

MX DataGrids, Listas y Árboles

List-based components son una parte potente de tu arsenal de RIA, y algunos la consideran como el “workhouse” de una aplicación. Las listas manejan la mayor parte del trabajo que involucra despliegue de datos y provee varias de las conveniencias que mejoran las usabilidad de aplicaciones Flex.

8.1. MX List genealogy

Cuando se trata de List-based components de la librería MX, las clases más importantes son List Base y AdvancedListBase. Actúan como la base para los controles de MX List-based.

ListBase y AdvancedListBase proveen soporte para atributos como el ancho y la altura, cuantas filas son desplegadas, y el despliegue básico de la información. También puede manejar la interacción con el usuario a través eventos de activación.

Mirando la figura 8.1, se puede ver que lo que el componente List es capaz de hacer, también lo es el componente Tree, por herencia.

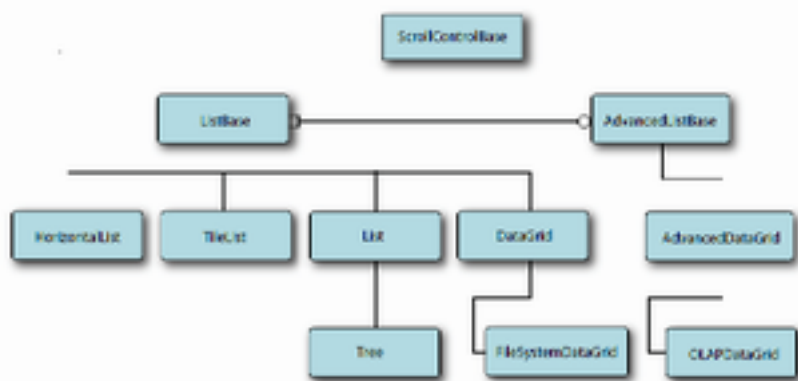


Figure 8.1 The List-based family of MX components and how they're related

8.1.1 Propiedades de ListBase y AdvancedListBase

Todos los componentes de listas y los árboles MX son derivados de ListBase y AvancesListBase, por eso cada uno incluirá las propiedades de esos objetos. Por ahora veremos las propiedades más comunes usadas, que son listadas en la tabla 8.1.

Flex es un entorno orientado a eventos, así que saber que eventos están disponibles para List-based MX components es muy importante.

Table 8.1 Common properties supported by MX List-based components

Property	Type	Description
<code>columnCount</code>	Number	Number of columns to be displayed.
<code>columnWidth</code>	Number	Width of the columns.
<code>dataProvider</code>	Object	Object containing the data to be displayed.
<code>iconField</code>	String	Fieldname, if any, that contains a reference to an icon. By default Lists look for a field called <code>icon</code> and use it if available.
<code>iconFunction</code>	Function	Optional function name you'd like to run for each icon. This can be used for formatting the icon.
<code>labelField</code>	String	Field name that contains a value to be displayed in the column.
<code>labelFunction</code>	Function	Optional function name you can call for each record. The function determines how the label is formatted and displayed.
<code>lockedColumnCount</code>	Number	Number of columns that always remain in view as the user scrolls horizontally. Used by <code>DataGrid</code> and <code>AdvancedDataGrid</code> .
<code>lockedRowCount</code>	Number	Similar to <code>lockedColumnCount</code> but applies to rows. Lets you specify how many rows at the top of a table are always in view as the user scrolls vertically.
<code>rowCount</code>	Number	Number of rows to display.
<code>rowHeight</code>	Number	Height of each row in pixels.
<code>selectable</code>	Boolean	Value that indicates whether you're allowing items to be selected by the user. Options are <code>true</code> (default) and <code>false</code> .
<code>selectedIndex</code>	Number	Row that has been selected (if any), if the component is selectable.
<code>selectedIndices</code>	Array	Array of all the selected indices; used when multiple selection is allowed.
<code>selectedItem</code>	Object	Link to the item that's currently selected. You can use this to access all the values pertaining to that row.
<code>selectedItems</code>	Array	Array of <code>selectedItems</code> for use when multiple selections are allowed.
<code>variableRowHeight</code>	Boolean	Value indicating whether each row can dynamically adjust its height to fit the content. Set this to <code>true</code> if you're allowing <code>wordWrap</code> . Options are <code>true</code> and <code>false</code> (default).
<code>wordWrap</code>	Boolean	Value indicating whether word wrapping is turned on. Options are <code>true</code> and <code>false</code> (default).

