

MAX 2006 Beyond Boundaries

Matthew Boles

Adobe Customer Training
Technical Lead

RI101H: Your First RIA with Flex 2

October 24-26, 2006



What You Will Learn



- Functionality of the Flex product family
- The basic elements of a Flex application, emphasizing components
- How to use containers and controls to build a UI in Flex
- How to handle system and user events
- How to retrieve remote data from Java objects



How Will You Learn It



- Slides/Lectures
- Demonstrations
- Hands-on Walkthroughs



Content Outline



- Understanding Flex
- Creating a Simple Flex Application with Components
- Using Controls
- Creating Event Handlers
- Retrieving Remote Data
- Q & A



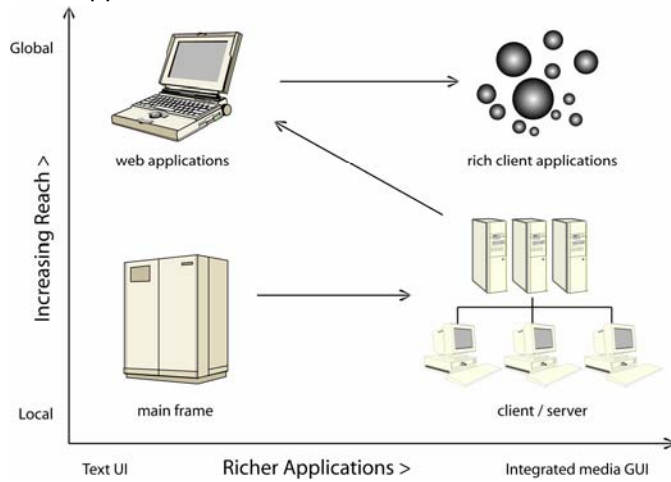
I. Understanding Flex



Why Flex?



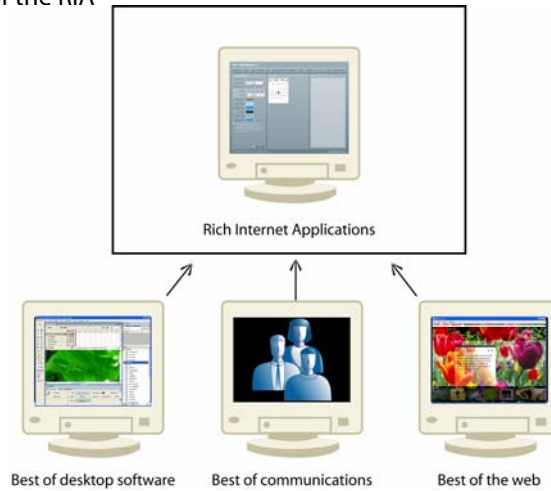
Evolution of Applications



Why Flex?



- The strength of the RIA



2006 Adobe Systems Incorporated. All Rights Reserved.

7



Adobe Flex 2 Product Family



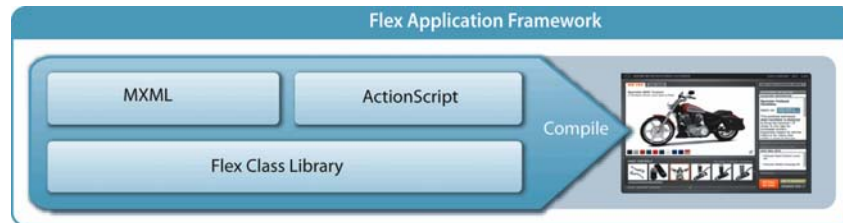
- The Flex product line provides the next generation of developer tools to build and deploy RIAs on the Flash Platform
 - Flex is an umbrella term for all the technologies of the Flex product line
- Product line consists of
 - Adobe Flex Builder 2
 - Adobe Flex 2 SDK
 - Adobe Flex Data Services 2
 - Adobe Flex Charting 2

