Flex

Flex

Flex es un conjunto de librerías para el desarrollo de RIAs

Es necesario tener instalada la Sdk(http://www.adobe.com/devnet/flex/flex-sdk-download.html)

Para implementar una RIA con felx se utiliza tanto el lenguaje script Action script, Como el lenguaje de marcas, MXML

ActionScript es a MXML como JavaScript es a HTML

MXML se usa para diseñar y ActionScript para asignar comportamiento a los componentes, o para implementar lógica de negocio.

Son tecnologías del lado del cliente y están basadas en un modelo orientado a eventos.

Actualmente las dos distribuciones de flex más utilizadas son

- Halo(mx) flex 3
- Spark flex 4

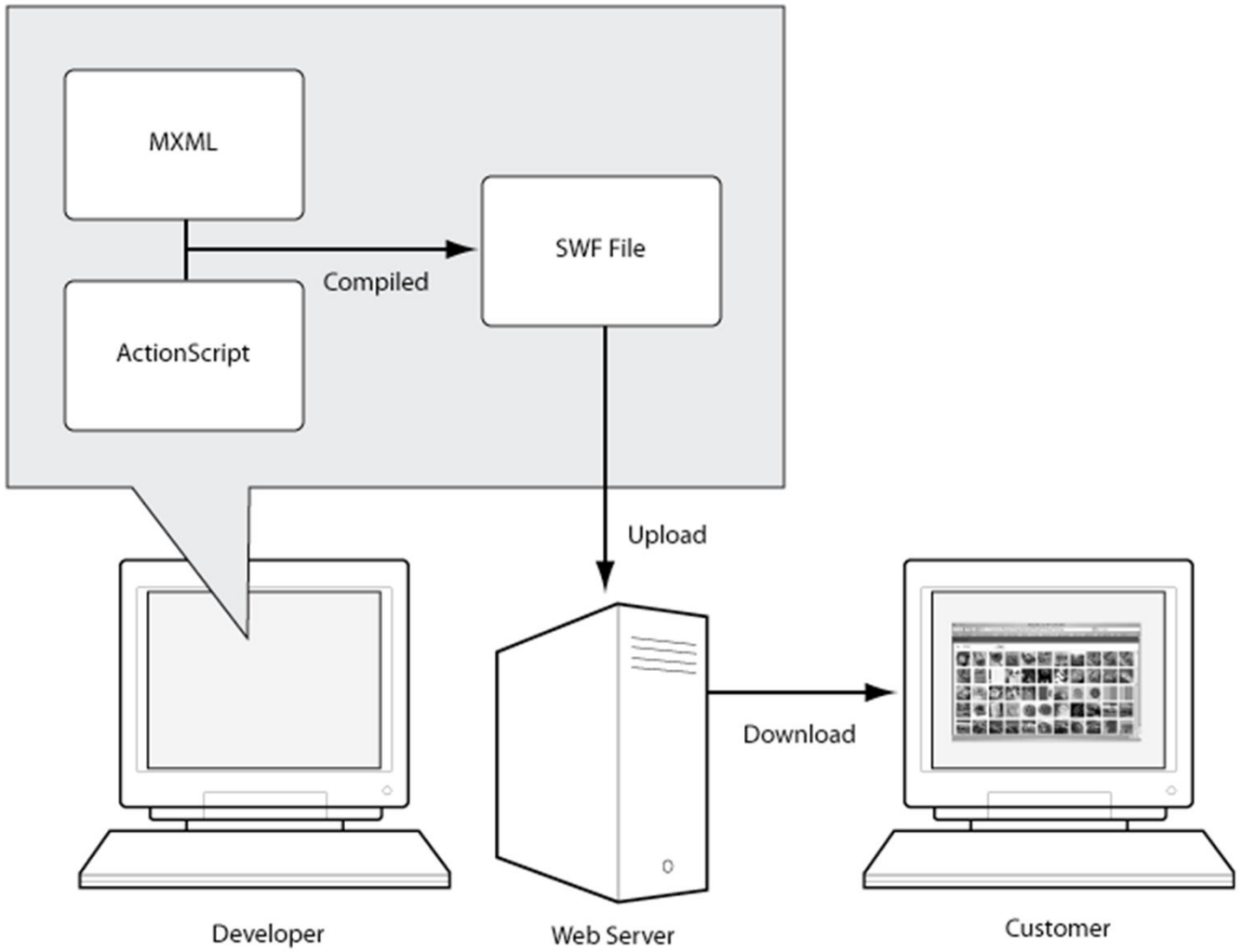
Flex

Para implementar una aplicación utilizando flex, se crea una jerarquía de paquetes con archivos as3 y/o mxml.

MXML - Contiene la estructura gráfica de la interfaz de usuario(también puede incluir script para manejo del comportamiento)

Action Script 3 - Comportamiento

Flex


MXML
ActionScript
Compiled
SWF File
Upload
Download
Developer
Web Server
Customer

Flex

Para programar utilizando mxml, existen una serie de componentes(tags),

Cada componente posee un atributo id con el cual se puede acceder a las propiedades o métodos de el mismo, de la misma manera como lo haría con una variable.

```
<fx:Script>
public function showit()
{
mx.controls.Alert.show('idMe is ' + idMe.text);
}
</fx:Script>
<s:Label id="idMe" text="Hi mom"/>
```

Componentes

Flex 3 MX Component	Flex 4 Spark Component
mx.controls.Button	spark.components.Button
mx.controls.ButtonBar	spark.components.ButtonBar
mx.controls.CheckBox	spark.components.CheckBox
mx.controls.ComboBox	spark.components.DropDownList (w/o editability)
mx.controls.HorizontalList	spark.components.List (with a HorizontalLayout)
mx.controls.HScrollBar	spark.components.HScrollBar
mx.controls.HSlider	spark.components.HSlider
mx.controls.Image	spark.primitives.BitmapImage (w/o support for external images)
