

A dark blue vertical bar on the left side of the slide. A blue arrow points to the right from the bar, containing the date.

23-5-2013

Flex 4 in Action

RIA

Several thin, curved lines in dark blue and light grey originate from the bottom left corner and sweep upwards and to the right.

Grupo 1

Juan Pablo Rozada, Camilo Pereyra, Juan Pereyra,
Hernan Pereyra, Yennifer Herrera

2.8 Resumen

Razones por las cuales usar Flex:

1. El SDK es libre
2. Usa el IDE de Eclipse
3. La interfaz de Flash Builder puede ser totalmente personalizada.
4. Es un lenguaje orientado a objetos
5. ActionScript es MXML como JavaScript es HTML, MXML se usa para diseñar sus componentes y ActionScript para la lógica de negocio, ActionScript es similar a JavaScript derivan de la misma norma de programación ECMAScript Son tecnologías del lado del cliente y están basadas en un modelo EventDriven, la diferencia es que ActionScript crea un objeto de evento con un controlador de eventos con el cual se puede acceder a casi todo sin necesidad de acceder el origen del evento

4. Diseño y contenedores

En este capítulo se habla de los bloques de construcción visual y los tipos de contenedores que puede utilizar para crear rápidamente diseños en una aplicación Flex.

Los contenedores son la base de todos los componentes, hace más fácil agrupar las colecciones de componentes dentro de estos, los contenedores están limitados a un conjunto específico de componentes, estos componentes pueden ser cualquier cosas desde botones, tablas e incluso otros contenedores.

4.1 Spark versus Halo (MX)

En este capítulo se comparan Los componentes de Halo también conocidos componentes MX, y los componentes de flex4 que es una nueva generación de componentes denominada Spark que va más allá de los componentes de flex3 como lo son los botones, campos de texto, contenedores, etc.

Los componentes de Spark tienen un conjunto de reglas diferentes a los de Halo/MX cuando se trata de posicionamiento, dimensionamiento y diseño.

Básicamente los componentes de spark permiten cambiar el algoritmo de diseño mientras que los de Halo/MX tienen su algoritmo de diseño dentro por lo tanto para cambiarlo se tendría que cambiar el tipo de contenedor.

El Layout se maneja a través del controlador de distribución, que utiliza tres etapas para averiguar la posición y el tamaño de cada componente visual:

- **Commitment pass (Compromiso de paso):** mira la configuración de las propiedades de todos los componentes (por propiedades ejemplo, altura y anchura). Durante esta etapa, cada componente tiene su método **commitProperties ()** ejecutada, la cual proporciona el controlador del layout las propiedades position/size-related. Este paso trata de averiguar lo que cada componente está solicitando su posición y tamaño.
- **Measurement pass (Medición de paso):** evalúa el tamaño por defecto de todos los componentes. Empieza por mirar en el componente más profundo (desde una

perspectiva de anidación) y se abre camino. En este paso se trata de calcular el tamaño predeterminado o explícito de cada componente. Algunas de las variables que intervienen en este cálculo incluyen sumando el ancho de todos los objetos secundarios interiores, grosor de los bordes, el relleno y los espacios entre los hijos. El controlador del layout pide a cada objeto para ejecutar su **measure-Size()** método para determinar el valor por defecto / tamaño explícito.

- **Layout pass:** Ahora que el controlador del Layout ha reunido toda la información de los pasos anteriores, se presenta todos los componentes mediante el establecimiento de la posición y el tamaño de cada componente. A diferencia del paso de medición, se inicia hacia afuera y se abre camino en los componentes anidados. Este paso requiere de cada componente el método **updateDisplayList ()** para que el componente pueda actualizar su pantalla.

Cada contenedor Spark apoya el cambio de su enfoque de diseño para uno de los siguientes:

- **BasicLayout:** También conocido como el esquema absoluto, en el que utiliza coordenadas x e y.
- **HorizontalLayout:** Componentes que se colocan uno tras otro horizontalmente en una sola fila.
- **VerticalLayout:** Los componentes se colocan uno tras otro verticalmente en una sola columna.
- **TileLayout:** muestra los componentes en forma de rejilla, creando tantas filas y columnas como sea necesario.

