

MX DataGrids, Listas y Árboles

Flex 4 in Action

MX List genealogy

- Listas basadas en componentes de la librería MX
 - ListBase
 - AdvancedListBase

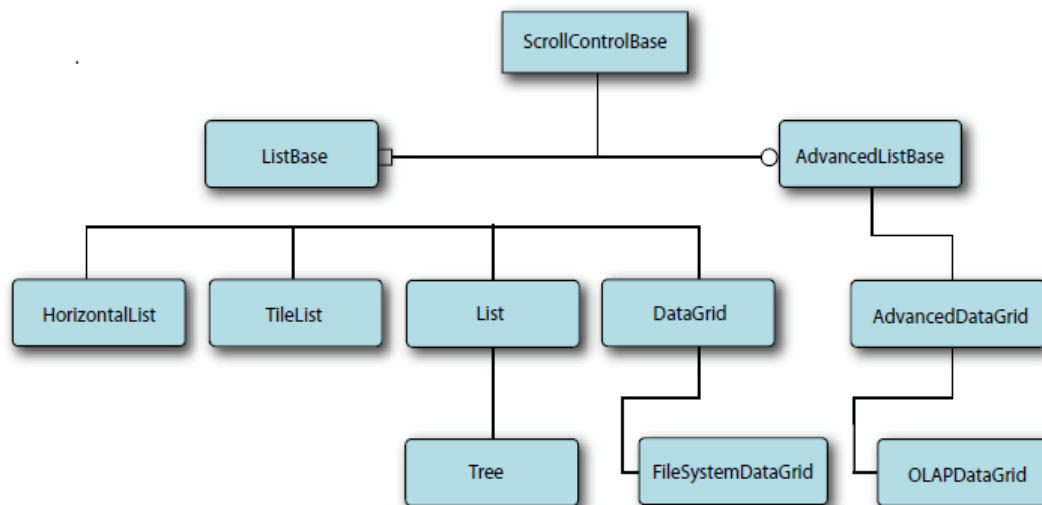


Figure 8.1 The List-based family of MX components and how they're related

Propiedad	Tipo	Descripción
columnCount	Number	Number of columns to be displayed
columnWidth	Number	Width of the columns
dataProvider	Object	Object containing the data to be displayed
iconField	String	Fieldname, if any, that contains a reference to an icon.
iconFunction	Function	Optional function name you'd like to run for each icon. This can be used for formatting the icon.
labelField	String	Field name that contains a value to be displayed in the column.
labelFunction	Function	Optional function name you can call for each record. The function determines how the label is formatted and displayed.
rowCount	Number	Number of rows to display.
rowHeight	Number	Height of each row in pixels.
selectable	Boolean	Value that indicates whether you're allowing items to be elected by the user. Options are true (default) and false.
selectedItem	Object	Link to the item that's currently selected. You can use this to access all the values pertaining to that row.
selectedIndex	Number	Row that has been selected (if any), if the component is selectable.
selectedItems	Array	Array of selectedItem for use when multiple selections are allowed.
variableRowHeight	Boolean	Value indicating whether each row can dynamically adjust its height to fit the content.

Propiedades comunes soportadas por listas basadas en componentes MX

Eventos MX ListBase

• Flex es un entorno orientado a eventos, así que saber que eventos están disponibles para listas basadas en componentes MX es muy importante.

Propiedad	Tipo	Descripción
change	Event	Triggers when the user selects a new row
dataChange	Event	Triggers when the data changes
itemClick	Event	Triggers when a user clicks a column
itemDoubleClick	Event	Triggers when a user double-clicks a column
itemRollOut	Event	Triggers when the user moves the mouse off an item
ItemRollOver	Event	Triggers when the user moves the mouse over an item

DataProvider

- La mayoría de los componentes que despliegan una serie de datos son alimentados por un objeto DataProvider.
- De lo único que debes preocuparte es de decirle al componente el nombre de la variable que contiene los datos que quieras presentar.
- Esto se puede hacer de varias maneras

Cargando un dataProvider

- ArrayCollection
 - evento **dataChange**
 - Notificación cambio de estado
- Tipos de colección
 - ArrayCollection
 - XMLListCollection
 - GroupingCollection
- Las listas basadas en componentes son los principales usuarios de las colecciones.

Inicializando colecciones

- Puedes usar dos métodos: MXML vía los tags asociados o utilizando ActionScript puro importando la clase ArrayCollection y después declarando una instancia de una variable, como se muestra a continuación ...
-
- `<fx:Script>`
- `<![CDATA[`
- `import mx.collections.ArrayCollection;`
- `public var myAC:ArrayCollection = new ArrayCollection([`
- `{ label:"Dan Orlando", data:"dorlando" },`
- `{ label:"Tariq Ahmed", data:"tahmed" },`
- `{ label:"John C Bland II", data:"jcbland" }`
- `]);`

Cargando colecciones

- ***Lista***

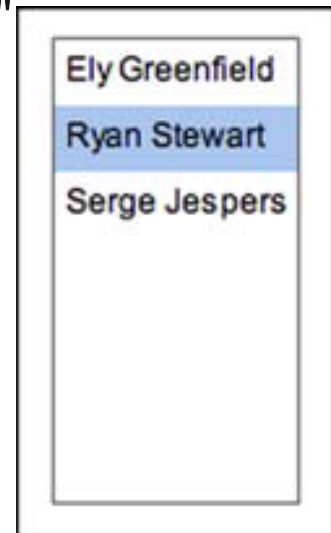
Las listas son el componente más liviano para desplegar información.

Puedes crear y cargar una lista de diversas maneras por ejemplo:

- Usando un **ComboBox** para desplegar un columna de nombres.
- Usando un **ArrayCollection**

Usando un componente ComboBox para desplegar una columna de nombres

- `<?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"`
- `height="300" width="400">`
- `<mx:List id="myFriends" x="10" y="10">`
- `<fx:String>Ely Greenfield</fx:String>`
- `<fx:String>Ryan Stewart</fx:String>`
- `<fx:String>Serge Jespers</fx:String>`
- `</mx:List>`



Otros tipos de listas...

- ***HorizontalList***: funciona exactamente igual que la lista por defecto, salvo que su orientación es horizontal.
- ***TileList***: `TileList` es similar al concepto de `sibling List`, pero en vez de tener una única columna crea una grilla con tiles de igual tamaño que contiene los ítems a presentar.

INVOCANDO UNA TILELIST

```
•<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:Script>
•<![CDATA[
•import mx.collections.ArrayCollection;
•[Bindable]
•public var myAC:ArrayCollection = new ArrayCollection([←CREATE DATA COLLECTION AND
•{name:"Dan Orlando", email:"dan@domain.com",    MAKE BINDABLE
•url:"http://danorlando.com"},
•{name:"Tariq Ahmed", email:"tariq@domain.com",
•url:"http://www.flexinaction.com"},
•{name:"John C Bland II", email:"john@domain.com",
•url:"http://www.johncblandii.com"},
•{name:"Joel Hooks C", email:"joel@domain.com",
•url:"http://www.joelhooks.com"}
•]);
•]]>
•</fx:Script>    ←SET DATAPROVIDER TO BINDABLE COLLECTION
•<mx:TileList id="myFriends" dataProvider="{myAC}" labelField="name"/>
•</s:Application>
```

email	name
tariq@domain.com	Tariq Ahmed
john@domain.com	John C Bland II
dan@domain.com	Dan Orlando

TileList vs Tile

- TileList
 - + Consume menos memoria
 - + Produce una respuesta inicial más rápida
 - Cuando los componentes son complejos y numerosos, hace que la aplicación se sienta más lenta (sluggish)
- Tile
 - + Scrolling perfecto
 - - Demora más en renderizar

DataGrid

- DataGrid ofrece características para ordenar las columnas y para intercambiar columnas de usuario – el usuario puede arreglar el orden de las columnas. Es similar al componente lista pero incluye el formato de múltiples columnas y encabezados de las columnas. Todo lo que discutimos acerca de la lista aplica al DataGrid, su invocación también es similar.

Contact Name	E-Mail
Tariq Ahmed	tariq@domain.com
Dan Orlando	dan@domain.com
John C Bland II	john@domain.com

Controlar la Ordenación

- sortableColumns
- **sorteable** en DataGridColumn

Flex provee a tag llamado DataGridColumn que trabaja en conjunction con el DataGrid.

Puedes usar el DataGridColumn tag para **controlar atributos** como estos:

- Column width
- Column header or title
- Word wrapping within a column
- Inline editing
- Instructions for handling mouse clicks in a column

Property	Type	Description
dataField		Indicates which field in your dataset is represented by the selected column.
headerText	String	Indicates the column title.
headerWordWrap	Special	Permits word wrapping for a column title. Options are true (word wrap enabled) and false (disabled). If no value is entered, defaults to the wordWrap property setting in the DataGrid component.
labelFunction	Function	If a function name is specified, passes the raw data to the function and displays the text that comes back. Useful for formatting raw data.
minWidth	Number	Specifies minimum width of a column.
resizeable	Boolean	Value that specifies whether the user is permitted to resize columns. Options are true (default) and false.
sortable	Boolean	Value that specifies whether the user is permitted to sort a column. Options are true (default) and false.
sortCompareFunction	Function	Adds the custom logic necessary to enable DataGrid to sort columns based on specified criteria such as numbers, dates, and so on.
sortDescending	Boolean	Indicates the default sort order.
visible	Boolean	Value that specifies whether the column is displayed.
width	Number	Specifies the width of the column.

