

Flex 4 en Acción

Resumen de la sección 12.3 a 13.2

12.3 Creando una barra de menú

La barra de menú se expande desde el menú para permitir varios niveles. Los menús se posicionan lado a lado, algo común en la mayoría de los diseños. Es una forma clara para que los usuarios puedan navegar a través de la aplicación sin tener que salir de la sección en la que se encuentran.

Más allá de las diferencias visuales obvias, la principal diferencia entre un menú común y una barra de menú es que la barra de menú no fue pensada para ser un mecanismo de pop-up. Está diseñada para que la aplicación Flex ofrezca el mismo tipo de funcionalidad que lo haría una aplicación de escritorio.

12.3.1 Creando una barra de menú

A continuación se crea una barra de menú simple. Notar las diferencias entre los distintos niveles del menú.

```
<?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">
  <fx:Declarations>
    <s:XMLListCollection id="menuData">
      <fx:XMLList>
        <menuItem label="File">
          <submenu label="New">
            <submenu label="Project"/>
            <submenu label="Request"/>
          </submenu>
          <submenu label="Print"/>
        </menuItem>
        <menuItem label="View">
          <submenu label="Users"/>
          <submenu label="Reports"/>
        </menuItem>
      </fx:XMLList>
    </s:XMLListCollection>
  </fx:Declarations>
  <mx:MenuBar id="menuBar" labelField="@label" dataProvider="{menuData}" />
</s:Application>
```

Luego de compilar y ejecutar el código anterior podrá ver un menú rudimentario.

A continuación mejoraremos dicho menú y pasaremos a buscarle una posición para ubicarlo.

12.3.2 Posicionando la barra de menú

Posicionar la barra de menú no es diferente que posicionar cualquier otro componente. Las reglas del layout están determinadas por el layout del contenedor padre o por las propiedades x e y. El siguiente pedazo de código posicionaría la barra de menú 10 píxeles a la izquierda y 20 píxeles de la parte de arriba.

```
<mx:MenuBar id="menuBar" x="10" y="20" labelField="@label"
dataProvider="{menuData}" />
```

12.3.3 Personalizando elementos de la barra de menú

Para realizar esto se deben de seguir las mismas reglas que sobre el menú común. Por ejemplo se pueden agregar radio buttons o iconos a una barra de menú como se haría sobre un menú normal.

A continuación se usa una estructura XML para personalizar los elementos del menú como se mencionó anteriormente.

```
<?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 />
</s:layout>
<fx:Script>
<![CDATA[
[Bindable]
[Embed(source="assets/icons/user.png")]
public var userIcon:Class;
]]>
</fx:Script>
<fx:Declarations>
<s:XMLListCollection id="menuData">
<fx:XMLList>
<menuitem label="File">
<submenu label="New Task">
<submenu label="Add Request" enabled="false"/>
<submenu type="separator"/>
<submenu label="Add Person" icon="userIcon">
<submenu label="Customer" type="radio" groupName="persons"/>
<submenu label="Employee" type="radio" groupName="persons"
toggled="true"/>
</submenu>
<submenu label="Auto Update" type="check" toggled="true"/>
</submenu>
<submenu label="Print"/>
</menuitem>
<menuitem label="View">
<submenu label="Users"/>
<submenu label="Reports"/>
</menuitem>
</fx:XMLList>
</s:XMLListCollection>
</fx:Declarations>
<mx:MenuBar id="menuBar" labelField="@label" iconField="@icon"
dataProvider="{menuData}" />
```

12.3.4 Manejando la interacción del usuario con la barra de menú

Los mismos eventos que se utilizan con un menú se utilizan con la barra de menú. A continuación se pasa a listarlos.

`change` El usuario cambia la selección actual.
`itemClick` El usuario hace clic y habilita un elemento del menú.
`itemRollOut` El puntero del mouse se aleja del elemento del menú.
`itemRollOver` El puntero del mouse se mueve sobre un elemento del menú.
`menuHide` Se cierra un menú o submenú.
`menuShow` Se abre un menú o submenú.

```
<?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 />
</s:layout>
<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;
}
]>
</fx:Script>
<fx:Declarations>
<s:XMLListCollection id="menuData">
<fx:XMLList>
<menuitem label="File">
<submenu label="New Task">
<submenu label="Add Request" enabled="false"/>
<submenu type="separator"/>
<submenu label="Add Person" icon="userIcon">
<submenu label="Customer" type="radio" groupName="persons"/>
<submenu label="Employee" type="radio" groupName="persons"
toggled="true"/>
</submenu>
<submenu label="Auto Update" type="check" toggled="true"/>
</submenu>
<submenu label="Print"/>
</menuitem>
<menuitem label="View">
<submenu label="Users"/>
<submenu label="Reports"/>
</menuitem>
</fx:XMLList>
</s:XMLListCollection>
</fx:Declarations>
<mx:MenuBar id="menuBar" labelField="@label" iconField="@icon"
dataProvider="{menuData}" itemClick="onMenuClick(event)" />
<mx:Spacer height="10" />
<mx:Text id="lastEvent"/>
</s:Application>
```

El código anterior muestra como al hacer clic sobre un elemento del menú se modifica la información que se está mostrando.