mx.controls.LinkBar	spark.components.ButtonBar (with a custom skin)
mx.controls.LinkButton	spark.components.Button (with a custom skin)
mx.controls.List	spark.components.List
mx.controls.NumericStepper	spark.components.NumericStepper
mx.controls.RadioButton	spark.components.RadioButton
mx.controls.RadioButtonGroup	spark.components.RadioButtonGroup
mx.controls.TextArea	spark.components.TextArea
mx.controls.TabBar	spark.components.TabBar
mx.controls.TextInput	spark.components.TextInput
mx.controls.TileList	spark.components.List (with a TileLayout)
mx.controls.ToggleButtonBar	spark.components.ButtonBar
mx.controls.VideoDisplay	spark.components.VideoPlayer
mx.controls.VScrollBar	spark.components.VScrollBar
mx.controls.VSlider	spark.components.VSlider
mx.core.Application	spark.components.Application
mx.core.Window	spark.components.Window
mx.core.WindowedApplication	spark.components.WindowedApplication
mx.containers.ApplicationControlBar	spark.components.Application (with the controlBarContent)
mx.containers.Canvas	spark.components.Group
mx.containers.ControlBar	spark.components.Panel (with the controlBarContent property)
mx.containers.HBox	spark.components.HGroup
mx.containers.Panel	spark.components.Panel
mx.containers.Tile	spark.components.Group (with a TileLayout)
mx.containers.VBox	spark.components.VGroup

Flex

Contenedores

Son colecciones de componentes que tienen como misión proveer una estructura visual a una aplicación, son una forma de esquematizar el diseño.

En Flex 4, se incorporan los siguientes contenedores Spark:

Application
Único por aplicación, es el contenedor root de la aplicación. (<s:Application/>), este reemplazaría al contenedor <mx:Application> de Halo/MX. Posee la propiedad preloader, que permite visualizar la barra de progreso mientras se carga la aplicación. Puede contener variables globales y funciones, permitiendo el acceso a ellas desde cualquier parte de la aplicación.