Propiedades de un dataGrid

Tree

- Debido a la naturaleza anidada de la pantalla, tiene sentido que los datos necesitan estar estructurado en consecuencia. Una forma de datos estructurado como XML. En Flex 3, los objetos XML y XMMList eran un tanto redundante, en Flex 4 se elimina el objeto XML en favor de XMMList y XMMListCollection.
- Como una regla, un objeto XMMList sólo puede ser declarado en las etiquetas `<fx:Declarations/>`. El fragmento que sigue presenta un objeto XMMList dentro de la etiqueta Declaraciones.
 - `<fx:Declarations>`
 - `<fx:XMMList id="myXML">`
 - `<friends>`
 - `<friend name="Dan Orlando"/>`
 - `<friend name="John C Bland II"/>`
 - `<friend name="Tariq Ahmed"/>`

```
<?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:XMLListCollection id="myXMLCollection">
      <fx:XMLList id="myXML">
        <friends label="Friends">
          <friend label="Tariq Ahmed"/>
          <friend label="John C Bland II"/>
          <friend label="Ryan Stewart"/>
          <friend label="Joel Hooks"/>
        </friends>
        <families label="Family">
          <family label="Laura Orlando"/>
          <family label="Doug Orlando"/>
        </families>
      </fx:XMLList>
    </mx:XMLListCollection>
  </fx:Declarations>
  <mx:Tree dataProvider="{myXMLCollection}"
    labelField="@label"
    width="300" height="200" />
</s:Application>
```



Cargando el XML data Tree

•DISTINTAS INTERACCIONES...

- Manejar la interacción del usuario pasándole un objeto evento a una función
- *Pasando datos a una función*
- *Acceder directamente a una fila seleccionada*
- *Enlazar a cierta fila*

Tabla de eventos que pueden ser utilizados para manejar la interacción con una lista basada en componentes

Evento	Descripción
click	Occurs when the user clicks a component.
doubleClick	Fires when the mouse button is pressed twice.
itemClick	Indicates which row and column were clicked.
change	Occurs when the user clicks a different row than the current selection.

Interactuando con listas basadas en componentes

Uso de Listas con la librería Spark

Introducción

En el capítulo anterior se vio cómo utilizar las listas simple de flex, ahora veremos cómo utilizarlas con la librería Spark.

La librería de Spark permite mas posibilidades de escalabilidad, por ejemplo, permite que un usuario avanzado de Flex modifique el diseño de la interfaz.

Genealogía

Como se vio en el capítulo 8 los componentes de la librería MX son derivados de ListBase y AdvancedListBase. En Spark el componente extiende de la ListBase de Spark ubicada en el paquete `spark.components.supportClasses` y esta a su vez extiende de `SkinnableDataContainer`.

Lo importante aquí es comprender que cada clase hereda de una clase ListBase propia de su librería sea MX o sea Spark.

Genealogía

Identificando el componente correspondiente

Flex automaticamente sabe cual de los namespaces usar.

Si en el mxml se encuentra el tag `<mx:List/>` usara la ListBase de la librería MX.

Si por el contrario utiliza `<s:List/>` usara la clase ListBase de la librería de Spark.

Genealogía

Elementos renderizados con las listas spark

En MX y Spark List extends ListBas.

En MX ListBase extiende de ScrollControlBase la cual extiende de UIComponent.

Por el contrario la ListBase de Spark extiende de SkinnableDataContainer la clase que es responsable de mostrar la información de los items usando item renderers.

Listas Spark - Controles

No es difícil crear una custom List in flex 4

Es tan fácil como versiones anteriores del SDK, la mejor forma es extender la clase ListBase de Spark o la clase SKinnableDataContainer.

Listas Spark - Controles

Control ButtonBar

El control ButtonBar es similar a el ToggleButtonBar de la librería MX.

Este control es Básicamente es una Lista ya que extiende de la clase ListBase

Es util en la creación de un menu es facil de crear usando el control de Spark Button-Bar.

Listas Spark - Controles

Control ButtonBar

```
<?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"
    minWidth="1024" minHeight="768">
    <s:layout>
        <s:VerticalLayout paddingLeft="20" paddingTop="20"/>
    </s:layout>
    <s:ButtonBar>
        <mx:ArrayCollection>
            <fx:String>Button One</fx:String>
            <fx:String>Second Button</fx:String>
            <fx:String>Button Three</fx:String>
            <fx:String>Last Button</fx:String>
        </mx:ArrayCollection>
    </s:ButtonBar>
</s:Application>
```

Listas Spark - Controles

Control Spark List

```
<?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"
    xmlns:comp="com.danorlando.components.*"
    minWidth="600" minHeight="450">
    <s:layout>
        <s:VerticalLayout gap="1" useVirtualLayout="true" />
    </s:layout>
    <s>List>
        <s:dataProvider>
            <mx:ArrayCollection>
                <fx:String>Menu Item 1</fx:String>
                <fx:String>Second Menu Item</fx:String>
                <fx:String>Third Menu Item</fx:String>
                <fx:String>Fourth Menu Item</fx:String>
                <fx:String>Last Menu Item</fx:String>
            </mx:ArrayCollection>
        </s:dataProvider>
    </s>List>
</s:Application>
```

Listas Spark - Controles

Control DropDown

```
<?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"
    minWidth="1024" minHeight="768">
    <s:layout>
        <s:VerticalLayout paddingLeft="20" paddingTop="20"/>
    </s:layout>
    <s:DropDownList width="200">
        <mx:ArrayCollection>
            <fx:String>Item One</fx:String>
            <fx:String>Second Item</fx:String>
            <fx:String>Item Three</fx:String>
            <fx:String>Last Item</fx:String>
        </mx:ArrayCollection>
    </s:DropDownList>
</s:Application>
```

Listas Spark - Controles

Control DropDown

```
<?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"
    minWidth="1024" minHeight="768">
    <s:layout>
        <s:VerticalLayout paddingLeft="20" paddingTop="20"/>
    </s:layout>
    <s:DropDownList width="200">
        <mx:ArrayCollection>
            <fx:String>Item One</fx:String>
            <fx:String>Second Item</fx:String>
            <fx:String>Item Three</fx:String>
            <fx:String>Last Item</fx:String>
        </mx:ArrayCollection>
    </s:DropDownList>
</s:Application>
```

Interactuando con Listas Spark

La interacción de los componentes es similar a de las listas MX

Evento por defecto en selección de ítems

En MX se lanzan `ListEvent.CHANGE` y `ListEvent.ITEM_CLICK`

En Spark resulta en los tres eventos: `selectionChanging`, `itemFocusChanged` y `selectionChanged`

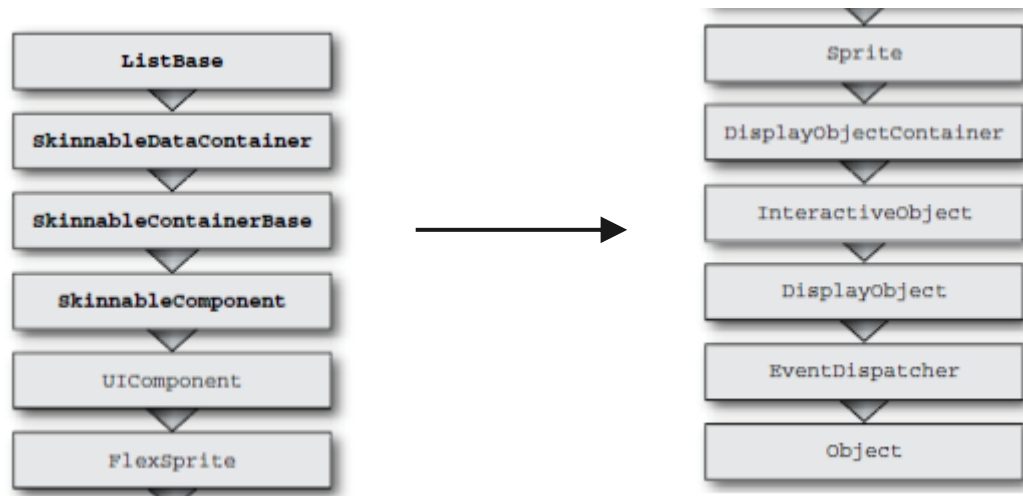
Objeto 'IndexChangeEvent'

Es el objeto que encapsula los tres eventos lanzados por Spark al hacer seleccionar un ítem de la lista

Arquitectura de Listas

Las listas en flex 4 están hechas de las mismas forma que flex 3 por lo que es fácil convertir aplicaciones de una versión a otra.