12.4 Usando Vistas en Pilas (View Stacks)

Una vista en pilas es un mecanismo interesante para navegar a través de la aplicación. Trabaja apilando los contenedores hijos uno sobre los otros. Es como un mazo de cartas, se puede seleccionar cualquier carta del mazo en cualquier momento.

Es un mecanismo sencillo comparado con otros contenedores de navegación Flex.

La única desventaja es que una vista en pila no tiene ninguna interfaz de usuario para controlar la navegación del usuario. Pero esto se puede resolver de manera sencilla.

12.4.1 Creando una Vista en Pilas

A continuación se muestra un ejemplo sencillo.

```
<?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">
  <mx:ViewStack id="stack" width="100%"
height="100%">
    <s:NavigatorContent label="TVs" width="100%" height="100%">
      <s:Label text="TV Content" />
    </s:NavigatorContent>
    <s:NavigatorContent label="Cameras" width="100%" height="100%">
      <s:Label text="Camera Content" />
    </s:NavigatorContent>
    <s:NavigatorContent label="Computers" width="100%" height="100%">
      <s:Label text="Computer Content" />
    </s:NavigatorContent>
  </mx:ViewStack>
</s:Application>
```

Cada uno de los contenedores hijos es un NavigatorContent. El NavigatorContent como e sun contenedor tiene la propiedad layout. Es el equivalente a Group.

No se usa Group porque los hijos dentro de un viewStack necesitan ser instanciados. Dando acceso al desarrollador a la creación de hijos de los contenedores hijos (varios niveles).

Ejemplo de elemento personalizado

```
<?xml version="1.0" encoding="utf-8"?>
<s:NavigatorContent xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/halo"
width="100%" height="100%">
  <s:Panel title="TVs" width="100%" height="100%">
    <s:layout>
      <s:VerticalLayout paddingLeft="20" paddingTop="20"
paddingRight="20" paddingBottom="20" />
    </s:layout>
    <s:HGroup width="100%" enabled="false">
      <s:TextInput width="100%" />
      <s:Button label="Search for TVs"/>
    </s:HGroup>
    <mx:HRule width="100%" />
    <mx:Spacer height="10" />
    <s:Label text="There are no TVs available." />
  </s:Panel>
</s:NavigatorContent>
```

Usando TVView.mxml como contenedor en el View Stack

```
<?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"
xmlns:views="views.*">
  <mx:ViewStack x="20" y="20" width="400" height="200">
    <views:TVView />
    <views:ComputersView />
  </mx:ViewStack>
</s:Application>
```

12.4.2 Agregar Navegación a la vista en Pila

La navegación no es automática. Se necesita especificar que contenedor es el visible en ese momento. Flex 4 proporciona la posibilidad de personalizar el Button-bar como se desee. Para ver esto en detalle consultar el capítulo 20.

Agregar Navegación

```
<?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"
xmlns:views="views.*">
<s:layout>
<s:VerticalLayout paddingLeft="20" paddingTop="10" paddingRight="20" />
</s:layout>
<s:ButtonBar dataProvider="{stack}" />
<mx:ViewStack id="stack" width="400" height="200">
<views:TVView label="Buy TVs" />
<views:ComputersView label="Buy Computers" />
</mx:ViewStack>
</s:Application>
```

Programáticamente mostrando que Vista es visible

```
<?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"
xmlns:views="views.*">
<s:layout>
<s:VerticalLayout paddingLeft="20" paddingTop="10" paddingRight="20" />
</s:layout>
<s:HGroup>
<s:Button label="TVs">
<s:click>
<![CDATA[
stack.selectedIndex = 0;
]]>
</s:click>
</s:Button>
<s:Button label="Computers">
<s:click>
<![CDATA[
stack.selectedChild = computers;
]]>
</s:click>
</s:Button>
</s:HGroup>
<mx:ViewStack id="stack" width="400" height="200">
<views:TVView label="Buy TVs" />
<views:ComputersView id="computers" label="Buy Computers" />
</mx:ViewStack>
</s:Application>
```

12.4.3 Manejando las interacciones del usuario con las Vistas en Pila

Viewstack es un componente hijo de la clase Container, por lo tanto hereda todas las propiedades de la misma. Pero como no se interactúa directamente solo un elemento es de interés: Change, el cual detecta los cambios en la selección.

El evento `change` genera un evento `IndexChangedEvent` con las propiedades mencionadas a continuación.