¿Suenan un poco desalentador? La buena noticia es que todos los componentes, tanto Spark y Halo / MX, tienen un comportamiento por defecto que permite a Flex hacer un buen trabajo de posicionamiento de forma automática. Así que vamos a saltar en él, comenzando con el layout absoluto.

4.2 Absolute layout.

Es un modo de composición en la que se controla la posición exacta y el tamaño de los elementos en su lugar dentro del contenedor.

Contenedores implementan el posicionamiento basado en una cuadrícula de coordenadas de dos dimensiones. La esquina superior izquierda de cualquier contenedor representa las coordenadas de origen 0,0. Cualquier artículo en el contenedor se coloca en respecto a ese punto.

TIP: En el ámbito Flex, la coordenada x se inicia a la izquierda y la y en la parte superior. A medida que aumenta el valor positivo de estas propiedades, se mueve hacia abajo y hacia la derecha.

Uno de los beneficios del uso de Absolute Layout en un contenedor es que brinda el control finegrained sobre la localización de los objetos y su tamaño. Otro de los beneficios de flex (técnicamente Flash Player) no es necesario para quemar el poder de procesamiento de cálculo.

Lo bueno es que se puede mezclar y combinar por tener un contenedor de absolut Layout, mientras que otros utilizan automática.

En el código Especificamos la clase Basic-Layout porque queríamos posicionamiento absoluto, y porque es el contenedor por defecto de la aplicación.

En flex hay que dejar librar la imaginación y pensar en un mundo más amplio ej en html posicionar dos objetos superpuesto normalmente es visto como un bug pero en flex se puede ver de forma benéfica dependiendo de cómo se utilice, otro aspecto interesante es que las propiedades x e y pueden tomar valores negativos y con eso poder posicionar objetos fuera de la visual.

TIP: El atributo Id permite nombrar exclusivamente una instancia de un componente igual que con JavaScript y HTML

Absolute Layout asume muchas cosas respecto al posicionamiento, lo cual puede o no satisfacer nuestras necesidades. Para resolver algunas de estas cosas podemos usar un método de restricción que determine el posicionamiento relativo a ciertos puntos de anclaje.

4.3 Constraint-based layout (Diseño basado en restricciones)

Si el suministro de coordenadas exactas es práctico para su aplicación, se puede utilizar una variación, en Absolute Layout conocida como Constraint-based layout (diseño basado en restricciones). En lugar de especificar coordenadas X e Y a los componentes de posición con respecto a la esquina superior izquierda del contenedor, constraint-based layout tiene elementos de posición relativa a los bordes del contenedor o su centro.

La ventaja de este método es que se puede configurar los componentes para mantener un posición relativa dentro del contenedor, incluso si el usuario cambia el tamaño de la ventana.

Para lograr los mismos resultados usando coordenadas fijas requeriría una considerable cantidad de código de ActionScript para realizar los cálculos y actualizar el componente de valores x e y en respuesta a los cambios de tamaño de la ventana.

4.3.1 Basic constraints

- **El top, bottom, left, y right:** son propiedades que permiten controlar la distancia desde el borde.
- **horizontalCenter y verticalCenter:** controlan la distancia desde el centro.
- **baseline property:** permite fijar la distancia entre el borde superior de la componente y su contenedor principal.

Flash Builder incluye una característica que le permite configurar fácilmente las limitaciones de componentes.

TIP: Dado que Flash tiene sus ancestros en una herramienta de animación, e termino Stage es a menudo utilizado en lugar de la página. Los desarrolladores de Flex vienen en dos variedades: Tradicional programadores y diseñadores de Flash. Se dará cuenta que en foros y publicaciones las personas que provienen del mundo flash tienden a utilizar el escenario Stage. Cuando usted oye la frase "centered on stage", significa que un ítem se centra en un Contenedor o página.

Cuando se quiere fijar la posición exacta de un componente dentro de un contenedor, no importa las dimensiones del navegador, use Absolute Layout. Cuando quiera que un componente sea capaz de moverse con respecto a las dimensiones cambiantes de una ventana, pero permanecer dentro de una determinada distancia del borde de un navegador, use Constraint-based layout para manejar esto automáticamente.

4.3.2 Enhanced constraints (restricciones mejoradas)

Algunas restricciones por ejemplo de filas y columnas son:

1. **Fixed:** determina la posición por un número fijo en pixel.
2. **Relative:** La posición está determinada por un porcentaje, en relación con el tamaño del recipiente.
3. **Content-sized:** similar a la restricción relativa, con la excepción de que la posición se basa en el contenido (la ampliación arriba y hacia abajo para acomodar el contenido).