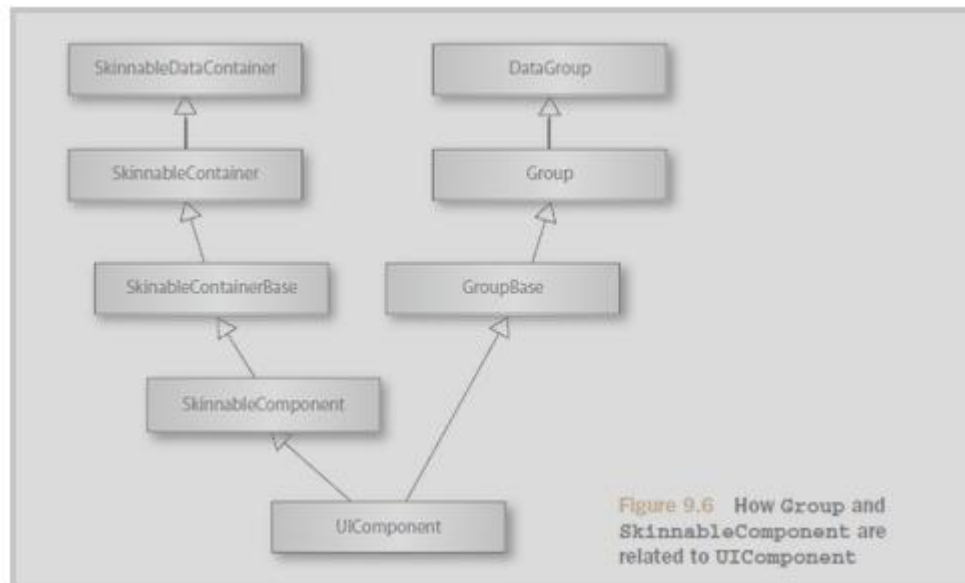
Clases que agrega Spark:



Componentes Custom

Para crear un componente custom (basico) extender de Group, en caso de querer modificar la apariencia utilizar la clase SkinnableContainer.

A continuacion un cuadro con la jerarquia de clases que presenta Spark:



Componentes Custom

Ejemplo de implementación

```
<?xml version="1.0"?>
<s:Panel title="Spark List Component Example"
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  width="75%" height="75%"
  horizontalCenter="0"
  verticalCenter="0">
  <fx:Script>
    <![CDATA[
      import mx.events.IndexChangedEvent;
      private function
selectionChangingHandler(evt:IndexChangedEvent):void
      {
```

Componentes Custom

Ejemplo de implementación

```
var item:* = list.dataProvider.getItemAt(evt.newIndex);
if (item.type != "travel")
{
    evt.preventDefault();
}
}
]]>
</fx:Script>
<s:VGroup left="20" right="20" top="20" bottom="20">
    <s:SimpleText text="Select a title to see the image."/>
    <s:List id="list"
        selectionChanging="selectionChangingHandler(event);"
        width="100%" height="100%" lineHeight="22">
```

Componentes Custom

Ejemplo de implementación

```
<s:layout>
  <s:VerticalLayout/>
</s:layout>
<s:itemRenderer>
  <fx:Component>
    <s:ItemRenderer>
      <s:states>
        <s:State name="normal" />
        <s:State name="hovered" />
        <s:State name="selected" />
      </s:states>
      <s:Rect left="0" right="0" top="0" bottom="0">
        <s:fill>
```

Componentes Custom

Ejemplo de implementación

```
        <s:SolidColor color="0x999999" alpha="0"
                    alpha.hovered="0.2"
                    alpha.selected="0.4" />
    </s:fill>
</s:Rect>
<s:SimpleText id="titleLabel" text="{data.title}"/>
<s:BitmapImage horizontalCenter="80" id="img"
    source="{data.image}"
    includeIn="selected" />
</s:ItemRenderer>
</fx:Component>
</s:itemRenderer>
<s:dataProvider>
```

Componentes Custom

Ejemplo de implementación

```
<s:ArrayList>
    <fx:Object type="travel" title="San Francisco"
        image="images/san_fran.jpg" />
    <fx:Object type="travel" title="San Francisco Lightrail"
        image="images/san_fran_lighrail.jpg" />
    <fx:Object type="travel" title="San Francisco Carriage"
        image="images/san_fran_carriage.jpg" />
</s:ArrayList>
</s:dataProvider>
</s>List>
</s:VGroup>
</s:Panel>
```

Personalización de la visualización de datos

Personalización de la visualización de datos

-
- Componentes basados en listas (List-based)
- Mapeo de Datos
- Array Collection, XMLListCollection
- Funciones de Campo (Label Functions)

Propiedad LabelFields

```
<s:List  
    id="myList"dataProvider="{contactCollection}"  
    labelField="email"  
>
```

Propiedad LabelFunction

- List, Combo-box, Datafields, mx:ArrayCollection
- Forma de Despliegue.
- Mas Funcionamiento aparte de visualizar.
Ej:<s:ButtonBar id="buttonBar"
labelFunction="concatenateName">

Propiedad LabelFunction

Ej de funcion:

```
<s:Application
```

```
.....
```

```
<fx:Script>
```

```
<![CDATA[ private function
```

```
    concatenateName(item:Object):String
```

```
{return item.firstName + " " + item.lastName;}}]]>
```

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

```
.....
```

Tipos de Funciones de etiqueta

- Cantidad de parámetros.
- Columa Simple:
(List, HorizontalaList, TitleList, Trees)
- Columnas Múltiples(fila ,columna):
(Listas Basadas en componentes, DataGrids, Advanced Data Grid, Print Data Grid, File System Data Grid y OLAP Data Grid.)

Funciones de LabelFunctions

- Concatenar(a String)
- Dar Formato(al dato bruto)
- Conversión de Formato(para el usuario)

Items Renderers

- Control Fino Sobre visualización de datos de un elemento.
- Pertenecen a la libreria SPARK
- Declarados como nodos raiz de un elementos renderizable MXML.
- 3 Tipos de elementos renderizables
 - Regular
 - Inline

Items Renderers (Regular)

Spark MXML Item Renderer (Objeto renderizable):

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<s:ItemRenderer 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:SimpleText id="label" paddingBottom="3" paddingTop="20" paddingLeft="25"
```

```
text="{ data.labelText }" />
```

```
<s:BitmapImage id="img" source="{ data.imgSource }"/>
```

```
</s:ItemRenderer> <?xml version="1.0" encoding="utf-8"?>
```

...(Implemento)

```
<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:SkinnableDataContainer itemRenderer="TileGroupIR">
```

```
<s:layout> <s:HorizontalLayout/> </s:layout>
```

```
<mx:ArrayCollection>
```

```
<fx:Object labelText="Kitty" imgSource="assets/IMG_0325.JPG"/>
```

```
<fx:Object labelText="Baseball" imgSource="assets/IMG_0359.JPG"/>
```

```
<fx:Object labelText="4th of July" imgSource="assets/IMG_0425.jpg"/>
```

```
<fx:Object labelText="July 4th Fun" imgSource="assets/IMG_0426.JPG"/>
```

```
</mx:ArrayCollection>
```

```
</s:SkinnableDataContainer>
```

```
</s:Application>
```

Elemento renderizable mas robusto se actualiza solo por un evento

```
<?xml version="1.0" encoding="utf-8"?>
<mx:HBox xmlns:fx="http://ns.adobe.com/mxml/2009" xmlns:s="library://ns.adobe.com/flex/spark" xmlns:mx="library://ns.adobe.com/flex/mx"
width="100%" creationComplete="initDisplay()" dataChange="checkEmail()" >
<fx:Script> <![CDATA[ import flash.net.*;
private function initDisplay() : void
{emailButton.visible = false;}
private function checkEmail() : void
{If( data.email.length > 0 ) emailButton.visible = true;
else
emailButton.visible = false;}
public function sendMail() : void
{var u:URLRequest = new URLRequest( "mailto:" + data.email );
navigateToURL( u, "_self" );
}]]>
</fx:Script>
<s:Button id="emailButton"
visible="false"
label="Send Email"
click="sendMail()"/>
</mx:HBox>
```

In-Line Item Renderer

- La renderizacion ya no es desacoplada como un componente separado, sino que es codificada como un hijo del contenedor padre usando la MXML tag `<itemRenderer/>`.
- Esto nos da buenos resultados en la prototipacion rápida de interfaces de prueba de conceptos, pero no es lo mejor para aplicaciones que pasan a la etapa de Producción.

Columna de una DataGrid con In-Line Item Renderer

EJ:

```
<mx:DataGrid id="dg" width="500" height="100" dataProvider="{myAC}">
    <mx:columns>
        <mx:DataGridColumn headerText="Name">
            <mx:itemRenderer>
                <mx:Component>
                    <mx:Text text="{data.firstName} {data.lastName}"/>
                </mx:Component>
            </mx:itemRenderer>
        </mx:DataGridColumn>
        <mx:DataGridColumn headerText="Email" itemRenderer="myRenderer"/>
    </mx:columns>
</mx:DataGrid>
```

Drop-In item renderer

-
- Algunos componentes que soportan son:
 - Button
 - CheckBox
 - DataFields
 - Image
 - Label
 - Text
 - TextArea
 - TextInPut

Renderizando una imagen de un DataGrid con Drop-in item renderer

```
<?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"
backgroundColor="white" >
<fx:Script><![CDATA[
import mx.collections.ArrayCollection;    [Bindable]
public var myAC:ArrayCollection = new ArrayCollection([
{ name : "Dan Orlando",thumbnail : "img/user.jpg" },{ name : "Tariq Ahmed", thumbnail :
    "img/img1.jpg" },{ name : "John C Bland", thumbnail : "img/img2.jpg" },,]);]]>
</fx:Script><mx:DataGrid id="dg" width="250" height="100" dataProvider="{ myAC }">
<mx:columns>
<mx:DataGridColumn headerText="Name" dataField="name" />
<mx:DataGridColumn headerText="Picture" dataField="thumbnail"
    itemRenderer="mx.controls.Image" /></mx:columns>
</mx:DataGrid>
</s:Application>
```

Item Editor

- Propiedad editable=true
- El usuario activa el evento de edición(al hacer doble click o click sobre el elemento por ejemplo).
- Por defecto no se encuentra habilitada.

Utilizar la edición personalizada en una columna del DataGrid

Componente:

.....Se lee la variable de la lista

[Bindable]

```
public var newEmployeeType:String;

...

