - Forbidden City</name>
  </picture>
</data>
```

You will use this XML to populate a list and when an item from the list is selected, a bigger image is displayed.

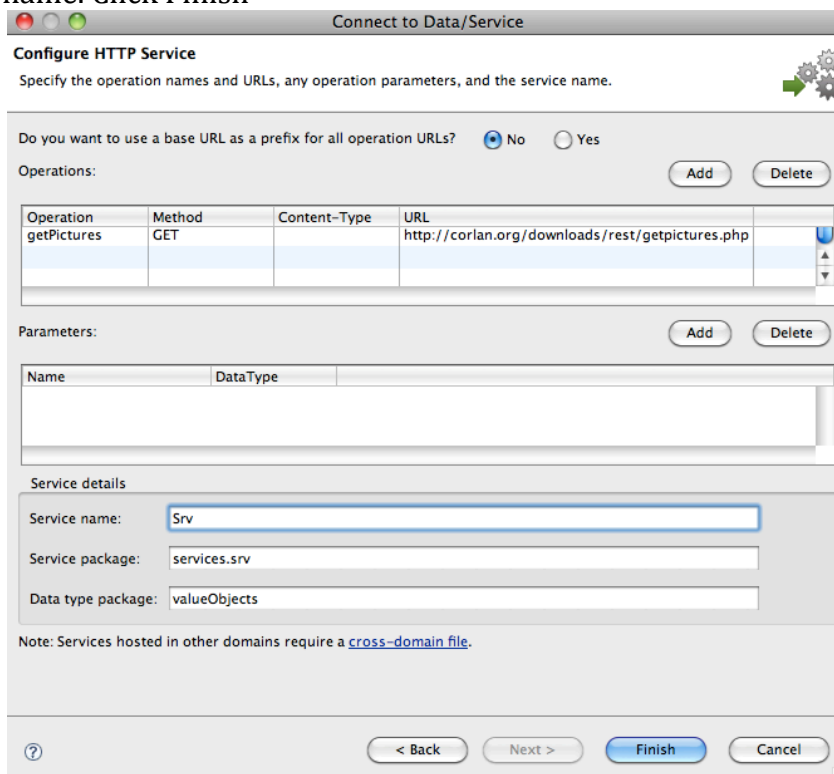


Instructions:

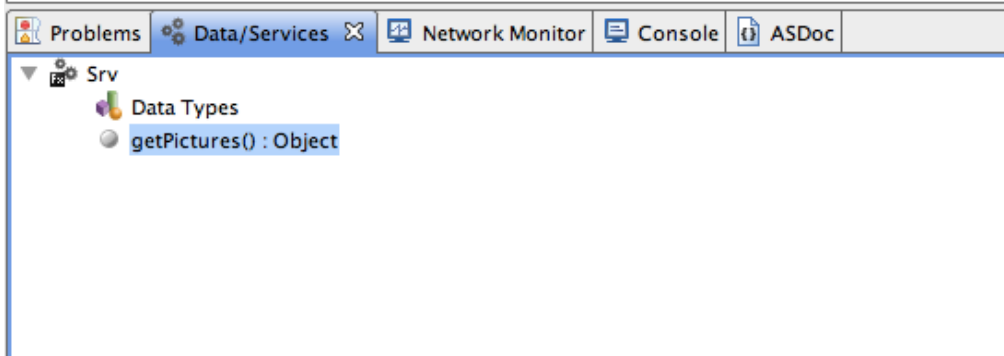
1. Import AIRBootCamp10-REST project from AttendeeProjects folder
2. The UI is already created, you need only to create the data service and to bind the data to the UI
3. Make sure you see the Data/Services view. If not, go to Window > Other Views... and select from the Flash Builder node Data/Services entry
4. Click on Connect to Data/Service... link from Data/Services view and select HTTP option from the wizard, then click Next



