Rich Internet Applications with Adobe® Flex and Java™ Technology

Tony Constantinides,
CEO
Constant Innovations Inc

Java

Sun microsystems

Using Adobe Flex and Java technology together to build next-generation Rich Internet Applications

GOAL

Agenda

- Introducing Adobe Flex
- Why Flex
- Streaming video and audio
- Changing look and feel at runtime (Skinning)
- Adding special effects
- Specialized Flex supported backend
- Flex and Java technology working together
- Asynchronous Messaging
- Using the Remoting Service
- Using the Flex Compiler API
- Summary and Q&A

How to get it?

- Download the free Flex 3 SDK from http://flex.org/download/
- Download the AIR SDK at http://www.adobe.com/products/air/tools/sdk
- Install the Flash debug runtime from the player directory inside the Flex SDK. Install the AIR runtime from Adobe web site
- Download the docs at http://livedocs.adobe.com/flex/3/flex3_documentation.zip
- Alternatively buy and download the Eclipse-based professional version called “Adobe Flex Builder” which already include all SDKs and docs

What is Adobe Flex?

- Is an RIA platform for building next-generation web apps
- Its uses a framework created over the Flash Player API
- Is browser and OS independent (by having its own plug-in and its own VM)
- Contains a free open source SDK that contains source code, compilers, debugger, and debug flash players which can be used to create web and desktop apps
- Uses XML based tags to describe your RIA graphical user interface and uses ActionScript 3.0 to code the interactivity
- Developers can buy Flex Builder which contains an environment built on Eclipse with source-code editor support, Wizards and Data Access features

Adobe Flex Technologies

- MXML (Macromedia XML)
- ActionScript 3.0 based on ECMA-262 draft 4
- Flex Framework which contain ActionScript-based graphical and utility components (Flex SDK)
- Adobe Integrated Runtime (AIR) which allows desktop apps to be written using the Flex framework and AIR SDK
- Eclipse 3.3 based Flex Builder IDE
  - Flash Debug Player or plugin/ActiveX
- AVM2 (virtual machine which runs in the Flash Player plugin , which also contains the AVM1 virtual machine for ActionScript 2.0 and 3.0 support)

Key facts about Flex

- Flex Applications are Flash Applications. Flash runtime or Flash Player must be present to run a Flex app.
- The Flex SDK contains the predefined class libraries and application services necessary to create Flex applications
- Flex applications are written using MXML and/or ActionScript 3.0. Then all code is compiled to a SWF format
- Data can be access by HTTP, WebServices, or in binary format using AMF (Active Message format)
- Developers typically layout their applications in MXML which is generated by a design tool in Flex Builder
- Developers code Actionscript 3.0 for all user interaction and GUI development

Why use Flex for RIA development?

- Very short learning curve for Java technology developers, (approx 2 weeks) due to ActionScript 3.0 using much the same syntax as Java technology
- Developer layouts a RIA application visually using a design tool and have your design be executed through code generation to a Flash video file (SWF)
- Supports Flash multimedia streaming technology which is useful for adding video and audio to your apps
- Has two-way call level support for JavaScript™ technology, and therefore compatible with all AJAX based technology
- Allows complete control over look and feel (skinning) at runtime using CSS (Cascading Style Sheet) settings for visual classes

Why Flex Builder? (Product is optional)

- Has express install that generates an HTML wrapper fro your flex apps so you are ready to deploy to a website right away. The wrapper will download the Flash runtime if the user does not already have it installed
- Contains an excellent profiler which visually shows memory allocation and performance bottleneck
- Has data access wizards that generate ActionScript code from a WSDL, a database schema, or an XML store
- Contain a design view that allows drag-and-drop layout of visual components, class property setting, and CSS file generation
- Provides all the benefits of Eclipse 3.3 including Quick Fix and other code editor refactoring support

What is AIR? (Adobe Integrated Runtime)