Se carga la variable con el tipo

private function init() : void
{
    newEmployeeType = data.type;
    .....

    cambia el valor al actualizar : private function myCB_changeHandler() : void{newEmployeeType = myCB.selectedItem.type;.....
    implementar los métodos...<mx:ComboBox id="myCB" dataProvider="{ typeAC }" labelField="type" change="myCB_changeHandler()" /> </mx:HBox>...
    .....mxml
<s:Application....<fx:Script><![CDATA[[Bindable]public var myAC:ArrayCollection = new ArrayCollection([
    { name:"John C Bland", email:"john@iscool.com",type:"Customer" },
    { name:"Tariq Ahmed", email:"tariq@iscool.com", type:"Employee" },
    { name:"Dan Orlando", email:"dan@iscool.... }
]);]]></fx:Script>
<mx:DataGrid id="dg" dataProvider="{myAC}" editable="true">
<mx:columns><mx:DataGridColumn headerText="Name" dataField="name" />
<mx:DataGridColumn headerText="EMail" dataField="email" editable="false" />
<mx:DataGridColumn headerText="Type" dataField="type" editorDataField="newEmployeeType"
itemEditor="myEditor" /></mx:columns>
</mx:DataGrid>
```

Eventos de Edición

- Se disparan muchos eventos para agregar lógica de negocio, parámetros, limpiar rutinas o agregar puntos de control(check points)
- 3 momentos:
 - **ItemEditBeginning**(Nos paramos sobre el elemento; Agregamos logica de negocio)
 - **ItemEditBegin**(Del evento anterior al editable en el pasaje de datos; Manipular dichos datos)
 - **ItemEditEnd**(El usr termina de editar, a punto de terminar el evento de edición; Comparación y validación del nuevo

Eventos

- **ListEvents**-----**Spark** o ComponentesMXList-Based
- **DataGridEvents**----**DataGrids** con ListEvents
- **AdvancedDataGridEvents**----**AdvancedDataGrids** con ListEvents
- Propiedades de los eventos de un objeto:
 - ColumnIndex
 - CurrentTarget
 - RowIndex
 - Type

Componentes Híbridos

-
- Editables
- Renderizables
- Función, pueden cambiar de :
 - Lectura
 - Lectura-Escritura
 - Escritura-Lectura
- Mantener el estado de edición por cambios rápidos y constantes.

EL QUE SIGUE

Navegaciones

Introducción

-
- Las navegaciones puede describir como la forma en que los diferentes componentes de una aplicación interactúan de forma coherente.

Componentes de Navegación

- Menu
- MenuBar
- ViewStack
- ButtonBar
- TabNavigator
- Accordion

Preparando Datos

-
- Antes de mostrar cómo funcionan estos componentes, veremos distintas formas de preparar los datos para cargar en los componentes.

Elementos para cargar datos.

- Arrays (anidados).
- ArrayCollection (anidados).
- MXML
- Models
 - Componentes XML
 - XML
 - XMLList
 - XMLListCollection

Arrays Anidados

```
protected var menuData:Array =  
[  
    {  
        label:'New',  
        children: [{label:'Task'}, {label:'Request'}, {label:'Person'}]  
    },  
    {  
        label:'Import',  
        children: [{label:'Image'}, {label:'Document'}, {label:'Project'}]  
    }  
];
```

ArrayCollection Anidados

```
import mx.collections.ArrayCollection;
protected var menuData:Array =
[
    {
        label: 'New',
        children: [{label:'Task'}, {label:'Request'}, {label:'Person'}]
    },
    {
        label: 'Import',
        children: [{label:'Image'}, {label:'Document'}, {label:'Project'}]
    }
];
protected var menuCollection:ArrayCollection =
new ArrayCollection(menuData);
```

MXML

- Sintaxis más amigable.
- Alto nivel de anidamiento

```
<s:ArrayCollection id="menuCollection">
  <fx:Array>
    <fx:Object label="'New'">
      <fx:children>
        <fx:Array>
          <fx:Object label="Task" />
          <fx:Object label="Request" />
          <fx:Object label="Person" />
        </fx:Array>
      </fx:children>
    </fx:Object>
    <fx:Object label="'Import'">
      <fx:children>
        <fx:Array>
          <fx:Object label="Image" />
          <fx:Object label="Document" />
          <fx:Object label="Project" />
        </fx:Array>
      </fx:children>
    </fx:Object>
  </fx:Array>
</s:ArrayCollection>
```

Models

- Pueden ser usado como fuente de datos en archivo property.
- Bajo nivel de anidamiento.
- Fácil de codificar

```
<fx:Model id="menuData">
  <menuinfo>
    <menuitem label="Task">
      <children label="Request"/>
      <children label="Person"/>
    </menuitem>
    <menuitem label="Import">
      <children label="Image"/>
      <children label="Document"/>
      <children label="Project"/>
    </menuitem>
  </menuinfo>
</fx:Model>
```

Componentes XML

- Cada nodo se puede nombrar como quiera.
- Bajo nivel de anidamiento.

```
<fx:XML id="menuData">
  <menuinfo>
    <menuitem label="Task">
      <submenu label="Request"/>
      <submenu label="Person"/>
    </menuitem>
    <menuitem label="Import">
      <submenu label="Image"/>
      <submenu label="Document"/>
      <submenu label="Project"/>
    </menuitem>
  </menuinfo>
</fx:XML>
```

XMLListCollection

- Ejemplo donde se usa como fuente de

.....

```
<fx:XML id="config">
  <configData>
    <appData>
      <appName name="My Application"/>
    </appData>
    <menuData>
      <menuitem label="Task">
        <submenu label="Request"/>
        <submenu label="Person"/>
      </menuitem>
    </menuData>
  </configData>
</fx:XML>
<s:XMLListCollection id="menuData"source="{config.menuData.menuitem}"/>
```

Trabajando con Menus

Menus: cargados con XMLListCollection.

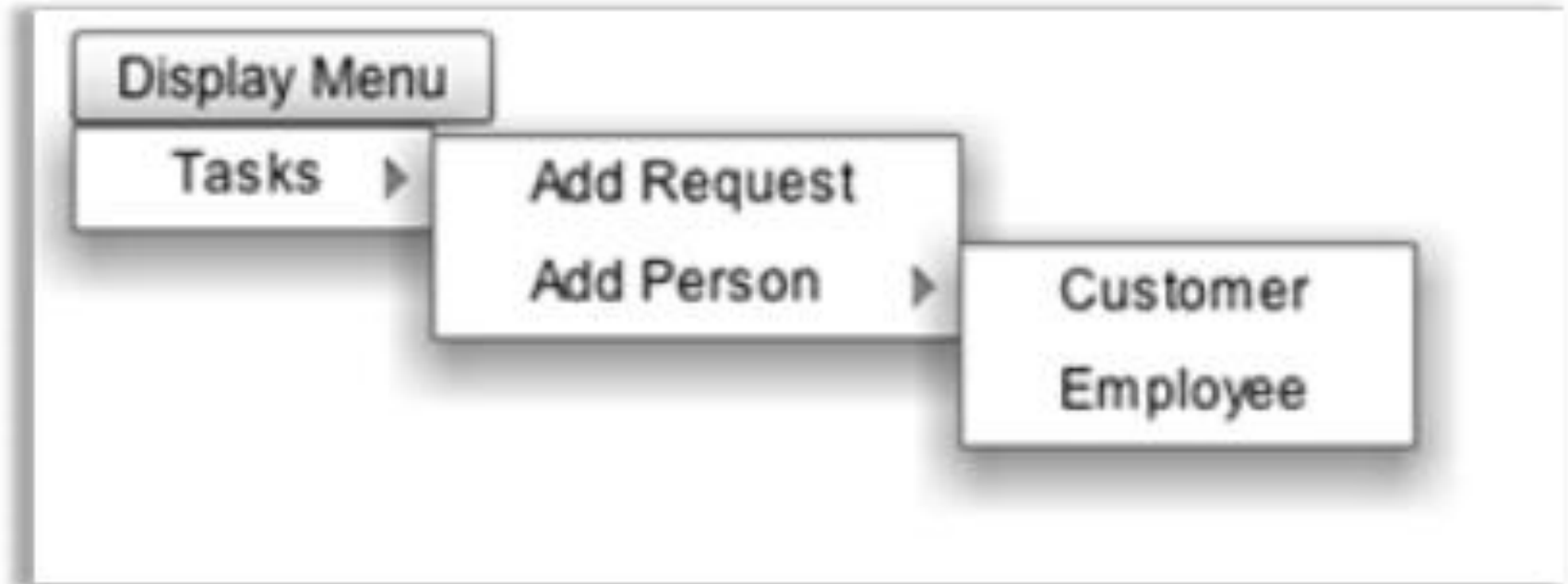
```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo">
  <s:layout>
    <s:VerticalLayout gap="0" />
  </s:layout>
  <fx:Declarations>
    <mx:XMLListCollection id="menuData">
      <mx:source>
        <fx:XMLList>
          <menuItem label="Tasks">
            <submenu label="Add Request"/>
            <submenu label="Add Person">
              <submenu label="Customer"/>
              <submenu label="Employee"/>
            </submenu>
          </menuItem>
        </fx:XMLList>
      </mx:source>
    </mx:XMLListCollection>
  </fx:Declarations>