Group
Contenedor básico para agrupar elementos, por defecto utiliza la clase BasicLayout para el posicionado absoluto. Sus contenedores hijos HGroup y VGroup proveen un posicionamiento de capas horizontal y vertical respectivamente.

SkinnableContainer
Es similar al de grupos, pero tiene la capacidad de incorporar pieles.

Flex

Listing 4.16 An `HGroup` inside a `Panel` is an easy way to add a margin from the edge.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx" >
  <s:Panel title="My Title">
    <s:HGroup top="5" bottom="5" left="5" right="5">
      <s:Button label="Button 1"/>
      <s:Button label="Button 2"/>
    </s:HGroup>
  </s:Panel>
</s:Application>
```

Constraints for padding

Figure 4.19 A `Panel` container adds a title bar and border around its content.

Flex

Contenedores

Panel
Contenedor con una barra de título y un frame.
Es usualmente usado como un contenedor top-level para la aplicación en su totalidad. Por defecto dibuja un borde alrededor de sus objetos secundarios.
Hereda de SkinnableContainer todas sus propiedades.

DataGroup
Está pensado para agrupar datos (ej. Arrays) que pueden ser renderizados, lo que permite una mejor visualización.

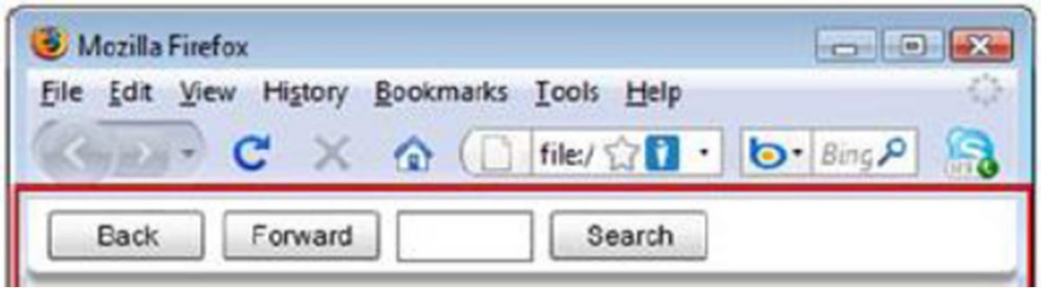
SkinnableDataContainer
Similar al DataGroup, pero una versión compatible con pieles. Los contenedores con la capacidad de pieles son más pesados y cargan con el contenido extra para el soporte de los mismos.

ApplicationControlBar
Crea un área en la parte superior de la aplicación que es similar al menú de archivo usado en la mayoría de aplicaciones de escritorio. Para usar este contenedor se combina con el contenedor de la aplicación.

Flex

Listing 4.17 ApplicationControlBar provides easy access to common functionality.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx" >
  <mx:ApplicationControlBar width="100%">
    <s:Button label="Back"/>
    <s:Button label="Forward"/>
    <s:TextInput width="60"/>
    <s:Button label="Search"/>
  </mx:ApplicationControlBar>
</s:Application>
```



Flex

Contenedores

BasicLayout
También conocido como el esquema absoluto, en el que utiliza coordenadas xy.

HorizontalLayout
Componentes que se colocan uno tras otro horizontalmente en una sola fila.

VerticalLayout
Los componentes se colocan uno tras otro verticalmente en una sola columna.

TileLayout
Muestra los componentes en forma de rejilla, creando tantas filas y columnas como sea necesario.

Canvas container
Son el tipo de contenedor más básico, están basados en los componentes Halo/MX.