8.1.2 Eventos MX ListBase

Como la clase `ListBase` lleva consigo un conjunto mínimo de propiedades que son heredadas por sus hijos, los eventos de la clase `ListBase` son también heredados por las clases que extienden de esta (tabla 8.2). Puedes especificar el nombre de una función o cualquier código Action Script que quieras ejecutar cuando ese evento ocurre. Esta función es referenciada como el manejador del evento.

Table 8.2 Events that are common to List-based components

Property	Type	Description
<code>change</code>	Event	Triggers when the user selects a new row
<code>dataChange</code>	Event	Triggers when the data changes
<code>itemClick</code>	Event	Triggers when a user clicks a column
<code>itemDoubleClick</code>	Event	Triggers when a user double-clicks a column
<code>itemRollOut</code>	Event	Triggers when the user moves the mouse off an item
<code>itemRollover</code>	Event	Triggers when the user moves the mouse over an item

Miremos de donde vienen los datos y como se lo pasas a una List-based components.

8.2 Entendiendo las colecciones y los *DataProvider*.

La mayoría de los componentes que despliegan una serie de datos son alimentados por un objeto *DataProvider*.

Algunas veces puedes encontrar este tipo de componentes referidas como data-driven control.

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, miramos algunas a continuación.

8.2.1 Alimentando el *dataProvider*

Una variable puede ser usada como un objeto de bajo nivel como un Arreglo, un Booleano o una String, pero usualmente es mucho más beneficioso alimentar un *dataProvider* con un data type que extiende de *ListCollectionView*, como es un *ArrayCollection*.

La mejor parte acerca de objetos colección es que disparan un evento ***dataChange*** cada vez que una parte de los datos dentro de la colección cambia. Esto significa que cualquiera de los componentes que están vinculados a la colección detectan este evento de forma automática de esta manera pueden actualizar su propio estado visual cada vez que se produce un cambio en la colección.

Otra ventaja clave de una colección es que si cambia de estado, la colección difunde un evento que indica lo que ha cambiado. Todo lo que está esperando por ese evento responderá en consecuencia.

Los objetos de bajo nivel no soportan esta notificación de cambios automática, si son alterados, ninguno de lo que estén usándolo se dará cuenta de que algo ha cambiado.

8.2.2 Tipos de colección

Todos los diferentes tipos de colección soportan un conjunto común de capacidades básicas, la cual cada tipo extiende aún más con las funciones específicas de contexto:

ArrayCollection: la celebridad de las colecciones, *ArrayCollection* está basada en *Array*.

XMLListCollection: Una envoltura alrededor de XML y objetos *XMLList* que añade las características de la colección estándar.

GroupingCollection: Usada explícitamente por *AdvancedDataGrid* para agrupar datos.

8.2.3 Usuarios de colección

List-based components son los principales usuarios de las colecciones.

La tabla 8.3 muestra que componentes MX son alimentados por una colección de objetos siendo pasada a un *dataProvider*.

Table 8.3 Components that can use a collection for their dataProvider

AdvancedDataGrid	Menu
ButtonBar	MenuBar
Charting components including Legend	OLAPDataGrid
ColorPicker	PopUpMenuButton
ComboBox	Repeater
DataGrid	TabBar
DataField	TiledList
HorizontalList	ToggledButtonBar
LinkBar	Tree
List	

8.3 Initializing collections

Puedes usar dos métodos para inicializar una colección. El primero es el acercamiento MXML vía los tags asociados, como sigue:

```
<mx:ArrayCollection id="myAC">
<fx:Object label="Dan Orlando" data="dorlando" />
<fx:Object label="Tariq Ahmed" data="tahmed" />
<fx:Object label="John C Bland II" data="jcbland" />
</mx:ArrayCollection>
```

Alternativamente, puedes inicializar una colección en ActionScript puro importando la clase y después declarando una instancia de una variable, como se muestra en la siguiente lista.

```
<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" }
]);
]]>
</fx:Script>
```

8.4 Cargando colecciones

Vamos a ver varias técnicas a continuación de como cargar las colecciones.