  <s:Button label="Display Menu" click="menu.show()" />
  <mx:Menu id="menu" showRoot="true" labelField="@label" dataProvider="{menuData}" />
</s:Application>
```

Resultado del componente Menu.

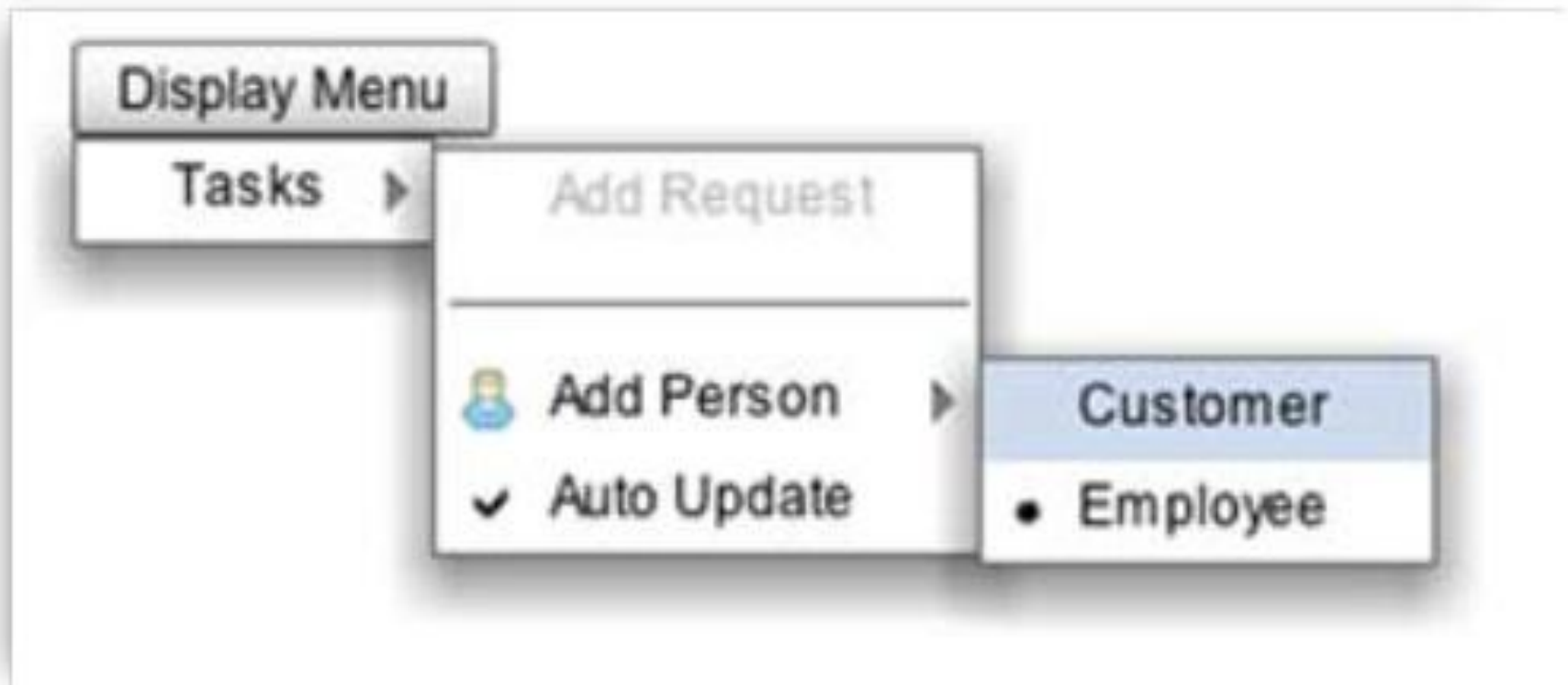
Llamado de la función:

```
<mx:Menu id="menu" showRoot="true" labelField="@label" dataProvider="{menuData}"
```



Customizando items del Menu

- Dar estilos al Menu.



Propiedades

- enabled - boolean
- icon - Class
- label - String
- toggled - boolean
- type - String

Interactuando con el Menu

```
<fx:Script>
  <![CDATA[
    import mx.events.MenuEvent;
    private function onMenuClick(event:MenuEvent):void
    {
      var item:XML = XML(event.item);
      lastEvent.text = "Selection: " + item.@label + ", Position: " + event.index +
        " Type: " + item.@personType;
    }
  ]]>
</fx:Script>
```

```
<mx:Button label="Display Menu" click="menu.show()"/>
```

```
<mx:Menu id="menu" showRoot="true" labelField="@label"
iconField="@icon"dataProvider="{menuData}" itemClick="onMenuClick(event)" />
```

Limitaciones del Menu

- El componente Menu consta de limitaciones debido a que solo podemos procesar un solo nodo raíz.
- Para resolver este problema podemos utilizar el componente MenuBar.

EL QUE SIGUE

MX List

- Listas basadas en componentes de la librería MX
 - ListBase
 - AdvancedListBase

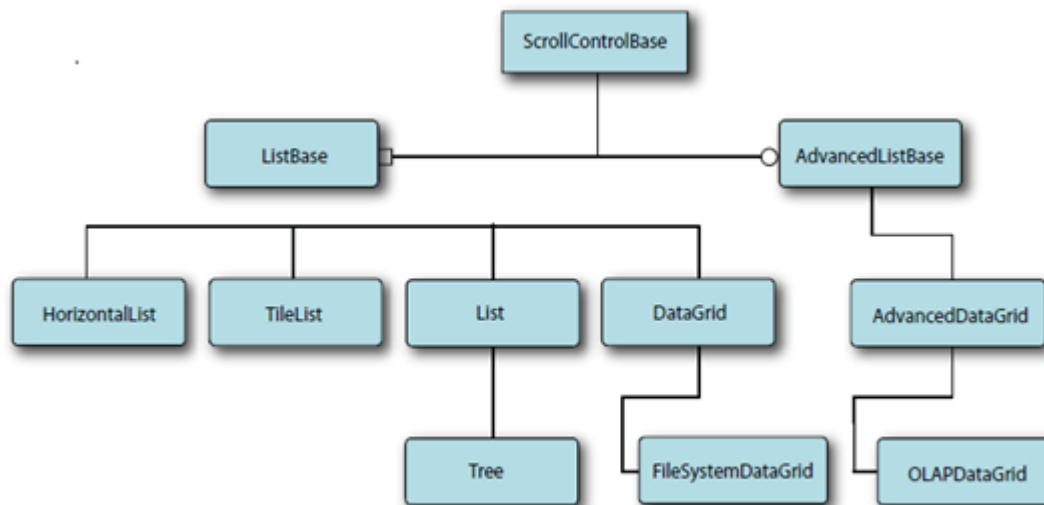


Figure 8.1 The List-based family of MX components and how they're related

Uso de Listas con la librería Spark

Introducción

En el capítulo anterior se vio cómo utilizar las listas simple de flex, ahora veremos cómo utilizarlas con la librería Spark.

La librería de Spark permite mas posibilidades de escalabilidad, por ejemplo, permite que un usuario avanzado de Flex modifique el diseño de la interfaz.

Genealogía

Como se vio en el capítulo 8 los componentes de la librería MX son derivados de ListBase y AdvancedListBase. En Spark el componente extiende de la ListBase de Spark ubicada en el paquete `spark.components.supportClasses` y esta a su vez extiende de `SkinnableDataContainer`.

Lo importante aquí es comprender que cada clase hereda de una clase ListBase propia de su librería sea MX o sea Spark.

Genealogía

Identificando el componente correspondiente

Flex automaticamente sabe cual de los namespaces usar.

Si en el mxml se encuentra el tag `<mx:List/>` usara la ListBase de la librería MX.

Si por el contrario utiliza `<s:List/>` usara la clase ListBase de la librería de Spark.

Genealogía

Elementos renderizados con las listas spark

En MX y Spark List extends ListBas.

En MX ListBase extiende de ScrollControlBase la cual extiende de UIComponent.

Por el contrario la ListBase de Spark extiende de SkinnableDataContainer la clase que es responsable de mostrar la información de los items usando item renderers.

Listas Spark - Controles

No es difícil crear una custom List in flex 4

Es tan fácil como versiones anteriores del SDK, la mejor forma es extender la clase ListBase de Spark o la clase SKinnableDataContainer.

Listas Spark - Controles

Control ButtonBar

El control ButtonBar es similar a el ToggleButtonBar de la librería MX.

Este control es Básicamente es una Lista ya que extiende de la clase ListBase

Es util en la creación de un menu es facil de crear usando el control de Spark Button-Bar.

Listas Spark - Controles

Control ButtonBar

```
<?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"
    minWidth="1024" minHeight="768">
    <s:layout>
        <s:VerticalLayout paddingLeft="20" paddingTop="20"/>
    </s:layout>
    <s:ButtonBar>
        <mx:ArrayCollection>
            <fx:String>Button One</fx:String>
            <fx:String>Second Button</fx:String>
            <fx:String>Button Three</fx:String>
            <fx:String>Last Button</fx:String>
        </mx:ArrayCollection>
    </s:ButtonBar>
</s:Application>
```

Listas Spark - Controles

Control Spark List