Flex

Listing 4.13 Using multiple containers for more complex layouts

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx" >
  <s:layout>
    <s:VerticalLayout/>
  </s:layout>
  <s:Group>
    <s:layout>
      <s:HorizontalLayout/>
    </s:layout>
     <s:Button label="Button 1"/>
     <s:Button label="Button 2"/>
  </s:Group>
  <s:Group>
    <s:layout>
      <s:HorizontalLayout/>
    </s:layout>
     <s:Button label="Button 3"/>
     <s:Button label="Button 4"/>
  </s:Group>
</s:Application>
```

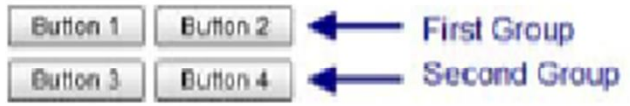


Figure 4.17 Two `Group` containers using horizontal layout coupled with an `Application` container using vertical layout

Flex

Listing 4.18 Associating a DataGroup's dataProvider with an array of strings

```
<fx:Script>
  <![CDATA[
    import mx.collections.ArrayCollection;
    [Bindable]
    public var someData:ArrayCollection =
           new ArrayCollection(["one","two","three"]);
  ]]>
</fx:Script>

<s:DataGroup
  dataProvider="{someData}"
  itemRenderer="spark.skins.default.DefaultItemRenderer">
    <s:layout>
      <s:HorizontalLayout/>
    </s:layout>
</s:DataGroup>
```

Flex

Listing 4.19 Using an array of components to display

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

  <fx:Declarations>
    <mx:ArrayCollection id="someData">
      <s:Button label="Button 1"/>
      <s:Button label="Button 2"/>
      <s:Button label="Button 3"/>
    </mx:ArrayCollection>
  </fx:Declarations>

  <s:DataGroup dataProvider="{someData}"
               itemRenderer="spark.skins.default.DefaultComplexItemRenderer">
    <s:layout>
      <s:TileLayout orientation="columns" requestedColumnCount="2"  />
    </s:layout>
  </s:DataGroup>
</s:Application>
```

1 Declare all non-visual tags first

Figure 4.21 A tiled layout of buttons using the `DataGroup` container

Podemos ver la declaración del tag <fx:Declarations/>, y es porque los tags no visualizables (por ejemplo <mx:ArrayCollection/>) no están permitidos para ser listados por si mismos si no son una propiedad del contenedor o clase superior

Flex

DividedBox, HDividedBox, and VDividedBox

Es un contenedor Halo/MX, permite que el usuario pueda redimensionar el área de trabajo

Listing 4.20 Making a dashboard with DividedBoxes

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx" >
  <s:Panel title="Report Dashboard"
           verticalCenter="0"
           horizontalCenter="0"  >
    <mx:DividedBox direction="horizontal" width="100%">
      <s:VGroup height="100%">
        <mx:Text text="Categories" fontWeight="bold"/>
        <s:Button label="Finance" width="100%"/>
        <s:Button label="Operations" width="100%"/>
        </s:VGroup>
        <mx:VDividedBox width="50%" height="100%">
          <s:VGroup width="100%" >
            <mx:Text text="Finance Reports" fontWeight="bold"/>
            <mx:Text text="2008 Q1 Sales"/>
            <mx:Text text="2008 Q2 Sales"/>
          </s:VGroup>
          <s:VGroup width="100%" >
            <mx:Text text="2008 Q1 Sales" fontWeight="bold"/>
            <mx:Text text="North America: $2,832,132"/>
            <mx:Text text="Europe: $1,912,382"/>
          </s:VGroup>
        </mx:VDividedBox>
      </mx:DividedBox>
    </s:Panel>
  </s:Application>
```

Flex

Form

En Flex la comunicación del lado del servidor es un proceso totalmente distinto a la aplicaciones web tradicionales.
Un contenedor de formulario es como cualquier otro contenedor, su cometido es organizar el diseño.

Ofrece tres tags con características especiales

1. Form: Contenedor principal.
2. FormHeader: Componente adicional para agregar los cabezales de secciones.
3. FormItem: Componente que permute asociar texto con cada componente de ingreso de datos.

Listing 4.21 Making use of form-related containers

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

  <mx:Form>
    <mx:FormHeading label="Contact Info"/>
    <mx:FormItem label="First Name">
      <s:TextInput/>
    </mx:FormItem>
    <mx:FormItem label="Last Name">
      <s:TextInput/>
    </mx:FormItem>
  </mx:Form>
</s:Application>
```