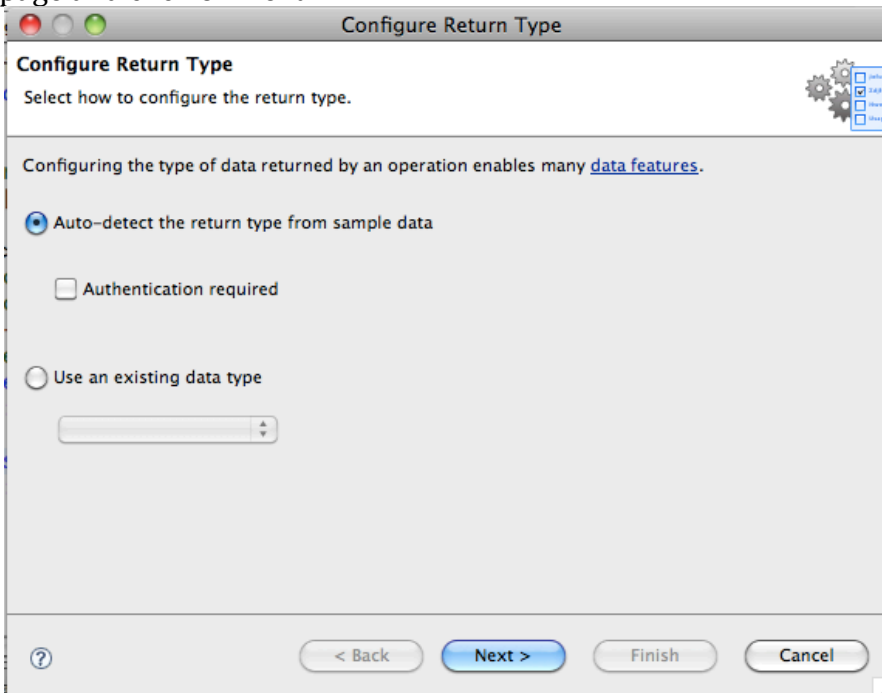
5. Set getPictures for Operation value and <http://corlan.org/downloads/rest/getpictures.php> for URL. Then set Srv as the service name. Click Finish



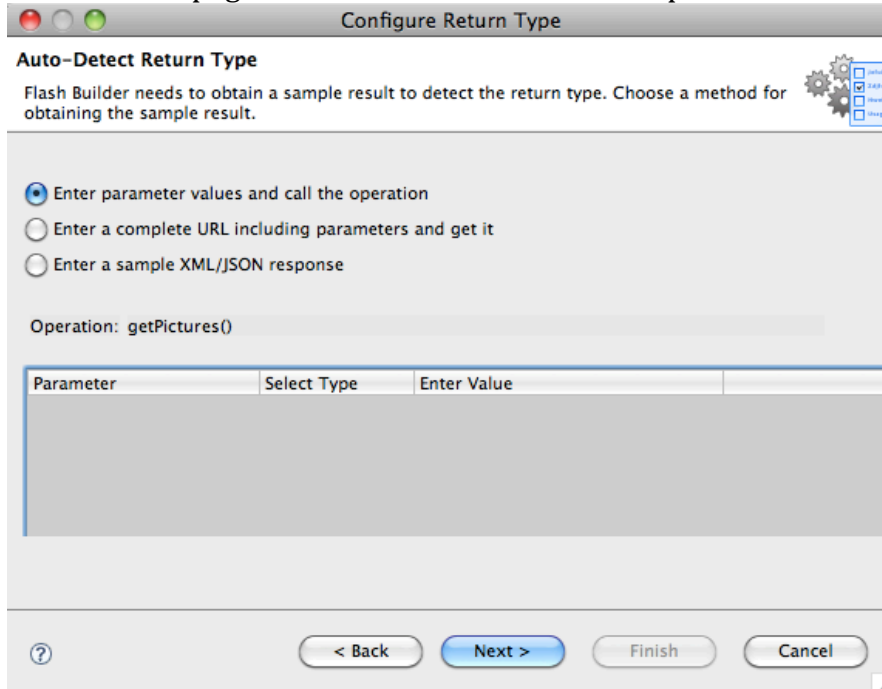
- You should see in the project source folder one new package (services.srv) and the service in the Data/Services view



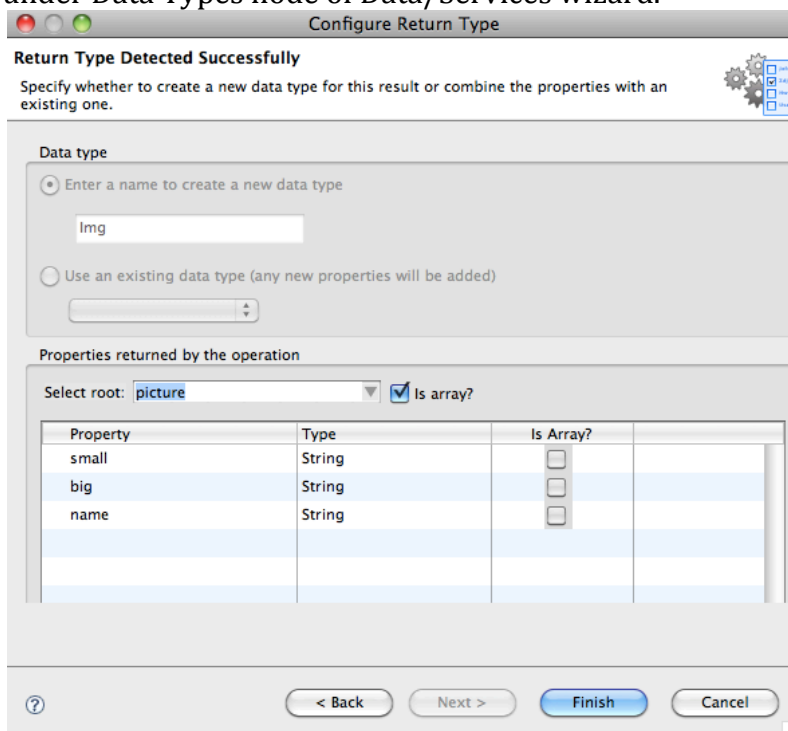
- Right-click on the getPictures() operation from the Data/Services view and choose Configure Return Type option. A wizard opens. Leave the default options in the first page and click on Next