2006 Adobe Systems Incorporated. All Rights Reserved.

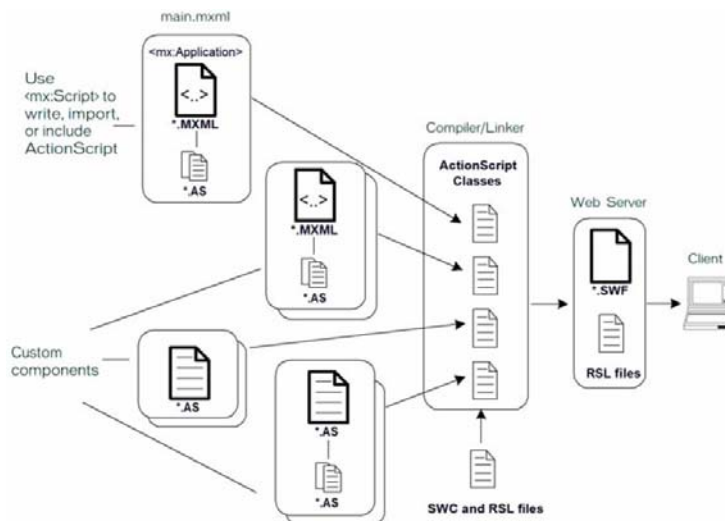
8



Flex Application Framework



Flex Application Process Flow



II. Creating a Simple Flex Application with Components



XML Document Type Declaration



```
<?xml version="1.0" encoding="utf-8"?>  
[other MXML tags]
```



<mx:Application> Tag



- Default container tag
- Use xmlns attribute for the namespace
 - A namespace is collection of names used in doc as MXML tags
 - Use a tag prefix to allow multiple namespaces

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  [other tags]
</mx:Application>
```



Label Control



- Use to display a single line of text
- Use text attribute for text to display

- As property (attribute)

```
<mx:Label text="Hello World!"/>
```

- As subtag

```
<mx:Label>
  <text>Hello World</text>
</mx:Label>
```



Creating a Binding



- Data binding is the process of binding the data of one object to another object
- Two ways to perform a binding
 - Curly braces ({})
 - <mx:Binding> tag
- Only covering curly braces in this session



Creating a Binding: Example



- Bind a Label's text value to the width of a Button
- ```
<mx:Label text="{myButton.width}"/>
```
- ```
<mx:Button id="myButton" label="Test Button"/>
```
- Value set by reference, so changes to the width of the Button reflected in Label's text value

```
<mx:Label text="{myButton.width}"/>
```

```
<mx:Button id="myButton" label="Test Button"
```

```
  click="myButton.width=300"/>
```



Walkthrough 1



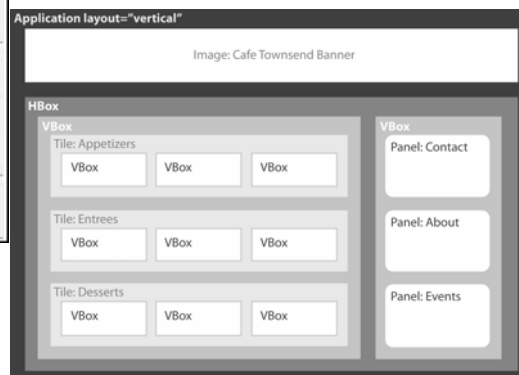
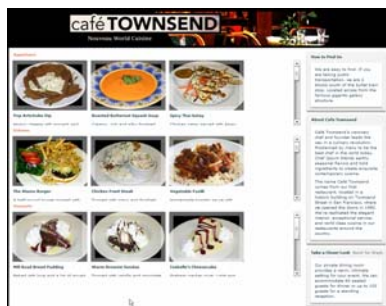
- Creating a simple MXML page
- Using a binding



Using Containers



- Containers control layout of UI controls and other containers



Using Containers



- Containers control layout of UI controls and other containers
- Application tag is a container
 - Define initial behavior with `Layout` property
 - absolute
 - vertical
 - Horizontal
- Containers used in this session
 - VBox
 - HBox
 - Panel



MXML Components



- Logically structure applications
- Produce more maintainable projects
- Each component often represents a different state of the application
- Only one Application tag on the main app, everything else is a component



Creating MXML Components



- To create an MXML component:
 - Create the component using the appropriate MXML tag in place of the <mx:Application> tag (VBOX, HBOX, FORM)
 - When invoking the MXML application, add an XML namespace