Flex

Grid

Es similar a una tabla en HTML, existe el tag GridRow para la entrada de columnas y GridItem para el ingreso de datos en cada celda

Si se desea que una celda abarque más de una columna, se puede utilizar el atributo colSpan en cualquier ítem del Grid.

Listing 4.22 Making use of the Grid container

```
<mx:Grid>
  <mx:GridRow>                                    <- Create the first row
    <mx:GridItem>
      <s:Button label="Rewind"/>
    </mx:GridItem>
    <mx:GridItem>
      <s:Button label="Play"/>
    </mx:GridItem>
    <mx:GridItem>
      <s:Button label="Forward"/>
    </mx:GridItem>
  </mx:GridRow>
  <mx:GridRow>                                    <- Create the second row
    <mx:GridItem colSpan="3">
      <s:Button label="STOP" width="100%"/>
    </mx:GridItem>
  </mx:GridRow>
</mx:Grid>
```

Flex

Botones

Button(Spark) El botón estándar, de uso múltiple para aceptar un clic del ratón.Permite fácilmente añadir una imagen en el botón.

ButtonBar(Spark) Dinámicamente genera una serie de botones en base a una matriz.

LinkButton(Halo) Es la versión de Flex de un vínculo HTML. No tiene bordes y pose un fondo transparente. Se puede agregar a un buttonbar.

ToggleButtonBar(Halo) Es una barra de botonoes, donde se destaca el último elemento oprimido.

PopUpButton(Halo) Un botón dual que combina la funcionalidad de dos botones(el lado izquierdo actúa como un botón normal, el lado derecho invoca otro objeto de interfaz de usuario). Se utiliza típicamente para crear una selección múltiple.

PopUpMenuButton (Halo) Descendiente del PopUpButton, se orienta específicamente a crear un menú desplegable.

Flex

Botones

Button(Spark) El botón estándar, de uso múltiple para aceptar un clic del ratón.Permite fácilmente añadir una imagen en el botón.

ButtonBar(Spark) Dinámicamente genera una serie de botones en base a una matriz.

LinkButton(Halo) Es la versión de Flex de un vínculo HTML. No tiene bordes y pose un fondo transparente. Se puede agregar a un buttonbar.

ToggleButtonBar(Halo) Es una barra de botonoes, donde se destaca el último elemento oprimido.

PopUpButton(Halo) Un botón dual que combina la funcionalidad de dos botones(el lado izquierdo actúa como un botón normal, el lado derecho invoca otro objeto de interfaz de usuario). Se utiliza típicamente para crear una selección múltiple.

PopUpMenuButton (Halo) Descendiente del PopUpButton, se orienta específicamente a crear un menú desplegable.

Flex

Validadores

Son un instrumento que permite validar los datos con los cuales se ingresaron campos. Contiene atributos para gestionar su funcionamiento y evento que las dispara

StringValidator
Permite validar que una cadena de caracteres cumple cierto formato o pertenece a cierto dominio.

NumberValidation
Permite validar si el valor ingresado es numérico y pertenece a cierto rango.

DateValidator
Permite validar que el campo ingresado es una fecha y pertenece a un rango.

RegExpValidator
Permite verificar que el valor ingresado pertenece a cierta expresión regular.