8.4.1 Lista

Las listas son el component más liviano para desplegar información.

Invocando una lista.

Puedes crear y cargar una lista de diversas maneras. Empecemos con el listado 8.2 que muestra un método simple de hacerlo:

Usando un ComboBox para desplegar un 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>
</s:Application>
```

Contiene una sola lista con los nombres.

En este caso los datos están acoplados al código, es necesario separarlos. Al componente Lista se le puede decir dónde buscar la información pero no como se rellena. De esta forma si es necesario cambiar la forma de gestionar la colección, la lista no se ve afectada.

Usando solo MXML, vamos a actualizar el ejemplo para separar los datos en un Array-Collection y después usarlo para alimentar la lista dataProvider.

```
<?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>
<s:ArrayCollection id="myAC">
<fx:Object label="Ely Greenfield"/>
<fx:Object label="Ryan Stewart"/>
<fx:Object label="Chet Haase"/>
</s:ArrayCollection>
</fx:Declarations>
<mx>List id="myFriends" dataProvider="{ myAC }" />
</s:Application>
```

Porque en este ejemplo usamos un array de objetos cuando la lista original del ejemplo usa:

`<mx:String>?` Puede usar un acercamiento a eso así:

```
<fx:String>Dan Orlando</fx:String>
```

Pero un String solo puede guardar un solo valor, y en una aplicación real lo más probable es que necesites trabajar con múltiples campos

Otra opción es la que mostramos a continuación, que usa la clase Array-Collection de ActionScript para cargarlo con un grupo de objetos.

```
<?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:Script>
<![CDATA[
import mx.collections.ArrayCollection;
[Bindable]
public var myAC:ArrayCollection = new ArrayCollection([
{ label:"Dan Orlando" },
{ label:"Tariq Ahmed" },
{ label:"John C Bland II" }
]);
]]>
</fx:Script>
<mx>List id="myFriends" dataProvider="{ myAC }" />
</s:Application>
```

Eso es sencillo. Pero a menos que los datos siempre tengan una etiqueta llamada label, se necesita una manera de identificar la etiqueta más exclusiva.

Especificando una etiqueta.

En el mundo real, tus datos son probablemente representativos de las columnas de la base de datos donde estan almacenados. Con todas estas columnas que son devueltas, tiene que decirle a la lista que campos se va a usar para desplegar la información, especificando la labelField.

Listing 8.5 toma el ejemplo anterior dando un paso más incorporando este concepto e informando a la lista que columna desplegar vía el LabelField.

Usando el campo labelField para decirle a la lista que columna presentar.

```
<?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:Script>
<![CDATA[
import mx.collections.ArrayCollection;
[Bindable]
public var myAC:ArrayCollection = new ArrayCollection([
{name:"Dan Orlando", email:"dan@domain.com",
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/" }
]);
]]>
</fx:Script>
<mx:List id="myFriends" dataProvider="{ myAC }" labelField="name"/>
</s:Application>

```

Ahora sabemos cómo invocar una Lista, cargarla y especificar que columna va a ser usada para desplegar. Hasta este punto, la lista despliega en formato vertical ahora vamos a ver como desplegar en forma horizontal.

8.4.2 HorizontalList

La orientación por defecto de una lista es vertical (el contenido se muestra desde arriba hacia abajo). La lista vertical tiene una contraparte, conocida como HorizontalList; funciona exactamente igual que la lista por defecto, salvo que su orientación es horizontal. Rivasando el listado 8.5, cambiemos la lista por HorizontalList.

Listing 8.6 Switching from List to HorizontalList

```

<?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:Script>
<![CDATA[
import mx.collections.ArrayCollection;
[Bindable]
public var myAC:ArrayCollection = new ArrayCollection([
{name:"Dan Orlando", email:"dan@domain.com",
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"}
]);
]]>
</fx:Script>
<mx:HorizontalList id="myFriends"
dataProvider="{ myAC }"
labelField="name" />
</s:Application>

```

8.4.3 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. No hay un headers en la columna ni nada para ordenar. TileList es practica si se desea mostrar un catalogo visual.

Estas listas desplegar su información tanto horizontal (por defecto) como verticalmente. La principal diferencia entre las dos es como es usada la scrollbar.

Si quieres controlar el número de filas o columnas, tienes que mirar en las propiedades **columnCount** y **rowCount** respectivamente.

INVOCANDO UNA TILELIST.

El ejemplo con el que veníamos trabajando vamos a modificarlo para mostrar cómo esta presentada la información ahora.