```
xmlns[:prefix]="subfoldername.*"

xmlns:local=""
```
 - Invoke component like any other MXML tag

```
<local:MyComp label="My Component" />
```



Demonstration 1: Using Properties and Method of Components



- Demo_Components.mxml



Walkthrough 2



- Using Components



III. Using Controls



Components Terminology



- Flex includes a **component-based development model** that you use to develop an application and its user interface
 - There are many pre-built components included with Flex, including both containers and controls
 - Use ASDoc to see details of component classes
- **Controls** are user-interface components that display content on the screen and provide ways to interact with the users
 - Examples are the Button, Image, TextArea and ComboBox controls



Components Terminology (cont)



- **Containers** are layout components that designate how components should be laid out on the screen
 - Examples are the HBox, VBox and Accordion containers
 - Within a container you place components, both controls and other containers, to appear within the container
- You can also extend the pre-built components or create new components as required by your application



ComboBox Control



- Supply data to the list-based controls using the **dataProvider** property
 - Data type of **dataProvider** often an ArrayCollection of objects
 - ArrayCollection class is a wrapper class that exposes an Array as a collection that can be accessed and manipulated using special methods and properties
- Same property to supply data to list-based controls
 - List
 - DataGrid
 - Tree
 - TileList
 - HorizontalList



ComboBox Control



- Specify the property of the objects in the ArrayCollection to be displayed by using the **labelField** or **labelFunction** property
- Example

```
<mx:ComboBox id="category"  
  dataProvider="{catData}"  
  labelField="catName"/>
```



Retrieving Data from a WebService



- Use the `<mx:WebService>` tag to retrieve remote data from a web service
- One of the three Remote Procedure Call service from Flex Data Service, along with
 - RemoteObject
 - HTTPService
- To use the WebService class
 - Supply a `wsdl` location
 - Call a remote method on some event
- To retrieve data at application startup
 - `load` event of WebService class itself
 - `creationComplete` event of Application
- User event during application interaction



Using Returned Results



- Results returned in the `lastResult` property of the method called
- Use the returned data as `webServiceInstance.methodName.lastResult`
- Bind the returned result to appropriate controls
- Later in session will use a `result` event to handle results
 - A better practice



Walkthrough 3



- Populating a ComboBox with Data Retrieved from a Webservice



Demonstration 2: Using a labelFunction



- Demo_LabelFunction.mxml



IV. Creating Event Handlers



Events



- Two types of events
 - System: Occur as a result of systematic code execution
 - User: Occur as a result of user interaction with the application



System Events



- Occur through the execution of code

Event	Description
creationComplete	Dispatched when the component has finished its construction, property processing, measuring, layout, and drawing
initialize	Dispatched when the component has finished its construction and has all initialization properties set
hide	Dispatched when an object's state changes from visible to invisible
result	The result event is dispatched when a service call successfully returns data

- Use the creationComplete event for default settings at startup; objects ready to use
- Events on the Application tag fire last



User Events



- Supports subset of DOM Level 3 events

Event	Description
click	A pointing device button is clicked over an element
change	An event that occurs when the value of a control changes, such as each key typed into a TextInput control or a new selection made in a ComboBox control
mouseDown	A pointing device button is pressed over an element. In the case of nested elements, this event type is always targeted at the most deeply nested element



Demonstration 3: System and User Events



- Demo_Events.mxml



Handling Events



- Handle user or system events in one of 2 ways
 - Using inline ActionScript
 - Using ActionScript functions in a script block
- Associate the ActionScript within the tag

```
<mx:TagName eventName="[ActionScript code or function call here]" />
```
- ActionScript can only be used as a value for an event, not properties, unless bindings ({}) are used



Handling Events in a Function



- If processing of a user event is more than one line of code, script it into a function
 - Enables reuse
- Code within `<mx:Script>` blocks
 - Treated as XML by parser
 - Wrap in CDATA to keep special characters from being processed



Handling Events in a Function: Example