Flex

Listing 6.1 A basic validator in action

```
<?xml version="1.0"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">
  <fx:Declarations>
    <mx:Validator source="{username}" property="text" required="true" />
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="Enter your username:"/>
    <s:TextInput id="username" />
  </s:VGroup>
</s:Application>
```

Nonvisuals to be declared

Listing 6.2 Using `StringValidator` to check character count

```
<?xml version="1.0"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">
  <fx:Declarations>
    <mx:StringValidator
      source="{username}" property="text"
      minLength="3" maxLength="20"
      trigger="{submitButton}" triggerEvent="click"
      tooShortError="Your username must be at least 3 characters"
      tooLongError="As if you'll remember that long of a username"
    />
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="Enter your username:"/>
    <s:TextInput id="username" />
    <s:Button label="Submit" id="submitButton"/>
  </s:VGroup>
</s:Application>
```

Used to trigger validation

Flex

Listing 6.3 NumberValidator used to check a value and if the number is an integer

```
<?xml version="1.0"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">
  <fx:Declarations>
    <mx:NumberValidator
      source="{age}" property="text" allowNegative="false"
      negativeError="I highly doubt you're that young!"
      minValue="5" maxValue="110" domain="int"
      trigger="{submitButton}" triggerEvent="click"/>
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="Enter your age:"/>
    <s:TextInput id="age" />
    <s:Button label="Submit" id="submitButton"/>
  </s:VGroup>
</s:Application>
```

Listing 6.4 Using the DateValidator to validate a single-source input

```
<?xml version="1.0"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">
  <fx:Declarations>
    <mx:DateValidator
      source="{birthday}" property="text" inputFormat="mm/dd/yyyy"
      allowedFormatChars="/"
      trigger="{submitButton}" triggerEvent="click"/>
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="Enter your birth date:"/>
    <s:TextInput id="birthday" />
    <s:Button label="Submit" id="submitButton"/>
  </s:VGroup>
</s:Application>
```

Flex

Listing 6.10 RegExpValidator matches text against a RegEx pattern

```
<?xml version="1.0"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">
  <fx:Declarations>
    <mx:RegExpValidator source="{ssn}" property="text"
                        flags="gmi"
                        expression="\d\{3\}.\d\{2\}.\d\{4\}"
                        noMatchError="Your SSN is unrecognized."
                        trigger="{submitButton}"
                        triggerEvent="click"/>
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="Social Security Number:"/>
    <s:TextInput id="ssn" />
    <s:Button label="Submit" id="submitButton"/>
  </s:VGroup>
```

Global, multiline, ignore case

Listing 6.14 Using the change event, you can validate in real time.

```
<?xml version="1.0"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">
  <fx:Declarations>
    <mx:StringValidator source="{address}"
                        minLength="5"
                        property="text"
                        trigger="{address}"
                        triggerEvent="change"/>
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">

     <s:Label text="What's your address?"/>
     <s:TextInput id="address"  />
     <s:Button label="Submit" id="submitButton"/>
   </s:VGroup>
 </s:Application>
```

Use change event to trigger validation

Flex

Importar módulos externos en mxml

Si el código as3 se encuentra en el directorio por defecto

```
<fx:Script source=“nomberClase.as" >
//incluye el código en el mxml, similar al include de c
```

Si el código se encuentra dentro de un paquete

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

  import uy.edu.tecnoinf.as3.nombreClase;
  import mx.collections.ArrayCollection;
   …
   ]]>
</fx:Script>
```

Para invocar una función en un archivo externo

```
{outerDocument.nombreFuncion(params)}
```

o

```
{outerDocument[‘nombreFunción'].call(null, new Array(params))}
```

Flex

Navegación

Se pueden definir estados

```
<s:states>
 <s:State name=“est_1"/>
 <s:State name=“est_2"/>
 </s:states>
```

Y definir componentes que se incluyen según el estado que se éste actualmente