TIP: Debido a que usted está asumiendo el control del Layout con Constraint, son casi exclusivamente utilizados con el Contenedor Canvas de Halo/MX-based. No como con contenedores Spark. Apoyar las Enhanced constraints, aunque los componentes Spark trabajarán como era de esperar con columnas y filas de restricción cuando se coloca dentro de un recipiente MX que utiliza tales restricciones.

La mejor manera de trabajar con Enhanced constraints es verlo como un mecanismo para dividir contenedores en secciones.

Si su aplicación lo requiere, puede combinar filas y columnas de restricción en restricciones (constraints).

RESTRICCIONES DE USO CONTENIDO DE TAMAÑO (CONTENT-SIZED CONSTRAINTS)

Content-Sized Constraints se activan automáticamente al no suministrar los valores de una restricción de las propiedades height y width. Flex escala todos los items al ancho o alto de la columna o fila, que a su vez se basan en el elemento más grande de su contenedor.

4.4 Automatic layout

Es una desviación considerable de Absolute Layout, en la que se determina explícitamente la localización y disposición de los elementos dentro de un contenedor.

4.4.1 Usando Layout Class

En la sección 4.1 dijimos que hay cuatro clases de diseño disponibles en los contenedores Spark, una de las cuales se utiliza para el Absolute Layout (la clase BasicLayout), y las restantes son HorizontalLayout, VerticalLayout y TileLayout las cuales nos proporcionan Automatic Layout.

La clase TileLayout organiza los elementos de una forma de grilla. Aunque cada celda es igual en tamaño, la clase TileLayout proporciona una serie de propiedades para el control de la dimensión de las celdas, la distancia entre ellas, contar, orientación, etc.

De forma predeterminada, la clase TileLayout exhibirá cada elemento de izquierda a derecha, fila por fila.

¿Y qué si usted quiere tener que ir de arriba a abajo y luego ir a la columna por columna?

Aquí es donde la API Reference puede ayudar a averiguar sus opciones.

Si tuvieras `TileLayout` en el código como tal

```
<s:layout>
```

```
    <s:TileLayout/>
```

```
</ s: Disposición>
```

Usted puede hacer clic en Mayús + F2 para levantar API Reference en ese componente. En la sección Propiedades, verías algo llamado Orientación, que controla la disposición de los elementos. Al hacer clic en esa propiedad, entonces usted dice que puede pasar en "filas" o "columnas" como valores.

4.4.2 Getting spaced out

El componente `<mx:Spacer/>` es otra herramienta su layout. El prefijo `mx` en la etiqueta nos indica que es un componente de Halo/MX, pero se puede mezclar con los dos Spark y componentes y contenedores de Halo/MX.

Usted puede comparar espaciador en una columna de vacío de HTML (`<td width="100%"> </td>`) o el truco `spacer.gif` HTML, que fue utilizado durante algún tiempo para distribuir elementos uniformemente. Usando API Language Reference, se ve que el `Spacer` soporta `width`, `height`, mínimo y máximo de versiones de esas propiedades. Estas propiedades puede contener un número (de píxeles) o un porcentaje.

Si usted usa la Automatic layout o Absolut Layout para sus proyectos, los mismos componentes son capaces de controlar su propio tamaño.

4.5 Variable and fixed sizing

Al igual que HTML, pero mucho más allá de lo que soporta HTML, todos los componentes visuales de Flex soportan tamaño variable y fijo. Esto es posible gracias a las propiedades de anchura y altura, que aceptan no sólo valores fijos para definir el tamaño del elemento en números absolutos (en píxeles), sino también porcentajes eso hace escalable el ítem en relación al tamaño del contenedor que lo hospeda.

4.5.1 Variable sizing

Si se quiere un botón con una escala parcial de un 80% de la anchura de su contenedor principal (Application Container), puede utilizar el siguiente código:

```
<s:Button label="My Button" width="80%"/>
```

Esto se traduce en un botón que se extiende en un 80% de la ventana del navegador.