```
<mx:Script>
  <![CDATA[
    function fillLabelControl():Void
    {
      myLabel.text="You Clicked";
    }
  ]]>
</mx:Script>

<mx:Button label="Click Me"
  click="fillLabelControl()" />

<mx:Label id="myLabel" />
```



Understanding the event object



- Every event object is an instance of the flash.events.Event class
- Each time an event occurs an event object is created
- Contains a large collection of named properties
- Some are standard properties, others depend on the event that occurs

```
<mx:Button label="Click Me" id="button1"  
click="fillLabelControl(event)"/>
```



Event Object Properties



- Some are the same for all events (*type* and *target*)
- Some are specific to the event broadcast
- Some are custom properties the developer specifies



Walkthrough 4



- Using a ComboBox control
- Handling an event



IV. Retrieving Remote Data



Data Retrieval Overview



- Three ways of accessing data from Flex
 - Dynamically retrieving XML data using HTTPService
 - Consuming web services
 - Calling Java methods/ColdFusion CFCs directly using RemoteObject
- Best performance is offered by the RemoteObject class
 - Uses Flex Data Services



Using Data from Java Objects



- Flex applications can call methods of Java objects that reside on the Java application server on which FDS is running
- Communication done with AMF
 - AMF (Action Message Format) is used in Flash Remoting; it is a binary protocol, is faster and uses less bandwidth than SOAP



<mx:RemoteObject>



- Example

```
<mx:Application ...
  creationComplete="simpleRO.outMethod()"/>

<mx:RemoteObject id="simpleRO"
  destination="sandwichService"/>

<mx:Label text="{simpleRO.outMethod.lastResult}"/>
```



Steps for RemoteObject Use



1. Create a RemoteObject instance using at least **id** and **destination** properties
2. Invoke the remote method on some event
- 3a. Use the returned data as ***remoteObjectInstance.methodName.lastResult***
- 3b. Use returned results in event handler from the event object



Remote Method Calls - Multiple Methods



- When different event handlers are used for multiple methods of a single remote object service, use the `<mx:method>` tag to assign result handlers for each remote method invoked
- The `<mx:method>` tag must be nested in `<mx:RemoteObject>` tag set
- Does not have an `id` property
- Must have a name that matches the name of the method invoked
- If no result handler or fault handler are specified, the handlers defined in `<mx:RemoteObject>` are used



Multiple Methods – Example



```
<mx:RemoteObject id="catRO"
  destination="sandwichService">
  <mx:method name="getTypes"
    result="catHandler(event)"/>
  <mx:method name="getSandwichesByType"
    result="sandwichHandler(event)"/>
</mx:RemoteObject>
```



DataGrid Control



- Often used to display datasets
- Formatted table where you can set editable cells
- Features
 - Resizable columns
 - Customizable column and row headers
 - Multiple modes of selection
 - Custom item editors and renderers
 - Paging of data
 - Drag and drop



DataGrid Control (cont)



- DataGrid creates a column for each property, a row for each record
 - Order of columns not related to data

style	number	key
Thai	6	0
Indian	14	1



DataGrid Columns



- DataGridColumn class refers to a column with the DataGrid
- Use a DataGridColumn to configure the DataGrid

```
<mx:DataGridColumn  
  dataField="No default."  
  headerText ="No default."  
  . . .  
>  
</>
```



Creating DataGridColumns



- One use of DataGridColumns is to control which columns and in what order they are displayed
- Columns are defined as DataGridColumn objects

```
<mx:DataGrid id="myDataGrid"  
  dataProvider="{myModel.result}">  
  <mx:columns>  
    <mx:DataGridColumn dataField="style"  
      headerText="Style" />  
    <mx:DataGridColumn dataField="number"  
      headerText="Available" />  
  </mx:columns>  
</mx:DataGrid>
```



Walkthrough 5



- Retrieving data from a remote object
- Displaying data in a DataGrid



Better by Adobe.™