```
<s:Group>
 <mx:LinkButton label=“Estado 1" click="currentState = ‘est_2'" includeIn=“est_1/>
 <mx:LinkButton label=“Estado 2" click="currentState = ‘est_1'" includeIn=“est_2"/>
</s:Group>
```

Flex

Navegación

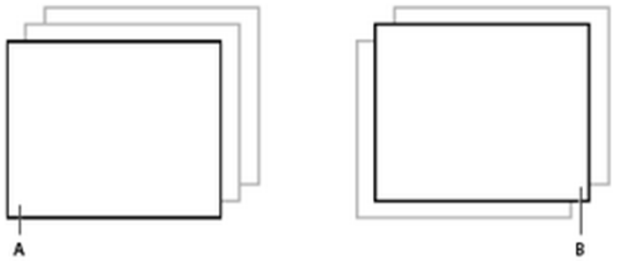
Utilizar contenedores de navegación (Tab Navigator)

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:WipeLeft id="myWL"/>
<mx:TabNavigator>
<mx:VBox label="Accounts" width="300" height="150" showEffect="{myWL}">
 <!-- Accounts view goes here. -->
<mx:Text text="This is a text control."/>
</mx:VBox>
 <mx:VBox label="Stocks" width="300" height="150" showEffect="{myWL}">
<!-- Stocks view goes here. -->
 <mx:Text text="This is a text control."/>
 </mx:VBox> <mx:VBox label="Futures" width="300" height="150" showEffect="{myWL}">
<!-- Futures view goes here. -->
<mx:Text text="This is a text control."/>
</mx:VBox>
 </mx:TabNavigator>
</mx:Application>
```

Flex

Navegación

Utilizar contenedores de navegación (View Stack)



A. *Child container 1 active* **B.** *Child container 0 active*

```
<!-- containers\navigators\VSLinkEffects.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:WipeUp id="myWU" duration="300"/>
<mx:WipeDown id="myWD" duration="300"/>
<mx:WipeRight id="myWR" duration="300"/>

<mx:VBox>
<mx:HBox borderStyle="solid">
<!-- Define the two buttons. Each uses the child container identifier to set the active child container. -->
<mx:Button id="searchButton" label="Search Screen" click="myViewStack.selectedChild=search;"/>
 <mx:Button id="cInfoButton" label="Customer Info Screen" click="myViewStack.selectedChild=custInfo;"/>
</mx:HBox>
<mx:ViewStack id="myViewStack" borderStyle="solid" width="100%"
creationCompleteEffect="{myWR}">
<mx:Canvas id="search" label="Search" hideEffect="{myWD}" showEffect="{myWU}">
 <mx:Label text="Search Screen"/>
 </mx:Canvas>
 <mx:Canvas id="custInfo" label="Customer Info" hideEffect="{myWD}" showEffect="{myWU}">
<mx:Label text="Customer Info"/>
</mx:Canvas>
</mx:ViewStack>
 </mx:VBox>
</mx:Application>
```

Flex

Navegación

Componentes MXML

```
<?xml version="1.0"?>
<!-- mxml/StateComboBox.mxml -->
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:dataProvider> <mx:String>AK</mx:String>
<mx:String>AL</mx:String>
<!-- Add all other states. -->
</mx:dataProvider>
</mx:ComboBox>
```

Podemos incrustar el contenido de un mxml en otro

```
<?xml version="1.0"?>
<!-- mxml/MXMLMyApplication.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:MyComp="*">
<MyComp:StateComboBox/>
</mx:Application>
```

Flex

Estilos

Se pueden usar hojas de estilo en cascada con en html

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

<fx:Style> .myFontStyle { fontSize: 15; color: #9933FF; } </fx:Style>

<s:Button id="myButton" styleName="myFontStyle" label="Click Me"/>


<fx:Style>

 @namespace s "library://ns.adobe.com/flex/spark";
  s|Button { fontSize: 15; color: #9933FF; }