4.5.2 Fixed sizing

Es igual que en el caso anterior con la diferencia que en vez de usar porcentajes se usan tamaños fijos en pixeles entonces para darle a un botón un ancho de 120 pixeles y un alto de 80 pixeles se hace de esta manera:

```
<s:Button label="Hello!" width="120" height="80"/>
```

No hay mucho más para decir acerca del dimensionado físico, es extremadamente simple. Ahora que puedes controlar el tamaño y posición de los componentes y estas familiarizado con los principios detrás del esquema de distribución automático, exploraremos los contenedores que utilizan estas características.

4.6 Containers

Los contenedores son colecciones de componentes que tienen como misión proveer una estructura visual a una aplicación, y la idea es que nos ayude a esquematizar el diseño.

Lo que ha cambiado significativamente desde los contenedores del Flex 3 Halo/MX a los del Spark en Flex 4, es que los Halo/Mx en sí proveían distintos tipos de esquemas. Por ejemplo Halo/MX tiene un contenedor llamado `<mx:HBox>` el cual organiza los componentes horizontalmente.

Pero con Spark la responsabilidad de la esquematización fue movida a clases que se encargan de los esquemas, esto permite utilizar cualquiera de los esquemas de diseño. Entonces con los estos contenedores es menos acerca de esquemas y más acerca de agrupar elementos.

A grandes rasgos se puede tomar una analogía en la que los contenedores Spark son más parecidos a los tag `<div>` de HTML y los esquemas de clases son más parecidos a los estilos CSS utilizados para controlar el posicionamiento de los tags div.

En Flex 4, se incorporan los siguientes contenedores Spark:

- Application: Único por aplicación, es el contenedor root de la aplicación.
- Group: contenedor básico para agrupar elementos, por defecto utiliza la clase `BasicLayout` para el posicionado absoluto. Sus contenedores hijos `HGroup` y `VGroup` proveen un posicionamiento de capas horizontal y vertical respectivamente.
- SkinnableContainer: es similar al de grupos, pero tiene la capacidad de incorporar pieles.
- Panel: Basado en el contenedor de pieles, agrega una barra de título y un frame.
- DataGroup: Está pensado para agrupar datos (ej. Arrays) que pueden ser renderizados, lo que permite una mejor visualización.
- SkinnableDataContainer: Similar al `DataGroup`, pero una versión compatible con pieles. El intercambio entre el modo con y sin piel y viceversa, puede llegar a complicar, pues la tarea principal es la de renombrar los componentes en uso. Los contenedores con la capacidad de pieles son más pesados y cargan con el contenido extra para el soporte de los mismos. Por lo general si no está planeado el uso de pieles se recomienda utilizar el modo sin pieles para evitar contratiempos.

4.6.1 Application container

En Flex las aplicaciones requieren un contenedor de inicio o maestro (`<s:Application/>`), este reemplazaría al contenedor `<mx:Application>` de Halo/MX. Otra característica del contenedor root, es la propiedad `preloader`, que es la barra de progreso que se ve al lanzar una aplicación Flex. Al ser el objeto de más nivel en la aplicación se lo puede utilizar para contener variables globales y funciones, permitiéndonos el acceso a ellas desde cualquier parte de nuestra aplicación.

Podremos decir de momento que el objetivo de los componentes personalizables es el de modularizar nuestro código, el siguiente código nos muestra un ejemplo:

Listing 4.11 Invoking a custom component

```
<?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:local="*"
               >
  <fx:Script>
    <![CDATA[
      [Bindable]
      public var myString:String = "hello";
    ]]>
  </fx:Script>
  <local:CustomComponent/>
</s:Application>
```



Podremos notar que el tag 1 define algo llamado local, con un comodín como path, esto le indica a Flex que cualquier componente precedido por este tag puede encontrarse en el directorio root de la aplicación.

Con nuestro componente local en su ubicación, y el contenedor de la aplicación siendo capaz de encontrarlo, nuestro componente personalizado podrá acceder a las variables de la aplicación, tal como se puede ver en el siguiente ejemplo:

Listing 4.12 CustomComponent.mxml—accessing the application's variables

```
<?xml version="1.0" encoding="utf-8"?>
<s:Group 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.core.FlexGlobals;
    ]]>
  </fx:Script>
  <s:Button label="{FlexGlobals.topLevelApplication.myString}"/>
</s:Group>
```

Luego de correr la aplicación debería de tener un botón con el texto “hello.”, se pueden tener varios tipos de contenedores en una misma aplicación que abarque todos nuestros archivos, pero se puede tener únicamente un contenedor `<s:Application/>`.