`newIndex` Number Especifica el número de indexación del nuevo `selectedIndex`
`oldIndex` Number Indica el número de indexación previo `selectedIndex`
`relatedObject` Object Hace referencia al contenedor al cual `newIndex` apunta

12.5 TabNavigator (Navegación en Pestañas)

La navegación en pestañas es muy utilizada y a sobrevivido al paradigma de la interfaz de usuario en lo que respecta a la navegación. Permite al usuario seleccionar lo que quiere ver en un espacio reducido de la pestaña.

12.5.1 Creando un TabNavigator

El proceso consiste en crear un tab para cada contenedor.

```
<?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"
xmlns:views="views.*">
<mx:TabNavigator width="200" height="100">
<s:NavigatorContent label="TVs" width="100%" height="100%">
<s:Label text="TV Content" />
</s:NavigatorContent>
<s:NavigatorContent label="Cameras" width="100%" height="100%">
<s:Label text="Camera Content" />
</s:NavigatorContent>
<s:NavigatorContent label="Computers" width="100%" height="100%">
<s:Label text="Computer Content" />
</s:NavigatorContent>
</mx:TabNavigator>
</s:Application>
```

12.5.2 Manjenedo la interacción del usuario con el tabNavigator

Como `tabNavigator` es un objeto hijo de `ViewStack`, se toman las propiedades de `vieStack`.

- `selectedIndex`—La posición de la pestaña seleccionada (comenzando en 0).
- `selectedChild`—El id de la pestaña seleccionada.

Respondiendo a los cambios en las pestañas en la navegación

```
<?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"
xmlns:views="views.*">
<s:layout>
<s:VerticalLayout paddingLeft="20" paddingTop="20" />
</s:layout>
<fx:Script>
<![CDATA[
import mx.controls.Alert;
import mx.events.IndexChangedEvent;
protected function onChange(event:IndexChangedEvent):void{
Alert.show("Changed to Tab Index " + event.newIndex +
"\nLabel: " + tabs.selectedChild.label);
}
]]>
</fx:Script>
<mx:TabNavigator id="tabs" width="200" height="100"
```

```

change="onChange(event)">
<s:NavigatorContent label="TVs" width="100%" height="100%">
<s:Label text="TV Content" />
</s:NavigatorContent>
<s:NavigatorContent label="Cameras" width="100%" height="100%">
<s:Label text="Camera Content" />
</s:NavigatorContent>
<s:NavigatorContent label="Computers" width="100%" height="100%">
<s:Label text="Computer Content" />
</s:NavigatorContent>
</mx:TabNavigator>
</s:Application>

```

12.6 Acordeón (Accordion)

12.6.1 Creando un acordeón

```

<?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"
xmlns:views="views.*">
<mx:Accordion width="200" height="100">
<s:NavigatorContent label="TVs" width="100%" height="100%">
<s:Label text="TV Content" />
</s:NavigatorContent>
<s:NavigatorContent label="Cameras" width="100%" height="100%">
<s:Label text="Camera Content" />
</s:NavigatorContent>
<s:NavigatorContent label="Computers" width="100%" height="100%">
<s:Label text="Computer Content" />
</s:NavigatorContent>
</mx:Accordion>
</s:Application>

```

Los acordeones siguen el mismo principio que los viewStacks y la navegación en pestañas al posicionar contenedores dentro de ellos.

12.6.2 Poblando un acordeón

Cualquier cosa que se pueda poner dentro de un contenedor se puede poner dentro de un acordeón, como miniaturas de fotos para un álbum de fotos o separar las secciones de un formulario largo.

12.6.3 Manejando la interacción del usuario con el acordeón

Aplica todo lo que se describe en la misma sección para la navegación en pestañas.

13 Pop-Ups

Flex ofrece un conveniente mecanismo en el administrador de pop-ups para ayudar a crear, borrar, posicionar, cerrar y destruir pop-ups. Conviene determinar el objetivo del pop-up previo a seleccionar su tipo.

13.1 Creando un pop-up

Todo pop-up es creado utilizando el pop-up manager. La clase PopUpManager maneja la inicialización de la ventana y la posición en las distintas capas de la aplicación.

13.1.1 Creando la ventana

Hay varias opciones para crear un pop-up, todas requieren al menos un archivo ventana para mostrar el pop-up. El administrador de pop ups llama a esta ventana y la renderiza por encima de la capa actual. El proceso es el mismo para otros pop ups. El proceso es el mismo para cerrar, se cierra primero el pop up de la capa más alta y se van cerrando las de las capas superiores primero.