   VBox Panel Button#button12 { color: #DDDDDD; }

</fx:Style>
```

Flex

Invocación de Servicios

Flex probé componentes para la invocación de servicios web o llamadas http. A través de la clase ResultObject obtenemos la respuesta del servidor.

HttpService es el componente que permite realizar invocaciones a través del protocolo http

WebService Permite invocar servicios web

Listing 15.1 The HTTPService object declared in MXML notation

```
<mx:HTTPService
    id="yahooHTTPService"
    url="http://search.yahooapis.com/WebSearchService/V1/webSearch"
    method="GET"                          <-- Request can be GET or POST
    makeObjectsBindable="true"            <-- Makes result object bindable
    result="responseHandler(event)"       <-- Declare the method to handle the res
    fault="httpFaultHandler(event)"       <-- Declare a method to handle a fault
    showBusyCursor="true">                <-- Show busy cursor while in progress
</mx:HTTPService>
```

Listing 15.3 Using the WebService component

```
<mx:WebService id="weatherService"                Declare the WSDL document location
   wsdl="http://www.webservicex.net/WeatherForecast.asmx?WSDL"
   fault="wsdlFault(event)">
     <mx:operation name="GetWeatherByZipCode"
        result="weatherResponse(event)"
        fault="weatherFault(event)"/>
     <mx:operation name="GetWeatherByPlaceName"
        result="weatherResponse(event)"
        fault="weatherFault(event)"/>
</mx:WebService>
```

(2) Multiple operation declarations within a WebService

Flex

Invocación de Servicios

Listing 15.2 Calling the `HTTPService` from ActionScript

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

  import mx.rpc.events.ResultEvent;
  import mx.rpc.events.FaultEvent;


     public function sendHttpRequest():void {
        var requestObj:Object = new Object();
        requestObj.appid = new String("YahooDemo");
        requestObj.query =
           new String("Flex in Action");
        requestObj.results = new int(2);
        yahooHTTPService.request = requestObj;
        yahooHTTPService.send();
     }

     private function
        responseHandler(e:ResultEvent):void {
        trace("Received a result: " + e.result);
     }

     private function
        httpFaultHandler(e:FaultEvent):void {
        trace("Received a Fault: " + e.message);
     }

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

1. Generic object created to send with request
2. Service variables passed into object
3. Generic object passed to property
4. `HTTPService.send()` method called
5. Response handler
6. Fault handler

Flex

Invocación de Servicio Web

```
<?xml version="1.0"?>
<!-- ds\rpc\WebServiceExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="useWebService();">

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.rpc.soap.mxml.WebService;
            import mx.rpc.events.ResultEvent;
            import mx.rpc.events.FaultEvent;

            private var asService:WebService;

            // Define the WebService component in ActionScript.
            public function useWebService():void {
                asService = new WebService();
                asService.destination = "ws-catalog";
                asService.addEventListener("result", wsResult);
                asService.addEventListener("fault", wsFault);
                asService.loadWSDL();
             }

             public function wsResult(event:ResultEvent):void {
                //Do something with the result.
             }

             public function wsFault(event:FaultEvent):void {
                var faultstring:String = event.fault.faultString;
                Alert.show(faultstring);
             }
        ]]>
    </mx:Script>
```

```
    <!-- Define the WebService component in MXML. -->
    <mx:WebService
        id="mxmlService"
        destination="ws-catalog"
        result="wsResult(event);"
        fault="wsFault(event);"/>

    <mx:Button label="MXML" click="mxmlService.getProducts();"/>
    <mx:Button label="AS" click="asService.getProducts();"/>

</mx:Application>
```

Flex

Referencias

http://livedocs.adobe.com/flex/3/html/help.htm

http://livedocs.adobe.com/livecycle/8.2/programLC/programmer/lcds/help.html?content=rpc_httpws_07.html

http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf69084-7a92.html

http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf67670-7ffe.html#WS2db454920e96a9e51e63e3d11c0bf69084-7a95

Bibliografía

Flex 4 InAction (Ahmed)

Fin