- Consist of a runtime that acts as a layer between your desktop app and the OS. This is what makes your AIR apps portable across platforms
- AIR uses an HTML rendering engine from http://webkit.org included in many browsers
- AIR reuses the Flex framework and adds additional classes to access the local file system, clipboard, and a local SQLlite database
- Supports creating non-standard Window classes, like circular windows or transparent square windows
- All AIR apps are signed by the developer using certificates -based security technology
- Create native Windows, Mac OSX and Linux GTK(soon) desktop apps

RIA with MXML and ActionScript 3.0

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlnx=
"http://www.adobe.com/2006/mxml"
usePreloader=true
frameRate= 60 pageTitle="JavaOne TS-5389"
backgroundGradientColors="[0xCCCCCC, 0x66CCFF]"
backgroundColor="0xCCCCCC" horizontalAlign="center"
applicationComplete="appComplete();">
<mx:Script>
<![CDATA[
 private function appComplete():void {
        myTextControl.text += "Hello RIA World!";
}]
</mx:Script>
 <mx:TextInput id="myTextControl" />
</mx:Application>
```

MXML demystified

- All Flex Web apps start with the <mx:Application> tag
- AIR Apps start with an <mx:WindowedApplication> tag
- XML Attributes allow the underlying Application Object to be preset with certain data attributes like framerate which tells the flash player to take advantage of the client video card capability
- We can integrate MXML and ActionScript by using the CDATA section of the <mx:Script> tag
- When the previous application object has finished loading the "applicationComplete" event is fired and the appComplete() ActionScript function is called
- The viewable text content of the TextInput control is then set with "Hello RIA world!"

Are MXML tags objects?

- MXML tags have frequently a 1-to-1 relationship with the underlying visual ActionScript class
- So <mx:Application> refers to the Application class under the MX package in the Flex framework
- The source code is under your Flex Builder install directory "/sdks/3.0.0/frameworks/project/framework/src"
- XML attributes set on the MXML tags are implemented by get and set functions of the property of the class
- Remember that all MXML will generate ActionScript code that is compiled along with your hand written ActionScript
- The tags cause the ActionScript objects to be created physically in memory in the Flash Player Virtual Machine

Demo Explorer

Adding video to Flex 3

- Use Flash Player 9 update 3, or version 9.0.115
- Full support for H.264 (MPEG-4). This includes High Definition video
- Full support for HE-AAC Audio, the successor to MP3
- Graphics Hardware acceleration
- Runs on Windows, Mac (OS X and Linux 32 bit)
- Multi-core CPU support. Bitmaps, filters, vectors, and video can now be split across processors to make playback more efficient while ActionScript is executed on the first processor
- Full screen support
- They used Codec SDK 7.0 from mainconcept
- Check them out at http://www.mainconcept.com

Using video and attaching a camera

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
       creationComplete="myVideo.pause();">
<mx:VideoDisplay id="myVideo"
                 creationComplete="initCamera();"
                  source="../assets/MyVideo.flv"
                  autoBandWidthDection="true"
                   height=300 width=300 />
<mx:Button label="||"   click="myVideo.pause();"/>
<mx:Button label="&gt;" click="myVideo.play();"/>
</mx:Application>
Import flash.media.Camera;
public var cam:Camera; // declare a camera varaible
public function initCamera():void {
  cam = Camaera.getCammera();
  // attach the camera to the video class
  myVideo.attachCamaera(cam);
}
```

Streaming video from the server - connect

```
// first create variables for the video name,
// connection and stream objects
private var videoURL:String = "Video.flv";
private var connection:NetConnection;
private var stream:NetStream;
public function VideoExample():void {
// create the connection
connection = new NetConnection();
// add the event listners for error handling
connection.addEventListener(NetStatusEvent.NET_STATUS,
                        netStatusHandler);
connection.addEventListener(
                          SecurityErrorEvent.SECURITY_ERROR,
                      securityErrorHandler);
connection.connect(null); // setup an empty connection
}
```