```
<?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"
    xmlns:comp="com.danorlando.components.*"
    minWidth="600" minHeight="450">
    <s:layout>
        <s:VerticalLayout gap="1" useVirtualLayout="true" />
    </s:layout>
    <s>List>
        <s:dataProvider>
            <mx:ArrayCollection>
                <fx:String>Menu Item 1</fx:String>
                <fx:String>Second Menu Item</fx:String>
                <fx:String>Third Menu Item</fx:String>
                <fx:String>Fourth Menu Item</fx:String>
                <fx:String>Last Menu Item</fx:String>
            </mx:ArrayCollection>
        </s:dataProvider>
    </s>List>
</s:Application>
```

Listas Spark - Controles

Control DropDown

```
<?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"
    minWidth="1024" minHeight="768">
    <s:layout>
        <s:VerticalLayout paddingLeft="20" paddingTop="20"/>
    </s:layout>
    <s:DropDownList width="200">
        <mx:ArrayCollection>
            <fx:String>Item One</fx:String>
            <fx:String>Second Item</fx:String>
            <fx:String>Item Three</fx:String>
            <fx:String>Last Item</fx:String>
        </mx:ArrayCollection>
    </s:DropDownList>
</s:Application>
```

Listas Spark - Controles

Control DropDown

```
<?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"
    minWidth="1024" minHeight="768">
    <s:layout>
        <s:VerticalLayout paddingLeft="20" paddingTop="20"/>
    </s:layout>
    <s:DropDownList width="200">
        <mx:ArrayCollection>
            <fx:String>Item One</fx:String>
            <fx:String>Second Item</fx:String>
            <fx:String>Item Three</fx:String>
            <fx:String>Last Item</fx:String>
        </mx:ArrayCollection>
    </s:DropDownList>
</s:Application>
```

Interactuando con Listas Spark

La interacción de los componentes es similar a de las listas MX

Evento por defecto en selección de ítems

En MX se lanzan `ListEvent.CHANGE` y `ListEvent.ITEM_CLICK`

En Spark resulta en los tres eventos: `selectionChanging`, `itemFocusChanged` y `selectionChanged`

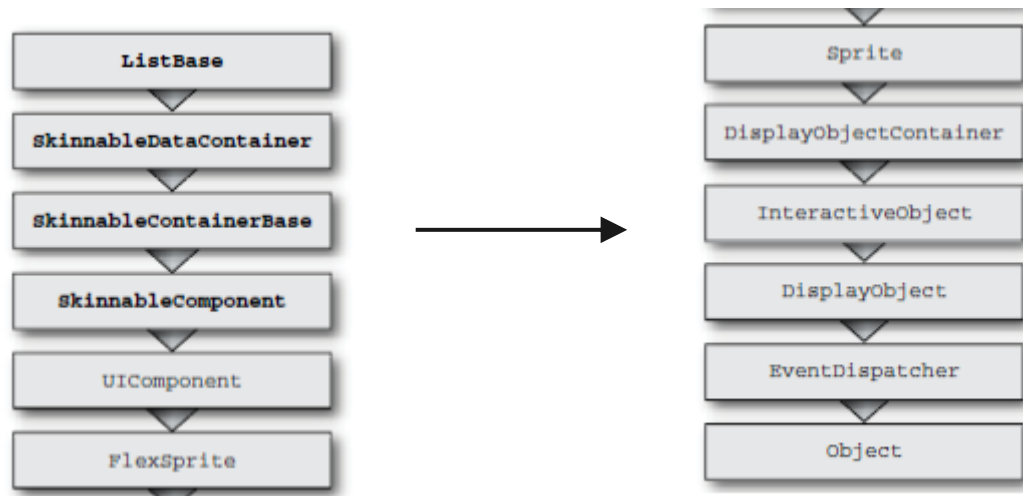
Objeto 'IndexChangeEvent'

Es el objeto que encapsula los tres eventos lanzados por Spark al hacer seleccionar un ítem de la lista

Arquitectura de Listas

Las listas en flex 4 están hechas de las mismas forma que flex 3 por lo que es fácil convertir aplicaciones de una versión a otra.

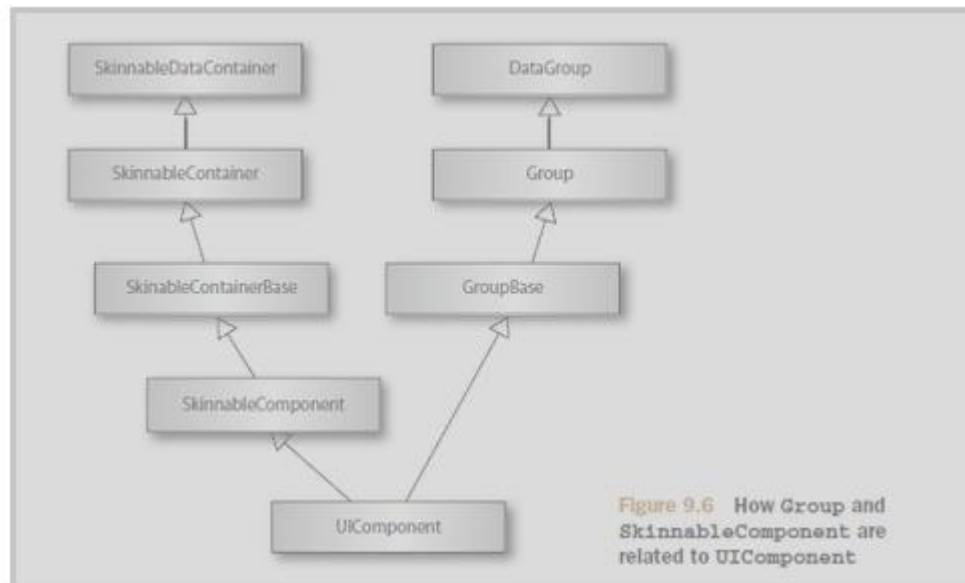
Clases que agrega Spark:



Componentes Custom

Para crear un componente custom (basico) extender de Group, en caso de querer modificar la apariencia utilizar la clase SkinnableContainer.

A continuacion un cuadro con la jerarquia de clases que presenta Spark:



Componentes Custom

Ejemplo de implementación

```
<?xml version="1.0"?>
<s:Panel title="Spark List Component Example"
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  width="75%" height="75%"
  horizontalCenter="0"
  verticalCenter="0">
  <fx:Script>
    <![CDATA[
      import mx.events.IndexChangedEvent;
      private function
selectionChangingHandler(evt:IndexChangedEvent):void
      {
```

Componentes Custom

Ejemplo de implementación

```
var item:* = list.dataProvider.getItemAt(evt.newIndex);
if (item.type != "travel")
{
    evt.preventDefault();
}
}
]]>
</fx:Script>
<s:VGroup left="20" right="20" top="20" bottom="20">
    <s:SimpleText text="Select a title to see the image."/>
    <s:List id="list"
        selectionChanging="selectionChangingHandler(event);"
        width="100%" height="100%" lineHeight="22">
```

Componentes Custom

Ejemplo de implementación

```
<s:layout>
  <s:VerticalLayout/>
</s:layout>
<s:itemRenderer>
  <fx:Component>
    <s:ItemRenderer>
      <s:states>
        <s:State name="normal" />
        <s:State name="hovered" />
        <s:State name="selected" />
      </s:states>
      <s:Rect left="0" right="0" top="0" bottom="0">
        <s:fill>
```

Componentes Custom

Ejemplo de implementación

```
<s:SolidColor color="0x999999" alpha="0"
              alpha.hovered="0.2"
              alpha.selected="0.4" />
</s:fill>
</s:Rect>
<s:SimpleText id="titleLabel" text="{data.title}"/>
<s:BitmapImage horizontalCenter="80" id="img"
              source="{data.image}"
              includeIn="selected" />
</s:ItemRenderer>
</fx:Component>
</s:itemRenderer>
<s:dataProvider>
```

Componentes Custom

Ejemplo de implementación

```
<s:ArrayList>
    <fx:Object type="travel" title="San Francisco"
        image="images/san_fran.jpg" />
    <fx:Object type="travel" title="San Francisco Lightrail"
        image="images/san_fran_lightrail.jpg" />
    <fx:Object type="travel" title="San Francisco Carriage"
        image="images/san_fran_carriage.jpg" />
</s:ArrayList>
</s:dataProvider>
</s>List>
</s:VGroup>
</s:Panel>
```

Personalización de la visualización de datos

Personalización de la visualización de datos

-
- Componentes basados en listas (List-based)
- Mapeo de Datos
- Array Collection, XMLListCollection
- Funciones de Campo (Label Functions)

Propiedad LabelFields

```
<s:List  
    id="myList"dataProvider="{contactCollection}"  
    labelField="email"  
>
```

Propiedad LabelFunction

- List, Combo-box, Datafields, mx:ArrayCollection
- Forma de Despliegue.
- Mas Funcionamiento aparte de visualizar.
Ej:<s:ButtonBar id="buttonBar"
labelFunction="concatenateName">

Propiedad LabelFunction

Ej de funcion:

```
<s:Application
```

```
.....
```

```
<fx:Script>
```

```
<![CDATA[ private function
```

```
    concatenateName(item:Object):String
```

```
{return item.firstName + " " + item.lastName;}}]]>
```

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