A continuación se muestra el componente TitleWindow en uso, que hace esto posible.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/halo"
width="400" height="100" layout="vertical"
showCloseButton="true"
close="closeMe()" ">
<fx:Script>
<![CDATA[
import mx.managers.PopUpManager;
protected function closeMe():void{
//we will write this code later in the chapter
}
}]>
</fx:Script>
<s:Group>
<s:Label text="Hello there! I'm a simple popup window." />
</s:Group>
</mx:TitleWindow>
```

En el ejemplo anterior se usa el componente Label para mostrar un mensaje sencillo.

13.1.2 Usando el PopUpManager para abrir la ventana

En el ejemplo a continuación se muestra un pop up y como se abre el mismo ni bien se carga la aplicación.

```
<?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"
applicationComplete="openSimpleWindow()" ">
<fx:Script>
<![CDATA[
import mx.core.IFlexDisplayObject;
import mx.managers.PopUpManager;
import windows.SimplePopupWindow;
protected var simpleWindow:SimplePopupWindow;
protected function openSimpleWindow():void{
simpleWindow = new SimplePopupWindow();
PopUpManager.addPopUp(simpleWindow, this, false);
PopUpManager.centerPopUp(simpleWindow);
}
}]>
</fx:Script>
</s:Application>
```

Este es el mecanismo más común y mas usado para llamar un pop-up.

13.1.3 Cerrando el pop-up

En el l pop up se tiene la posibilidad de activar o desactivar la X superior izquierda para cerrar el mismo, esto se hace seteando a true la propiedad ShowCloseButton, y al utilizarse se llama al evento close y puede utilizarse para ejecutar eventos que estén

escuchando al mismo.

De la misma forma que se tiene la función `addPopUp`, la función `removePopUp` puede ser utilizada para referenciar cualquier objeto hijo.

13.2 Controlando la posición de la ventana

El pop up no se abre en el centro de la pantalla por defecto. Sino que se muestra en la esquina superior izquierda del objeto padre. Se debe indicarle manualmente la posición en la que se debe mostrar la ventana.

13.2.1 Usando el método `centerPopUp()`

El administrador de pop ups tiene el método `centerPopUp` que permite centrar los pop ups. El mismo toma un único argumento que es del tipo `IFlexDisplayObject` que es cualquier componente visual.

Con el ejemplo anterior como base, se usaría así:

```
protected function openSimpleWindow():void{
    simpleWindow = new SimplePopupWindow();
    PopUpManager.addPopUp(simpleWindow, this, false);
    PopUpManager.centerPopUp(simpleWindow);
}
```

La estrategia más fácil para trabajar con el posicionamiento de los pop ups es nombrar un padre tan cercano al root de la aplicación como sea posible y definirle a este padre la vista más grande posible dentro del espacio de la ventana.

13.2.2 Calculando la posición de la ventana

Aunque posicionar manualmente la ventana en la pantalla otorga mayor control el trabajo que requiere es mayor ya que se debe calcular la posición.

A continuación se muestra el pedazo de código que se usa para determinar la posición y el tamaño de la ventana.

```
var currentX:Number = x;
var currentY:Number = y;
var currentWidth:Number = ancho;
var currentHeight:Number = altura;
```

En el ejemplo a continuación se muestra cómo usar la posición del padre para calcular la posición del hijo y además se le pone 10px de padding.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/halo"
    width="300" layout="vertical" showCloseButton="true"
    close="closeMe()">
    <fx:Script>
    <![CDATA[
        import mx.managers.PopUpManager;
        protected function closeMe():void{
            PopUpManager.removePopUp(this);
        }
    ]]>
    </fx:Script>
    <!-- Content -->
</mx:TitleWindow>
```

```

    }
    protected function moveWindow(location:String):void{
    var newX:Number = 0;
    var newY:Number = 0;
    var padding:Number = 10;
    if(location == "center"){
    PopUpManager.centerPopUp(this);
    return;
    }
    if(location.indexOf("bottom") > -1)
    newY = (parent.height - this.height)-padding;
    if(location.indexOf("top") > -1)
    newY = padding;
    if(location.indexOf("left") > -1)
    newX = padding;
    if(location.indexOf("right") > -1)
    newX = (parent.width - this.width)-padding;
    move(newX,newY);
    }
    ]]>
</fx:Script>
<s:HGroup width="100%">
<s:Button label="Top left" width="100%"
click="moveWindow('topleft');"/>
<s:Button label="Top right" width="100%"
click="moveWindow('topright');"/>
</s:HGroup>
<s:VGroup width="100%" horizontalAlign="center">
<s:Button label="Center" click="moveWindow('center')"/>
</s:VGroup>
<s:HGroup width="100%">
<s:Button label="Bottom left" width="100%"
click="moveWindow('bottomleft');"/>
<s:Button label="Bottom right" width="100%"
click="moveWindow('bottomright');"/>
</s:HGroup>
</mx:TitleWindow>

```

Se vuelve a recordar que cuando se determina la posición por coordenadas todas las posiciones son relativas al tamaño y coordenadas del padre y no de la ventana principal.