Streaming video from the server – show, play

```
 // Connect to a video on the server to stream
 private function connectStream():void {
 // create the stream
 var stream:NetStream = new NetStream(connection);
 // add event handlers in cause of errors
stream.addEventListener(NetStatusEvent.NET_STATUS,
                        netStatusHandler);
stream.addEventListener(AsyncErrorEvent.ASYNC_ERROR,
                       asyncErrorHandler);
 // create the video and attach to the stream
 var video:Video = new Video();
video.attachNetStream(stream);
 // finally play the video
 stream.play(videoURL);
 // makes sure its visible to the user
 addChild(video);
}
```

Streaming audio from the server

```
// import the needed sound and URLRequest clases
import flash.media.Sound;
import flash.media.SoundLoaderContext;
import flash.net.URLRequest;
// create the Sound class
var songStreamed:Sound = new Sound();
// prepare an URL request for the named sound file
var req:URLRequest = new
          URLRequest("http://av.adobe.com/bigSound.mp3");
// create 8 seconds of buffering when loading sound
var context:SoundLoaderContext = new
                    SoundLoaderContext(8000, true);
// stream the song from the website and play it
songStreamed.load(req, context);
var channel:SoundChannel = songStreamed.play();
// call OnPlayCmplete function when song is over
channel.addEventListner(Event.SOUND_COMPLETE, OnPlayComplete);
```

Defining your look and feel with CSS

```
/* styles/runtime/assets/CoolStyles.css */
/* CSS that defines the App and Button class Style */
Application {
 backgroundImage: "orangeBackground.gif";
 theme-color: #9DBAEB;
}

Button {
  fontFamily: Tahoma;
  color:      #000000;
  fontSize:   18;
  fontWeight: bold;
  text-roll-over-color: #000000;
  upSkin:     Embed(source="orb_up_skin.gif");
  overSkin:   Embed(source="orb_over_skin.gif");
  downSkin:   Embed(source="orb_down_skin.gif");
}
```

Load and apply CSS styles by button selection

```
/* Compile the CSS file and load at runtime */
<mx:Application
 xmlns:mx =”http://www.adobe.com/2006/mxml”>
 <mx:Script>
 <![CDATA[
  import mx.styles.StyleManager;
  public function applyRuntimeStyleSheet():void{
  StyleManager.loadStyleDeclarations(
    "../assets/CoolStyles.swf", true);// apply right away
  }
 ]]>
 </mx:Script>
 <mx:Label text="Click the button to load a new CSS-based
             SWF file"/>
 <mx:Button id="CSSButton" label="Click Me"
        click="applyRuntimeStyleSheet()"/>
 </mx:Application>
```

Adding special effects

```
<!-- define two special effects -->
<mx:Dissolve id="dissolveOut" duration="1000"
         alphaFrom="1.0" alphaTo="0.0"/> <mx:Dissolve
id="dissolveIn" duration="1000"
             alphaFrom=0.0”  alphaTo=”1.0”/>
<!-- create a checkbox control -->
<mx:CheckBox id="cb" label="visible" selected="true"/>
<!-- apply the effects to an image and a button
     when the checkbox is selected or unselected -->
<mx:Image  source=”@Embed(source='asserts/myPic.png')”
           visible=”{cb.selected}”
           hideeffect=”{dissolveOut}”
       showffect=”{dissolveIn}”/>
<mx:Button label="Purchase"
           visible="{cb.selected}"
       hideEffect="{dissolveOut}"
   showEffect="{dissolveIn}"/>
```

Java and Flex Integration

Flex Data Access and Security choices

- Read the Flash Player 9.0 Security paper
- Add a crossdomain.xml file to the server with the data, -(recommended). This file restrict access to only certain domain on the web site OR
- Upload your SWF file to the same server as the web server. SWF files are usually stored in a directory under your web root OR
- Create a proxy on your web server that calls the data service, and put your SWF file on the same server as the proxy
- Now choice between HTTPService, WebServices SOAP based (requires a WSDL), or Remote Access using Adobe binary protocol --Active Message Format