- In the second page of the wizard select the first option and click Next

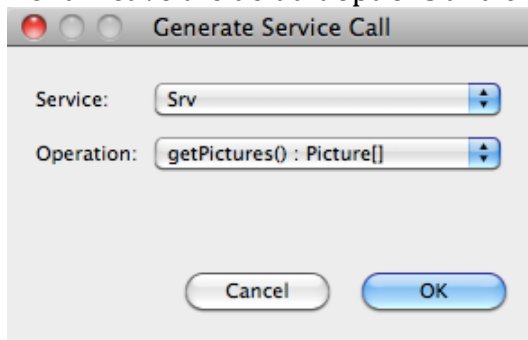


- Set the name of the data type to `Img` and select picture for Select root combo box. Make sure that Is array? option is checked. Click Finish. Now you should have a Picture node under Data Types node of Data/Services wizard.

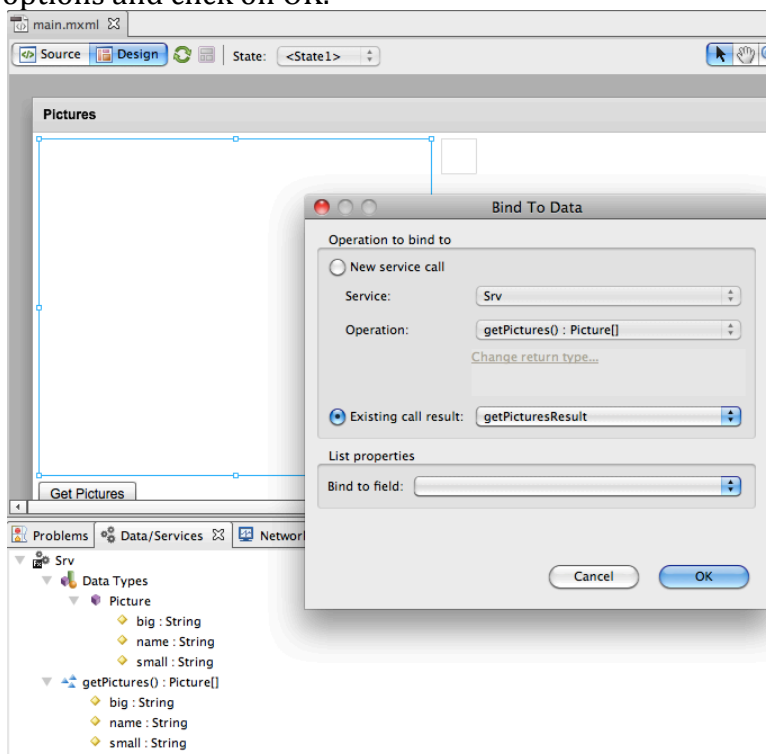


- It is time to make a call to the operation defined earlier. For this go to Design view, select the button and right-click. Select Generate Service Call... from the contextual

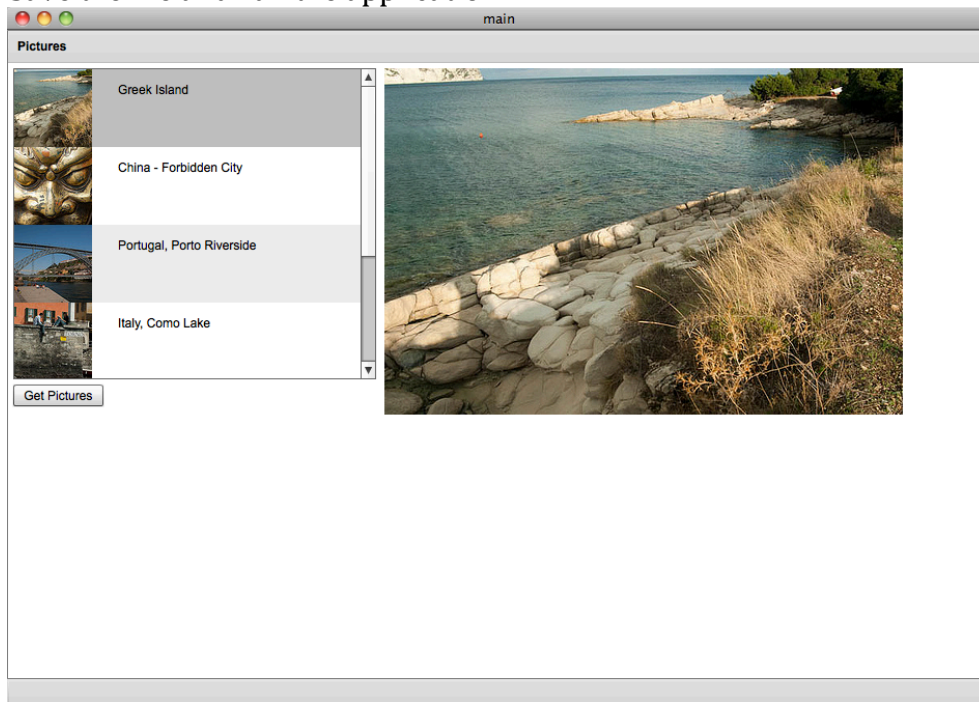
menu. Leave the default options and click OK



11. Now let's bind the result of the operation to the list. While being in Design view, right-click on the list and select Bind to Data... from the contextual menu. Let the default options and click on OK.



12. Save the file and run the application.



Solution:

See AIRBootCamp10-REST from CompletedProjects folder.

Project 11. Window API's

This project will demonstrate how to open and close another window using the windowing API's.

In AIR you can create additional windows using either `flash.display.NativeWindow` or `mx.core.Window`. The latter one is a wrapper over `NativeWindow` and it allows you to easily add Flex components.

Here are some examples of controlling the Window:

```
window.open();
window.alwaysInFront = true;
window.close();
window.maximize();
window.minimize();
window.orderInBackOf(otherWin);
window.orderInFrontOf(otherWin);
window.orderToBack();
window.orderToFront();
window.restore();
```

Please note that some of the Window properties are read-only after the Window/`NativeWindow` was initialized: `maximizable`, `minimizable`, `resizable`, `systemChrome`, `transparent`, `type`.

Instructions:

1. Import the AIR project named AIRBootCamp11-Window from AttendeeProjects
2. Run the project and play with the buttons
3. Look at the code

Solution:

```
<?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[
            import spark.components.Button;
            import mx.core.Window;

            private var newWindow:Window;

            private function createWindow():void
            {
                if (newWindow && !newWindow.closed)
                    return;
                newWindow = new Window();
                newWindow.title = "My Window";
                newWindow.width = 200;
                newWindow.height = 200;
                newWindow.open();
            }

            private function addButton():void
```

```

    {
        if (!newWindow)
            return;
        var b:spark.components.Button = new Button();
        b.label = "My Label";
        newWindow.addChild(b);
    }

    ]]>
</fx:Script>
<s:Button label="Open New Window" click="createWindow()"/>
<s:Button label="Always in Front" click="newWindow.alwaysInFront = true"/>
<s:Button label="Maximize" click="newWindow.maximize()"/>
<s:Button label="Close the Window" click="newWindow.close()"/>
<s:Button label="Add a button to the Window" click="addButton()"/>

</s:WindowedApplication>

```

Project 12. Volume Storage Detection

AIR 2 brought a series of new features that makes the integration of AIR applications with the underlying operating system even smoother. You can detect volume storage mounting and unmounting, you can open a file with the default application, you can create serve sockets and use UDP sockets, you can start and communicate with native processes. In the next three exercises you will see some of these features in action.

Instructions:

1. Import the AIRBootCamp12-VolumeStorage project from AttendeeProjects
2. There is a FileTree component that you'll use to display the content of a folder or of volume disk, a label for displaying the name of the current volume
3. Start with adding the content of the init() function. This function will be called once the application is initialized. You use StorageVolumeInfo.storageVolumeInfo property to register one listener for mount and unmount events. Next the default system volume is saved to a class variable

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME_MOUNT, onVolumeMount);
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME_UNMOUNT, onVolumeUnmount);
f = fileTree.directory;
lbl.text = fileTree.directory.name;
```
4. Next add the code to onVolumeMount() function. From the event object (StorageVolumeChangeEvent) you can get the name and path of the new volume storage

```
fileTree.directory = e.storageVolume.rootDirectory;
lbl.text = e.storageVolume.name;
```
5. Finally, add the code to onVolumeUnmount() function. You just restore the default volume path and name

```
fileTree.directory = f;
lbl.text = f.name;
```

Solution:

```
<?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"
creationComplete="init()">
    <fx:Script>
        <![CDATA[
            private var f:File;

            private function init():void
            {

                StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME
_MOUNT, onVolumeMount);

                StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME
_UNMOUNT, onVolumeUnmount);

                f = fileTree.directory;
                lbl.text = fileTree.directory.name;
            }
        ]]>
    </fx:Script>
</s:WindowedApplication>
```

```
private function onVolumeMount(e:StorageVolumeChangeEvent):void
{
    fileTree.directory = e.storageVolume.rootDirectory;
    lbl.text = e.storageVolume.name;
}

private function onVolumeUnmount(e:StorageVolumeChangeEvent):void
{
    fileTree.directory = f;
    lbl.text = f.name;
}
]]>
</fx:Script>

<s:Label id="lbl" x="10" y="10"/>

<mx:FileSystemTree id="fileTree" x="10" y="40"/>

</s:WindowedApplication>
```

Project 13. Open a file with Default Application

Starting with AIR 2 you can delegate to the operating system to handle a file with the default application registered on that system. For example if you have a File that points toward a image, you can ask the OS to display the image using the default application.