```
<?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:Script>
<![CDATA[
import mx.collections.ArrayCollection;
[Bindable]
public var myAC:ArrayCollection = new ArrayCollection([
{name:"Dan Orlando", email:"dan@domain.com",
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>
<mx:TileList id="myFriends" dataProvider="{myAC}" labelField="name"/>
</s:Application>
```

←CREATE DATA COLLECTION AND MAKE BINDABLE

←SET DATAPROVIDER TO BINDABLE COLLECTION

Compila y corre la aplicación, se puede ver el array collection:



Por defecto la TileList despliega de izquierda a derecha. Para controlar este comportamiento agrega la propiedad de dirección y setéala vertical, el resultado será el siguiente:

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

Se puede setear la propiedad **columnWidth**, como se muestra a continuación de la imagen en este caso sería necesario ya que hay campos cortados.



```
<mx:TileList x="0" y="0" id="myFriends"
dataProvider="{myAC}"
labelField="name"
columnWidth="150" />
```

TILELIST VERSUS TILE

Como pauta general, TileList suele ser la forma más lógica de para usar. Usar una TileList consume menos memoria y produce una respuesta inicial más rápida, porque renderiza sólo la parte visible de los datos.

La ventaja es que desde una perspectiva de la memoria, la TileList está generando el mínimo número de objetos necesarios para mostrar en la pantalla.

El inconveniente puede ser la usabilidad si el contenido de la TileList es complejo y numeroso, lo que hace que la aplicación se sienta lenta.

En contraste con la forma en que el ítem renderizado es manejado por la List-based components , un Tile renderiza todos su hijos la vez, ya sea dentro de las dimensiones del escenario de la ventana gráfica actual o no.

Table 8.4 Additional properties of the DataGrid component

Property	Type	Description
<code>resizeableColumns</code>	Boolean	Determines whether the user is allowed to resize the column. Applies to all columns. Options are <code>true</code> (default) and <code>false</code> .
<code>sortableColumns</code>	Boolean	Determines whether the user is allowed to sort the columns. Applies to all columns. Options are <code>true</code> (default) and <code>false</code> .

8.4.4 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. La tabla 8.4 lista algunas propiedades adicionales del DataGrid que son específicas a su naturaleza multicolumn.

INVOCANDO UN DATAGRID

En referencia al árbol genealógico de los componentes basados en List, DataGrid es un hermano cercano a la lista. Como se ha demostrado en el listado 8.8, la invocación de un DataGrid es similar a la invocación de una lista.

8.8 Invoking a DataGrid, which is similar to invoking a List

```
<?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">
  <mx:DataGrid id="dg" width="500" height="200" >
    <mx:dataProvider>
      <fx:Object name="Tariq Ahmed" email="tariq@domain.com"/>
      <fx:Object name="John C Bland II" email="john@domain.com"/>
      <fx:Object name="Dan Orlando" email="dan@domain.com"/>
    </mx:dataProvider>
  </mx:DataGrid>
</s:Application>
```

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

En la salida que muestra la figura 8.5 vemos que a diferencia de ejemplos anteriores es que por ejemplo DataGrid incluye el encabezado y esta hecho de múltiples columnas. Además que cuenta

con varias características que hace que sea más amigable para el usuario como hacer click en el título de una columna y ordenarla, variar el tamaño de las columnas.

Al igual que con una lista, su dataProvider se rellena externamente. Tomando del ejemplo de ActionScript anterior, se enlaza el dataProvider a una variable que contiene la información (véase la siguiente lista).

8.9. Using an ArrayCollection to feed the DataGrid's dataProvider