Specialized Flex supported backend

- **BlazeDS** is an open-source Flex backend service which runs as a Java Technology Server for Flex Clients. Its purpose is:
- **Remoting** – Remoting provides a call and response model for accessing external data from Flex or Ajax application
- **Messaging** - Developers can push messages from the server on an event basis, or can transmit to other connected Flash clients in real time
- **Action Message Format (AMF)** transport protocol support
- **Software Clustering:** For messaging applications deployed in a cluster using Publish/subscribe or point-to-point messaging deployment
- **Proxy-Service**s –For SOAP-compliant web services and HTTP (REST) services). Used to avoid the Security cross-domain restrictions

Java technology and Flex working together

- Adobe sells a high-end Enterprise product that provides more infrastructure for J2EE projects called Adobe LifeCycle ES. Included is a JSP tag library for Flex among other services
- Consider using JSP™ frameworks as your Java API to Flex
- If you want Flex to call an EJB™ architecture, write a Java class that wraps that call, and remote call that from Flex
- Use BlazeDS as your entry point in Flex/Java technology integration. Install the WAR files and configure your endpoints. Note it comes with an administrator app and uses Tomcat 6.0 as a servlet container. Get BlazeDS at http://opensource.adobe.com/wiki/display/blazeds/Release+Builds
- Look at other RIA platform servers like WEBORB from Midnight coders at http://www.themidnightcoders.com/weborb/
- If you do not need direct access, use Flex mx:HttpService or XML/SOAP access using mx:WebService calls instead

Asynchronous Messaging

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
        creationComplete="consumer.subscribe()">
<mx:Script>
<![CDATA[
import mx.messaging.events.MessageEvent;
import mx.messaging.messages.AsyncMessage;
private function send():void {
        var message:AsyncMessage = new AsyncMessage();
        message.body.chatMessage = msg.text;
        producer.send(message);
        msg.text = "Hello from Flex";
}
private function messageHandler(event:MessageEvent):void {
log.text += event.message.body.chatMessage + "\n";
        } ]]> </mx:Script>
```

Using the Remoting Service – Configuration

- Point to your different server configuration by passing the services-config.xml file location to Flex Builder “additional Compiler arguments” text area
- Define the technology adapter. This translate the specific AMF message into the specific technology (Java-object)
- Set the “Channel”. This defines the location where the server with the remote service exists and what technology the server is using. The default is “AMF”
- Set the “Destination handles”. These are the grouping of adapters, channels, and custom properties that Flex will reference by the “id”
- By setting the “source” property in the XML this is the exact Java class name that the destination is referencing.
- Use remote-config.xml for the destinations

Sample remote-config.xml

```
<service id="remoting-service"
    class="flex.messaging.services.RemotingService"
    messageTypes="flex.messaging.messages.RemotingMessage">
    <adapters>
        <adapter-definition id="java-object"
            class="flex.messaging.services.remoting.adapters.JavaAdapter"
            default="true"/>
    </adapters>
    <default-channels>
        <channel ref="flex-amf3"/>
    </default-channels>
    <destination id="CustomerServiceRO">
        <properties>
            <!-- Actual Java class name on the server under /WEB_INF/Classes -->
            <source>com.samples.javaon2008.CustomerService</source>
             <scope>application</scope>
        </properties>
      <adapter ref=”java-object”/>
    </destination>
</service
```

Working together - Java Server side

```
public class CustomerService implements ValueListService {
public int getNumElements() {
      CustomerDAO dao = new CustomerDAO();
      return dao.getSize();

} // Called by the Flex client in the browser or desktop
  public CustomerEntryVO[] getElements(int begin,int count)
  {
     CustomerDAO dao = new CustomerDAO();
     try {
      CustomerEntryVO[] list =
                     dao.getList(begin,count);
       } (catch (SQLExceiptn ex) {
         ex.printStatckTrace();
       }
    }
  }
```

Calling the Java Server to get data