Instructions:

1. Import AIRBootCamp13-OpenWithDefault project from AttendeeProjects
2. The UI is already created, you have to add the content of the two functions. The FileSystemTree component displays the content of the desktop folder
3. First add the code for init() function. Set the directory property of the FileSystemTree to the desktop folder:

```
fileTree.directory = File.desktopDirectory;
```

4. Then add the code for the itemSelected() function. If the user selected a file and not a folder, then you open that file with the default application:

```
var f:File = fileTree.selectedItem as File;
if (f.isDirectory)
    return;
f.openWithDefaultApplication();
```

Solution:

```
<?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/ha1o" >

    <fx:Script>
        <![CDATA[
            private function init():void
            {
                fileTree.directory = File.desktopDirectory;
            }

            private function itemSelected(e:Event):void
            {
                var f:File = fileTree.selectedItem as File;
                if (f.isDirectory)
                    return;
                f.openWithDefaultApplication();
            }
        ]]>
    </fx:Script>
    <s:Label text="Desktop Folder Content:" x="10" y="10"/>

    <mx:FileSystemTree id="fileTree" directory="{File.desktopDirectory}" x="10" y="40"
height="300" width="400" change="itemSelected(event)"/>
</s:WindowedApplication>
```

Project 14. Native Processes

Starting with AIR 2 you have the ability to start a native process and you use standard input and output to communicate with the process. Of course you can kill the process when you want. You can start a Java program, you can call any executable from the local machine.

However, there is a trade off when you use NativeProcesses: you can not pack and distribute your application as an AIR file. Instead, you have to create native installers using the operating system for which you want to create the installer (you will create an EXE file on Windows and DMG on Mac).

In order to enable Native Processes feature, you have to add to the application descriptor file this node (at runtime you can check if this feature is enabled by checking this NativeProcess.isSupported):

```
<supportedProfiles>extendedDesktop</supportedProfiles>
```

Instructions:

1. Import the project AIRBootCamp14-NativeProcesses from AttendeeProjects
2. If you are on Windows, the code should work as it is. Run the app and select the ipconfig.exe executable from your machine. Then click on Run button. You should see the output of this command as it'd appear into a terminal window.
3. If you are on a Mac you have to comment the lines bellow the the comments for windows and uncomment the lines for Mac. On a Mac you use the command say.

Solution:

```
<?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="730"
height="380">
  <fx:Script>
    <![CDATA[
      import flash.events.Event;
      import mx.controls.Alert;

      //for Mac
      private var path:String = "";
      //for Windows
      private var path:String = "C:\\WINDOWS\\system32\\ipconfig.exe";
      private var nativeProcess:NativeProcess;
      private var processBuffer:ByteArray;

      private function onClick():void {
        var f:File = new File();
        f.addEventListener(Event.SELECT, onFileSelection);
        f.browse();
      }

      private function onFileSelection(e:Event):void {
        path = (e.target as File).nativePath;
        txtPath.text = path;
      }

      private function runCommand():void {
        var npInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
        //for Windows
        npInfo.executable = new File(path);
        //for Mac

```

```

//      npInfo.executable = new File("/usr/bin/say");
//      var args:Vector.<String> = new Vector.<String>;
//      //for Mac
//      args.push("Hello there");
//      npInfo.arguments = args;
//      processBuffer = new ByteArray();
//      nativeProcess = new NativeProcess();
//      nativeProcess.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA,
onStandardOutputData);
//      //this.nativeProcess.addEventListener(Event.STANDARD_OUTPUT_CLOSE,
onStandardOutputClose);
//      nativeProcess.addEventListener(NativeProcessExitEvent.EXIT,
onStandardOutputClose);
//      nativeProcess.addEventListener(ProgressEvent.STANDARD_ERROR_DATA,
onIOError);
//      nativeProcess.addEventListener(IOErrorEvent.STANDARD_OUTPUT_IO_ERROR,
onIOError);
//      nativeProcess.addEventListener(IOErrorEvent.STANDARD_ERROR_IO_ERROR,
onIOError);
//      nativeProcess.addEventListener(IOErrorEvent.STANDARD_INPUT_IO_ERROR,
onIOError);
//      nativeProcess.start(npInfo);
    }

    private function onStandardOutputData(e:ProgressEvent):void {
processBuffer.length);
        nativeProcess.standardOutput.readBytes(processBuffer,
    }

    private function onStandardOutputClose(e:Event):void {
        txtOut.text = new String(processBuffer);
    }

    private function onIOError(e:Event):void {
        Alert.show("STANDARD_OUTPUT_IO_ERROR: " +e.toString());
    }
]]>
</fx:Script>

<s:Button x="345" y="34" label="Browse..." click="onClick()"/>
<s:Label x="10" y="10" text="Path to ipconfig.exe:"/>
<s:TextInput id="txtPath" text="C:\WINDOWS\system32\ipconfig.exe" x="10" y="34"
width="327"/>
<s:TextArea id="txtOut" x="10" y="74" width="697" height="274"/>
<s:Button x="637" y="34" label="Run" click="runCommand()"/>

</s:WindowedApplication>