```
.....
```

Tipos de Funciones de etiqueta

- Cantidad de parámetros.
- Columa Simple:
(List, HorizontalaList, TitleList, Trees)
- Columnas Múltiples(fila ,columna):
(Listas Basadas en componentes, DataGrids, Advanced Data Grid, Print Data Grid, File System Data Grid y OLAP Data Grid.)

Funciones de LabelFunctions

- Concatenar(a String)
- Dar Formato(al dato bruto)
- Conversión de Formato(para el usuario)

Items Renderers

- Control Fino Sobre visualización de datos de un elemento.
- Pertenecen a la libreria SPARK
- Declarados como nodos raiz de un elementos renderizable MXML.
- 3 Tipos de elementos renderizables
 - Regular
 - Inline

Items Renderers (Regular)

Spark MXML Item Renderer (Objeto renderizable):

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<s:ItemRenderer 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:SimpleText id="label" paddingBottom="3" paddingTop="20" paddingLeft="25"
```

```
text="{ data.labelText }" />
```

```
<s:BitmapImage id="img" source="{ data.imgSource }"/>
```

```
</s:ItemRenderer> <?xml version="1.0" encoding="utf-8"?>
```

...(Implemento)

```
<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:SkinnableDataContainer itemRenderer="TileGroupIR">
```

```
<s:layout> <s:HorizontalLayout/> </s:layout>
```

```
<mx:ArrayCollection>
```

```
<fx:Object labelText="Kitty" imgSource="assets/IMG_0325.JPG"/>
```

```
<fx:Object labelText="Baseball" imgSource="assets/IMG_0359.JPG"/>
```

```
<fx:Object labelText="4th of July" imgSource="assets/IMG_0425.jpg"/>
```

```
<fx:Object labelText="July 4th Fun" imgSource="assets/IMG_0426.JPG"/>
```

```
</mx:ArrayCollection>
```

```
</s:SkinnableDataContainer>
```

```
</s:Application>
```

Elemento renderizable mas robusto se actualiza solo por un evento

```
<?xml version="1.0" encoding="utf-8"?>
<mx:HBox xmlns:fx="http://ns.adobe.com/mxml/2009" xmlns:s="library://ns.adobe.com/flex/spark" xmlns:mx="library://ns.adobe.com/flex/mx"
width="100%" creationComplete="initDisplay()" dataChange="checkEmail()" >
<fx:Script> <![CDATA[ import flash.net.*;
private function initDisplay() : void
{emailButton.visible = false;}

private function checkEmail() : void
{If( data.email.length > 0 ) emailButton.visible = true;
else
emailButton.visible = false;}

public function sendMail() : void
{var u:URLRequest = new URLRequest( "mailto:" + data.email );
navigateToURL( u, "_self" );
}]]>
</fx:Script>
<s:Button id="emailButton"
visible="false"
label="Send Email"
click="sendMail()"/>
</mx:HBox>
```

In-Line Item Renderer

- La renderizacion ya no es desacoplada como un componente separado, sino que es codificada como un hijo del contenedor padre usando la MXML tag `<itemRenderer/>`.
- Esto nos da buenos resultados en la prototipacion rápida de interfaces de prueba de conceptos, pero no es lo mejor para aplicaciones que pasan a la etapa de Producción.

Columna de una DataGrid con In-Line Item Renderer

EJ:

```
<mx:DataGrid id="dg" width="500" height="100" dataProvider="{myAC}">
    <mx:columns>
        <mx:DataGridColumn headerText="Name">
            <mx:itemRenderer>
                <mx:Component>
                    <mx:Text text="{data.firstName} {data.lastName}"/>
                </mx:Component>
            </mx:itemRenderer>
        </mx:DataGridColumn>
        <mx:DataGridColumn headerText="Email" itemRenderer="myRenderer"/>
    </mx:columns>
</mx:DataGrid>
```

Drop-In item renderer

-
- Algunos componentes que soportan son:
 - Button
 - CheckBox
 - DataFields
 - Image
 - Label
 - Text
 - TextArea
 - TextInPut

Renderizando una imagen de un DataGrid con Drop-in item renderer

```
<?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"
backgroundColor="white" >
<fx:Script><![CDATA[
import mx.collections.ArrayCollection;    [Bindable]
public var myAC:ArrayCollection = new ArrayCollection([
{ name : "Dan Orlando",thumbnail : "img/user.jpg" },{ name : "Tariq Ahmed", thumbnail :
    "img/img1.jpg" },{ name : "John C Bland", thumbnail : "img/img2.jpg" },,]);]]>
</fx:Script><mx:DataGrid id="dg" width="250" height="100" dataProvider="{ myAC }">
<mx:columns>
<mx:DataGridColumn headerText="Name" dataField="name" />
<mx:DataGridColumn headerText="Picture" dataField="thumbnail"
    itemRenderer="mx.controls.Image" /></mx:columns>
</mx:DataGrid>
</s:Application>
```

Item Editor

- Propiedad editable=true
- El usuario activa el evento de edición(al hacer doble click o click sobre el elemento por ejemplo).
- Por defecto no se encuentra habilitada.

Utilizar la edición personalizada en una columna del DataGrid

Componente:

.....Se lee la variable de la lista

[Bindable]

```
public var newEmployeeType:String;

...

Se carga la variable con el tipo

private function init() : void
{
    newEmployeeType = data.type;
    .....

    cambia el valor al actualizar : private function myCB_changeHandler() : void{newEmployeeType = myCB.selectedItem.type;.....
    implementar los métodos...<mx:ComboBox id="myCB" dataProvider="{ typeAC }" labelField="type" change="myCB_changeHandler()" /> </mx:HBox>...
    .....mxaml
<s:Application....<fx:Script><![CDATA[[Bindable]public var myAC:ArrayCollection = new ArrayCollection([ { name:"John C Bland",
    email:"john@iscool.com",type:"Customer" }, { name:"Tariq Ahmed", email:"tariq@iscool.com", type:"Employee" }, { name:"Dan Orlando",
    email:"dan@iscool.... }
]);]]></fx:Script>

<mx:DataGrid id="dg" dataProvider="{myAC}" editable="true">

<mx:columns><mx:DataGridColumn headerText="Name" dataField="name" /><mx:DataGridColumn headerText="EMail" dataField="email" editable="false" />
<mx:DataGridColumn headerText="Type" dataField="type" editorDataField="newEmployeeType"
itemEditor="myEditor" /></mx:columns>

</mx:DataGrid>
```

Eventos de Edición

- Se disparan muchos eventos para agregar lógica de negocio, parámetros, limpiar rutinas o agregar puntos de control(check points)
- 3 momentos:
 - **ItemEditBeginning**(Nos paramos sobre el elemento; Agregamos logica de negocio)
 - **ItemEditBegin**(Del evento anterior al editable en el pasaje de datos; Manipular dichos datos)
 - **ItemEditEnd**(El usr termina de editar, a punto de terminar el evento de edición; Comparación y validación del nuevo

Eventos

- **ListEvents**-----**Spark** o ComponentesMXList-Based
- **DataGridEvents**----**DataGrids** con ListEvents
- **AdvancedDataGridEvents**----**AdvancedDataGrids** con ListEvents
- Propiedades de los eventos de un objeto:
 - ColumnIndex
 - CurrentTarget
 - RowIndex
 - Type

Componentes Híbridos

-
- Editables
- Renderizables
- Función, pueden cambiar de :
 - Lectura
 - Lectura-Escritura
 - Escritura-Lectura
- Mantener el estado de edición por cambios rápidos y constantes.

EL QUE SIGUE

Navegaciones

Introducción

-
- Las navegaciones puede describir como la forma en que los diferentes componentes de una aplicación interactúan de forma coherente.

Componentes de Navegación

- Menu
- MenuBar
- ViewStack
- ButtonBar
- TabNavigator
- Accordion

Preparando Datos

-
- Antes de mostrar cómo funcionan estos componentes, veremos distintas formas de preparar los datos para cargar en los componentes.

Elementos para cargar datos.

- Arrays (anidados).
- ArrayCollection (anidados).
- MXML
- Models
 - Componentes XML
 - XML
 - XMLList
 - XMLListCollection

Arrays Anidados

```
protected var menuData:Array =  
[  
    {  
        label:'New',  
        children: [{label:'Task'}, {label:'Request'}, {label:'Person'}]  
    },  
    {  
        label:'Import',  
        children: [{label:'Image'}, {label:'Document'}, {label:'Project'}]  
    }  
];
```

ArrayCollection Anidados

```
import mx.collections.ArrayCollection;
protected var menuData:Array =
[
    {
        label: 'New',
        children: [{label:'Task'}, {label:'Request'}, {label:'Person'}]
    },
    {
        label: 'Import',
        children: [{label:'Image'}, {label:'Document'}, {label:'Project'}]
    }
];
protected var menuCollection:ArrayCollection =
new ArrayCollection(menuData);
```

MXML

- Sintaxis más amigable.
- Alto nivel de anidamiento

```
<s:ArrayCollection id="menuCollection">
  <fx:Array>
    <fx:Object label="'New'">
      <fx:children>
        <fx:Array>
          <fx:Object label="Task" />
          <fx:Object label="Request" />
          <fx:Object label="Person" />
        </fx:Array>
      </fx:children>
    </fx:Object>
    <fx:Object label="'Import'">
      <fx:children>
        <fx:Array>
          <fx:Object label="Image" />
          <fx:Object label="Document" />
          <fx:Object label="Project" />
        </fx:Array>
      </fx:children>
    </fx:Object>
  </fx:Array>
</s:ArrayCollection>
```

Models

- Pueden ser usado como fuente de datos en archivo property.
- Bajo nivel de anidamiento.
- Fácil de codificar

```
<fx:Model id="menuData">
  <menuinfo>
    <menuitem label="Task">
      <children label="Request"/>
      <children label="Person"/>
    </menuitem>
    <menuitem label="Import">
      <children label="Image"/>
      <children label="Document"/>
      <children label="Project"/>
    </menuitem>
  </menuinfo>
</fx:Model>
```

Componentes XML

- Cada nodo se puede nombrar como quiera.
- Bajo nivel de anidamiento.

```
<fx:XML id="menuData">
  <menuinfo>
    <menuitem label="Task">
      <submenu label="Request"/>
      <submenu label="Person"/>
    </menuitem>
    <menuitem label="Import">
      <submenu label="Image"/>
      <submenu label="Document"/>
      <submenu label="Project"/>
    </menuitem>
  </menuinfo>
</fx:XML>
```

XMLListCollection

- Ejemplo donde se usa como fuente de

.....

```
<fx:XML id="config">
  <configData>
    <appData>
      <appName name="My Application"/>
    </appData>
    <menuData>
      <menuitem label="Task">
        <submenu label="Request"/>
        <submenu label="Person"/>
      </menuitem>
    </menuData>
  </configData>
</fx:XML>
<s:XMLListCollection id="menuData"source="{config.menuData.menuitem}"/>
```

Trabajando con Menus

Menus: cargados con XMLListCollection.