```
<?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:Script>
<![CDATA[
import mx.collections.ArrayCollection;
[Bindable]
public var myAC:ArrayCollection = new ArrayCollection([
{name:"Dan Orlando", email:"dan@domain.com",
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"}
]);
]]>
</fx:Script>
<mx:DataGrid id="dg" width="500" height="150" dataProvider="{myAC}" >
<mx:columns>
<mx:DataGridColumn dataField="name"
headerText="Contact Name"
width="300" />
<mx:DataGridColumn dataField="email"
headerText="E-Mail"
width="200" />
<mx:DataGridColumn dataField="url"
headerText="Web Site"
width="200" />
</mx:columns>
</mx:DataGrid>
</s:Application>
```

CONTROLAR LA ORDENACIÓN.

sortableColumns es el principal interruptor que permite activar o desactivar la ordenación de columnas para todo el DataGrid. Para desactivar la ordenación, setea sortableColumns en false. Para activar o desactivar la ordenación por una columna base, establezca la propiedad **sorteable** en DataGridColumn en true o false según sea necesario.

ESPECIFICACIÓN DE LOS TÍTULOS COLUMNA CON DATAGRIDCOLUMN

El dataGrid es una elemento brillante, con poca información, puede deducir los nombres de campos como los títulos de las columnas.

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

Vamos a probar usar el DataGridColumn para etiquetar la cabecera de cada columna.

8.10 With DataGridColumn you can control attributes such as column titles.

```
<?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>
<s:ArrayCollection id="myAC">
<mx:Object name="Tariq Ahmed" email="tariq@domain.com"/>
<mx:Object name="Dan Orlando" email="dan@domain.com"/>
<mx:Object name="John C Bland II" email="john@domain.com"/>
</s:ArrayCollection>
</fx:Declarations>
<mx:DataGrid id="dg" width="500" height="150" dataProvider="{myAC}">
<mx:columns>
<mx:DataGridColumn dataField="name"
headerText="Contact Name" width="300"/>
<mx:DataGridColumn dataField="email"
headerText="E-Mail" width="200"/>
</mx:columns>
</mx:DataGrid>
</s:Application>
```

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

Table 8.5 Properties of the DataGridColumn

Property	Type	Description
<code>dataField</code>	String	Indicates which field in your dataset is represented by the selected column.
<code>headerText</code>	String	Indicates the column title.
<code>headerWordWrap</code>	Special	Permits word wrapping for a column title. Options are <code>true</code> (word wrap enabled) and <code>false</code> (disabled). If no value is entered, defaults to the <code>wordWrap</code> property setting in the DataGrid component.
<code>labelFunction</code>	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.
<code>minWidth</code>	Number	Specifies minimum width of a column.
<code>resizeable</code>	Boolean	Value that specifies whether the user is permitted to resize columns. Options are <code>true</code> (default) and <code>false</code> .
<code>sortable</code>	Boolean	Value that specifies whether the user is permitted to sort a column. Options are <code>true</code> (default) and <code>false</code> .
<code>sortCompareFunction</code>	Function	Adds the custom logic necessary to enable DataGrid to sort columns based on specified criteria such as numbers, dates, and so on. By default, the DataGrid uses a basic string comparison.
<code>sortDescending</code>	Boolean	Indicates the default sort order. Options are <code>true</code> (descending) and <code>false</code> (default, ascending).
<code>visible</code>	Boolean	Value that specifies whether the column is displayed. Options are <code>true</code> (default) and <code>false</code> .
<code>width</code>	Number	Specifies the width of the column. Can be given as a fixed number (in pixels) or a percentage of the window in which the column appears. If no columns have their width set, DataGrid distributes all columns evenly; if some are set and others aren't, the remaining unallocated space is distributed evenly among the unspecified columns.
<code>wordWrap</code>	Special	Similar to <code>headerWordWrap</code> , but specific to the column's content. If you provide any value, content wraps; otherwise, it defaults to the <code>wordWrap</code> property setting in DataGrid. If you enable <code>wordWrap</code> , be sure the DataGrid's <code>variableRowHeight</code> property is set to <code>true</code> .

8.4.5 Tree

El concepto de un árbol se remonta tan lejos como los sistemas operativos visuales y es algo sin duda en lo que has tenido experiencia con (Explorador de Windows y Buscador de Apple son buenos ejemplos). Un descendiente de la lista, un árbol se convierte en vital cuando se necesita mostrar datos jerárquicos, archivos y carpetas de un sistema operativo son los más obvios ejemplos.

Datos jerárquicos

Debido a la naturaleza anidada de la pantalla, tiene sentido que los datos necesitan estar estructurado en consecuencia. Una forma de datos estructurado como tal- XML. En Flex 3, los objetos XML y XMLList eran un tanto redundante, por lo que es importante saber que Flex 4 elimina el objeto XML en favor de XMLList y XMLListCollection.

Como una regla, un objeto XMLList sólo puede ser declarado en las etiquetas `<fx:Declarations/>`.

El fragmento que sigue presenta un objeto XMLList dentro de la etiqueta Declaraciones.