```
<mx:Application xmlns:mx="/2006/mxml">
<mx:RemoteObject id="MyRemoteJavaClass"
      destination="CustomerServiceRO"
      result="handleResult(event)"
      endpoint="messagebroker/amf?id=flex-amf3"/>
 <mx:DataGrid>
       dataProvider
       ="{MyRemoteJavaClass.getElements.lastResult}"
        updateComplete="done()">
  <mx:columns>
   <mx:DataGridColumn dataField="name"/>
   <mx:DataGridColumn dataField="rank"/>
   <mx:DataGridColumn dataField="offerings"/>
  </mx:columns>
 </mx:DataGrid>
</mx:Application>
```

More fun with Java Technology Using the Flex Compiler API

Calling the Flex Compiler API from Java Technology

- Flex 3 includes a Java-based compiler API that lets you compile SWF and SWC files from Java applications
- The API supports all the same options as the mxmlc and compc command-line compilers
- The API includes classes like Application, Logger and Project
- The API is contained in flex-oem-compiler.jar
- The capability to create application and library source files at run time and to compile these files into Flex applications and libraries
- Create Flex applications in memory and compile them to SWF files without ever having an MXML file on disk

Creating a Project and SWF from Java Technology

```
Import flex2.tools.oem.Application;
import flex2.tools.oem.Project;
import flex2.tools.oem.Configuration;
import flex2.tools.oem.Library;
import java.io.*;
public class MyProjectCreator {
public static void main(String[] args) {
String assetRoot = "../assets/";
String outputRoot = "../apps/";
Project project=new Project();
// Create the Application.
Application app = new Application(new File(outputRoot,
"TestAppWithComponents.mxml"));
app.setOutput(new File(outputRoot, "TestAppWithComponents.swf"));
app.setLogger(new ComplexLogger());
// go to next slide
```

Creating a Library (SWC) from Java Technology

```
Library lib=new Library(); // Create the Library.
lib.setOutput(new File(assetRoot, "MyComponents.swc"));
lib.addComponent(new File(assetRoot, "MyButton.mxml"));
lib.addComponent(new File(assetRoot, "MyLabel.mxml"));
lib.setLogger(new ComplexLogger());
// Add the new SWC file to the library-path.
Configuration config = app.getDefaultConfiguration();
config.addLibraryPath(new File[]
{new File(assetRoot, "MyComponents.swc")});
app.setConfiguration(config);
// Add Application and Library objects to the Project.
project.addBuilder(app);
project.addBuilder(lib);
project.dependsOn(app, lib);
project.build(true); // finally do the custom build from Java
}
```

More fun with Java Technology
Using the Flex Builder extensibility API

Flex Builder 3 Extensibility

```
// Get all the class names in a Flex Project
IFlexProject flexProject = workspace.getProject();
CMFactory.getRegistrar().registerProject(flexProjet
                        .getProject(), null);
sychronized ICMFactory getLockObject() {
IProject project = CMFactory.getManager()
.getProjectFor(flexPorject.getProject());
IClassNameIndex classIndex = (IClassNameIndex)
      project.getIndex(IClassNameIndex.ID);
IClass[] allClasses = classIndex.getAllClasses();
String [] allClassesNames = new
String(allClasses.length);
for (int i = 0; i < allClasses.length; i++) {
   allClassesNames[i] =
allClasses[i].getQualifiedName();
} }
```

Summary

- Flex can be used for web and desktop applications from the same source by re-targeting the SDK
- Flex provides streaming video, audio, special effects and runtime skinning(via CSS) for your RIA
- Flex does not lock you into any particular web technology, API, JavaScript library, browser version, and OS
- There are public API that allows quick integration to common Web software Platform like Yahoo, eBay, Amazon, YouTube, and Facebook
- ActionScript 3.0 and MXML are easy for Java technology developers to learn (less than two weeks learning curve)
- The Flex framework and the compilers are open-sourced

THANK YOU

Tony Constantinides, CEO,
Constant Innovations Inc.
625 Powell Street, #41
San Francisco, CA
http://www.riajava.com

TS-5389