```

Project 15. Drag and Drop

Adobe AIR lets you integrate your application with the underlying OS by allowing drag & drop gestures. You can enable drag & drop gestures for components inside of the same application too. Like for example you have two UI components, and you want to drag an item from one to another one.

In this example you will explore how to handle the dragging of an existent file on an AIR application, and how to drag some text from the AIR application to a browser or text editor.

The following kinds of data can be transferred:

- Bitmaps
- Files
- Text
- URL strings
- Serialized objects
- Object references (valid only within the originating application)

Instructions:

1. Import the AIR project called AIRBootCamp14-DragDrop
2. Drag the Button to a text editor
3. Drag a text file from your computer into the application

Solution:

```
<?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"
creationComplete="onCreationComplete()">
    <fx:Script>
        <![CDATA[
            import flash.desktop.ClipboardFormats;
            import flash.events.NativeDragEvent;
            import flash.filesystem.File;
            import flash.filesystem.FileMode;
            import flash.filesystem.FileStream;

            private function onCreationComplete():void
            {
                addEventListener(NativeDragEvent.NATIVE_DRAG_ENTER, onDragIn);
                addEventListener(NativeDragEvent.NATIVE_DRAG_DROP, onDragDrop);
            }

            private function onDragIn(e:NativeDragEvent):void
            {
                if(e.clipboard.hasFormat(ClipboardFormats.FILE_LIST_FORMAT)) {
                    var files:Array =
e.clipboard.getData(ClipboardFormats.FILE_LIST_FORMAT) as Array;
                    if(files.length == 1) {
                        NativeDragManager.acceptDragDrop(this);
                    }
                }
            }

            private function onDragDrop(e:NativeDragEvent):void
```

```

    {
        var arr:Array = e.clipboard.getData(ClipboardFormats.FILE_LIST_FORMAT) as
Array;

        var f:File = File(arr[0]);
        var fs:FileStream = new FileStream();
        fs.open(f, FileMode.READ);
        var data:String = fs.readUTFBytes(fs.bytesAvailable);
        fs.close();
        outputField.text = data;
    }

    private function handle_imageClick():void
    {
        var clipboard:Clipboard = new Clipboard();
        clipboard.setData(ClipboardFormats.TEXT_FORMAT, "This Adobe AIR Boot Camp!
Woohooo!");

        var dragImage:BitmapData = new BitmapData(dragButton.width,
dragButton.height);
        dragImage.draw(dragButton);
        var offset:Point = new Point(dragButton.mouseX * -1, dragButton.mouseY * -
1);

        NativeDragManager.doDrag(dragButton, clipboard, dragImage, offset);
    }
}]]>
</fx:Script>

<s:TextArea top="10" right="10" bottom="10" left="251" id="outputField" />
<s:RichText text="Drag a Text File into the Application" width="233" height="148" top="11"
left="10"/>
<s:Button id="dragButton" label="Drag Me!" mouseDown="handle_imageClick()" top="50"
left="10"/>
</s:WindowedApplication>

```