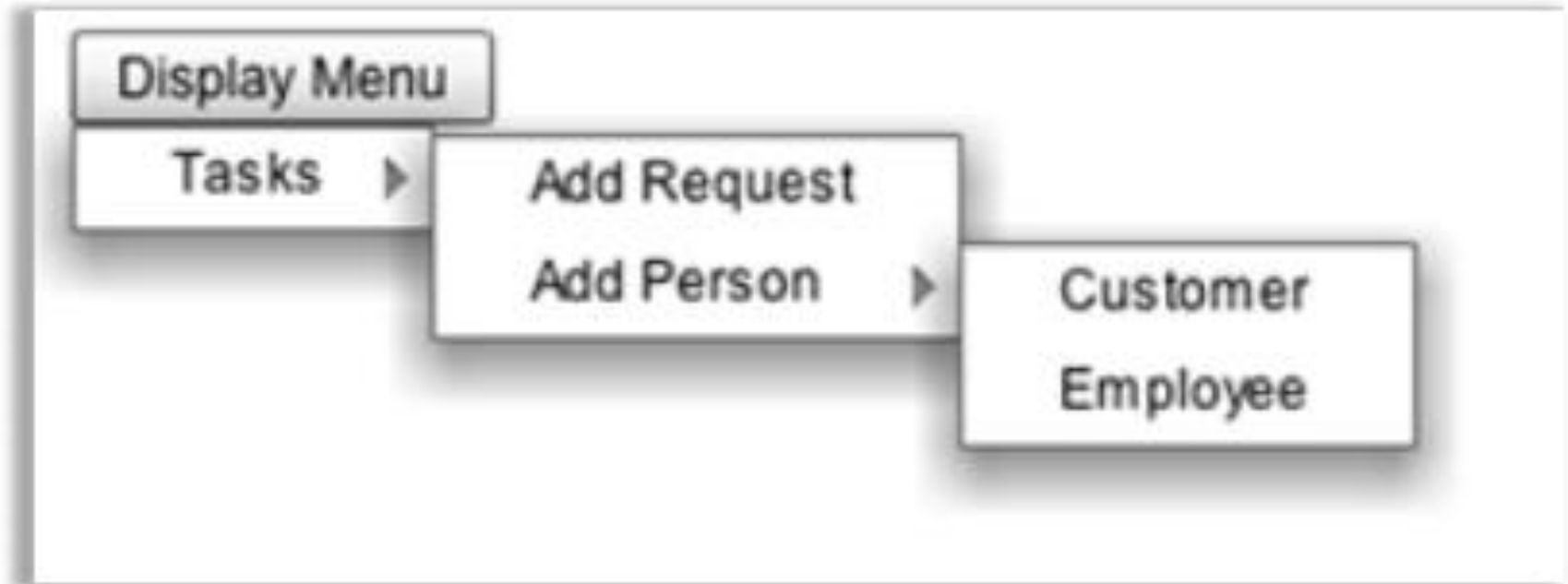
```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo">
  <s:layout>
    <s:VerticalLayout gap="0" />
  </s:layout>
  <fx:Declarations>
    <mx:XMLListCollection id="menuData">
      <mx:source>
        <fx:XMLList>
          <menuItem label="Tasks">
            <submenu label="Add Request"/>
            <submenu label="Add Person">
              <submenu label="Customer"/>
              <submenu label="Employee"/>
            </submenu>
          </menuItem>
        </fx:XMLList>
      </mx:source>
    </mx:XMLListCollection>
  </fx:Declarations>

  <s:Button label="Display Menu" click="menu.show()" />
  <mx:Menu id="menu" showRoot="true" labelField="@label" dataProvider="{menuData}" />
</s:Application>
```

Resultado del componente Menu.

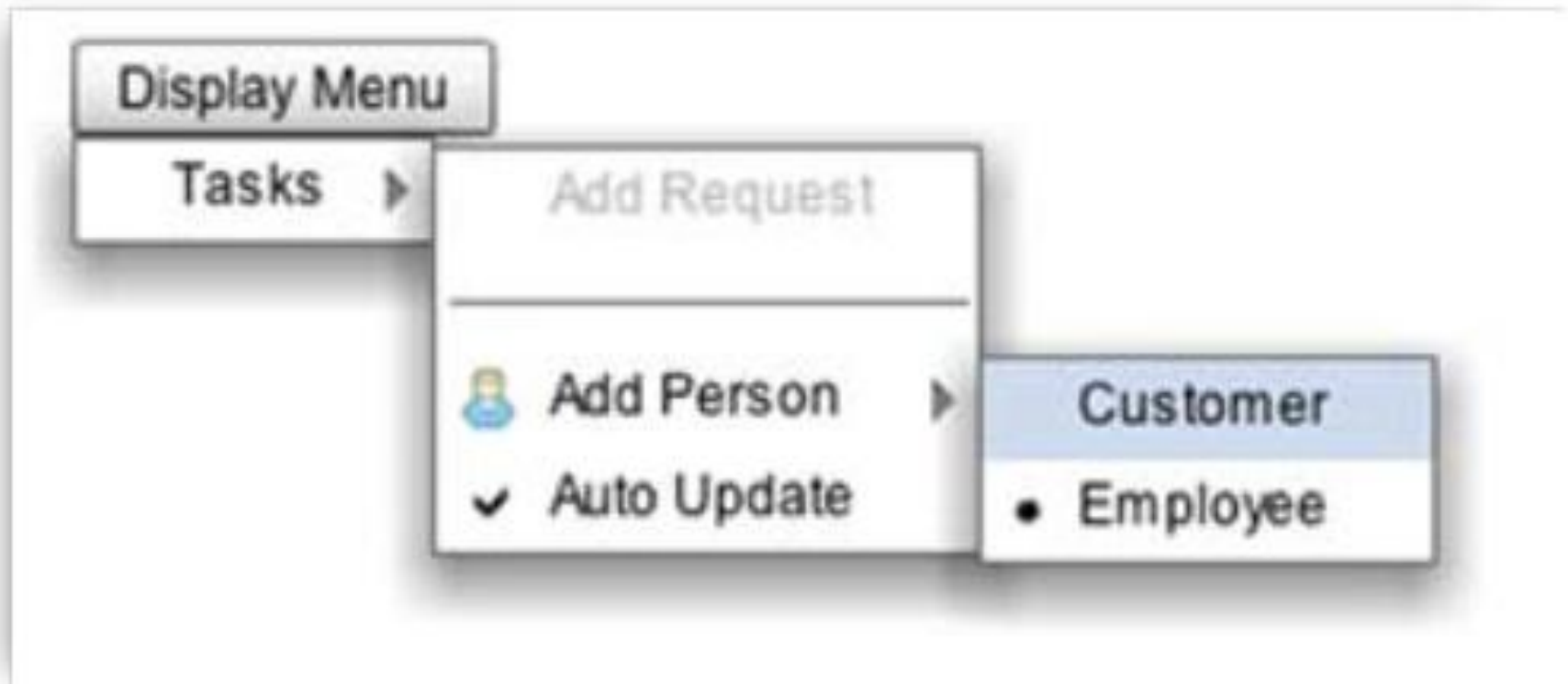
Llamado de la función:

```
<mx:Menu id="menu" showRoot="true" labelField="@label" dataProvider="{menuData}"
```



Customizando items del Menu

- Dar estilos al Menu.



Propiedades

- enabled - boolean
- icon - Class
- label - String
- toggled - boolean
- type - String

Interactuando con el Menu

```
<fx:Script>
    <![CDATA[
        import mx.events.MenuEvent;
        private function onMenuClick(event:MenuEvent):void
        {
            var item:XML = XML(event.item);
            lastEvent.text = "Selection: " + item.@label + ", Position: " + event.index +
                " Type: " + item.@personType;
        }
    ]]>
</fx:Script>
```

```
<mx:Button label="Display Menu" click="menu.show()"/>
```

```
<mx:Menu id="menu" showRoot="true" labelField="@label"
iconField="@icon"dataProvider="{menuData}" itemClick="onMenuClick(event)" />
```

Limitaciones del Menu

- El componente Menu consta de limitaciones debido a que solo podemos procesar un solo nodo raíz.
- Para resolver este problema podemos utilizar el componente MenuBar.

EL QUE SIGUE