```
<fx:Declarations>
<fx:XMLList id="myXML">
<friends>
<friend name="Dan Orlando"/>
<friend name="John C Bland II"/>
```

```

<friend name="Tariq Ahmed"/>
</friends>
</mx:XML>
</fx:Declarations>

```

Un XMLList también puede introducir datos desde un archivo separado utilizando el atributo: source:<mx:XMLList source="my.xml" id="myXML"/>

XMLListCollection tiene la responsabilidad de envolver un objeto XMLList, como un ArrayCollection hace un Array, añadiendo la funcionalidad de la implementación de la ICollection. El siguiente fragmento de código muestra cómo hacer esto:

```

<mx:XMLListCollection id="myXMLCollection">
<mx:XMLList id="myXML">
<friends label="Friends">
<friend label="Elad Elrom"/>
</friends>
<families label="Family">
<family label="Laura Orlando"/>
</families>
</mx:XMLList>
</mx:XMLListCollection>

```

INVOKING A TREE

El ejemplo 8.11 crea una estructura de archivos de contactos.

8.11 Feeding the Tree XML data

```

<?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>

```

Se muestra en la figura 8.7 la salida. Están representados los nodos de nivel superior por carpetas, y los nodos finales muestran un icono de archivo.



8.5 Interacting with MX List-based components

Otro rol de las listas es dar al usuario un medio para interactuar con su aplicación a través de un componente que soporta una amplia variedad de interacción tipos.

8.5.1 List events

Table 8.6 Table events that can be used to handle interaction with List-based components

Event	Description
click	Occurs when the user clicks a component. This is a high-level event that's universal to many applications in Flex.
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.

8.5.2 Passing the event to a function

El listado 8.12 encarga al DataGrid a llamar a algunos ActionScript (en este caso, una función) que pasa a lo largo del objeto evento que se crea como parte del proceso.

Listing 8.12 Handling a user interaction by passing the event object to a function

```
<?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:Script>
<![CDATA[
import mx.collections.ArrayCollection;
import mx.events.ListEvent;
import mx.controls.Alert;
[Bindable]
public var myAC:ArrayCollection = new ArrayCollection([
{name:"Dan Orlando", email:"dan@domain.com"},
{name:"John C Bland II", email:"john@domain.com"}
]);
public function contactDataGrid_clickHandler( event:ListEvent ):void
{
Alert.show( "You clicked on row:" + evt.rowIndex + " and col:" +
evt.columnIndex + "." +
"Which is for " + event.currentTarget.selectedItem.name
);
}
}]>
</fx:Script>
<mx:DataGrid id="contactDataGrid" width="500" height="150"
dataProvider="{ myAC }"
itemClick="contactDataGrid_clickHandler( event )">
<mx:columns>
<mx:DataGridColumn dataField="name"
headerText="Contact Name"
width="300" />
<mx:DataGridColumn dataField="email" headerText="E-Mail" width="200"/>
</mx:columns>
</mx:DataGrid>
</s:Application>
```

Enfoquémonos en la siguiente línea. `itemClick="contactDataGrid_clickHandler(event)"`. Se indica la aplicación se ejecute la función de controlador de eventos especificada cuando

contactDataGrid distribuye un evento `itemClick`. El argumento de evento es pasado en el constructor porque la función del controlador tendrá que acceder a la información acerca de la interacción (por ejemplo, de dónde viene, los datos que se une al elemento de envío, y así sucesivamente).

Esta función, a su vez acepta el objeto de evento como parámetro y tiene acceso a varias propiedades pertenecientes a la misma. Cuando se utiliza este método, la propiedad principal en el que estás interesado es **`currentTarget.selectedItem`**.

La propiedad **`currentTarget`** es una referencia, a apuntador, a cualquier item que el evento esta relacionado con.

En este caso, significa `currentTarget` apunta a tu DataGrid.

Yendo un nivel más profundo, ahora que tiene una referencia al DataGrid, se accede a la propiedad **`selectedItem`**, que contiene una referencia a los datos de la fila con la que está asociado. Al continuar para profundizar, puede acceder a los campos de datos específicos sobre la línea:

```
evt.currentTarget.selectedItem.name
```

Eso sí, no está limitado a acceder a los datos mostrados, se puede acceder a cualquier campo que proporciona los datos. Usted puede hacerlo incluso si usted tiene muchos más campos:

```
{name:"Tariq Ahmed", email:"tariq@domain.com", domain:"www.dopejam.com"}
```

En este momento, está desplegando solo los campos `name` y `email`. Todavía tienes acceso al campo de dominio en `selectedItem`:

```
evt.currentTarget.selectedItem.domain
```

Similar a `selectedItem` es `selectedIndex`. Es un número que le informa sobre qué fila se hizo clic (0 es la primera fila).

8.5.3 Pasando información a una función

```
<mx:DataGrid id="dg" width="500" height="150" dataProvider="{ myAC }"
itemClick="handleClick( event.currentTarget.selectedItem )">
<mx:columns>
<mx:DataGridColumn dataField="name"
headerText="Contact Name" width="300" />
<mx:DataGridColumn dataField="email"
headerText="E-Mail" width="200" />
</mx:columns>
</mx:DataGrid>
```

Al hacer clic en un elemento, su función se invoca junto con una referencia al elemento clickeado. Esto te permite acceder a las distintas propiedades del objeto:

```
public function handleClick (data: Object): void
{
Alert.show ("Nombre:" + data.name "Email:" + data.email);
}
```

El inconveniente de este enfoque es que la función no tiene ninguna referencia al componente que provocó el evento. Sin esta información, la función no se puede llevar a cabo tareas como el cambio de color del componente en rojo si se produce un problema.

8.5.4 Accessing the selected row directly

Un enfoque más sencillo es acceder al componente directamente. La ventaja de este enfoque (aparte de que es más fácil de implementar) es que usted puede fácilmente acceder a cualquiera de las propiedades específicas de ese tipo de componente (en este caso, una DataGrid):

```
<mx:DataGrid id="dg" dataProvider="{myAC}" itemClick="handleClick()"> (...)
```

Tenga en cuenta que no estas pasando nada a través del manejador itemClick, La función entonces accede directamente al valor:

```
public function handleClick():void
{
    Alert.show("Name:" + contactDataGrid.selectedItem.name + ", Email:" +
contactDataGrid.selectedItem.email);
}
```

Es una técnica simple, pero hace el trabajo, y te beneficias con la facilidad de mantenimiento con respecto a la función, que no le importa quién lo llamó o por qué se llamaba.

8.5.5 Binding to a selected row

Este es el enfoque más sencillo.

Mediante el uso de función de enlace de Flex, se puede mostrar o almacenar el elemento seleccionado sin necesidad de utilizar controladores de eventos. Este fragmento de código muestra cómo utilizar esta técnica sin una función, mediante la unión de la selección actual a un elemento que pide el valor:

```
<mx:DataGrid id="contactDataGrid" width="500" height="150"
dataProvider="{myAC}">
<mx:columns>
<mx:DataGridColumn dataField="name" headerText="Name" width="300" />
<mx:DataGridColumn dataField="email" headerText="E-Mail" width="200"/>
</mx:columns>
</mx:DataGrid>
<mx:Label text = "The selected person is { + dg.selectedItem.name +
dg.selectedItem.email + }." fontSize="16" />
```

8.6 Resumen

List-based components, sin duda, una parte fundamental de muchas aplicaciones Flex y el caballo de batalla visual de mostrar los datos con Flex. Parte de esto se debe a que las listas sean un paradigma común de UI. La mayoría de las aplicaciones Flex que construiremos interactúan con un nivel intermedio y de nivel de base de datos, los cuales se ejecutan en un servidor. Usted finalmente a realizará peticiones de datos a los niveles de servidor.

Una vez que se muestran los datos, la última pieza del rompecabezas es el manejo de las interacciones del usuario. Usted tiene muchas opciones para lograr esto, se puede pasar objetos de evento entero a una función de controlador de eventos, se puede pasar sólo los datos, o los datos pueden ser recuperados por la función directamente. Para los casos simples, usted puede tomar ventaja del enlace de datos.