4.6.2 Canvas container

Estos contenedores son el tipo más básico que se puede tener, son livianos y no muy robustos, están basados en los componentes Halo/MX. Por esto último es que optaremos por los contenedores basados en Spark (por ejemplo agruparlos usando la clase `BasicLayout`).

La única razón por la cual podríamos querer usar el contenedor canvas, sería en el caso que necesitemos utilizar restricciones mejoradas (visto en la sección 4.3.2)

4.6.3 Group-based containers and SkinnableContainer

El contenedor de grupo es un contenedor simple que se empareja con una de las clases de diseño, fueron utilizadas en la sección 4.4.1 con el contenedor de aplicación, pero se pueden utilizar igual de un modo simple con el contenedor de grupo, tal como muestra el cuadro 4.13.

Demos un paso a un nivel superior mediante el uso de dos contenedores de grupo, el primero deja sus elementos de forma horizontal, mientras que su contenedor aplicación padre utiliza un diseño vertical.

Listing 4.13 Using multiple containers for more complex layouts

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

  <s:layout>
    <s:VerticalLayout/>
  </s:layout>
  <s:Group>
    <s:layout>
      <s:HorizontalLayout/>
    </s:layout>
    <s:Button label="Button 1"/>
    <s:Button label="Button 2"/>
  </s:Group>
  <s:Group>
    <s:layout>
      <s:HorizontalLayout/>
    </s:layout>
    <s:Button label="Button 3"/>
    <s:Button label="Button 4"/>
  </s:Group>
</s:Application>
```

Se pueden anidar así como mezclar y combinar los contenedores y los diseños del modo que deseemos para crear nuestro diseño deseado. En este caso, obtendremos 2 filas de botones, tal como se puede ver en la siguiente imagen:



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

De un modo alternativo, podemos en vez de especificar la clase de diseño junto con el contenedor de grupo `<s:Group/>`, usar el contenedor hijo del grupo `<s:HGroup/>` para el diseño horizontal, o el `<s:VGroup/>` para el diseño vertical. Proveen la ventaja de no tener que generar tanto código.

El contenedor `SkinnableContainer` funciona del mismo modo, podemos reemplazar el contenedor de grupo del ejemplo 4.13 con este otro y tendremos exactamente el mismo resultado.

La diferencia está en que el contenedor `SkinnableContainer` nos permite agregar una colección de diseños en el formato de una sola piel.

Existe un capítulo dedicado exclusivamente a pieles y temas, pero para tener una idea de cómo funciona, podemos decir que se crea un archivo de piel por separado, ahí le decimos al contenedor de pieles que lo use, hay dos reglas básicas para crear un archivo de piel:

- 1- Podemos dar soporte a todos los componentes que estarían incluidos, en este caso el componente puede estar en modo normal o deshabilitado. Podremos saberlo mirando

en la API de referencia del componente al que se desea dar soporte (el contenedor de pieles), y de este modo podremos ver los estados de los distintos componentes.

- 2- Se debe definir un contenedor adicional con el contenido correspondiente a las pieles, se debe colocar todo el contenido relacionado dentro de ese contenedor (aún así estamos creando una piel para otro contenedor)

A continuación un ejemplo de una clase piel llamada CoolSkin.mxml

Listing 4.14 CoolSkin.mxml—a file that defines a skin class

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

  <s:states>
    <s:State name="normal" />
    <s:State name="disabled" />
  </s:states>

  <s:Rect height="100%" width="100%">
    <s:fill>
      <s:LinearGradient>
        <s:entries>
          <mx:GradientEntry color="#92A1B9"/>
        </s:entries>
      </s:LinearGradient>
    </s:fill>
  </s:Rect>

  <s:Group id="contentGroup" left="5" top="5" right="5" bottom="5">
    <s:layout>
      <s:BasicLayout/>
    </s:layout>
  </s:Group>
</s:SparkSkin>
```

Rectangle used in background

Content goes in here

Use constraints to add padding

Estamos haciendo uso de nuestra piel, más abajo podemos ver como el contenedor SkinnableContainer especifica el uso de la propiedad skinClass, la cual se encarga de cargar la piel definida anteriormente

Listing 4.15 A SkinnableContainer making use of a skin

```
<s:SkinnableContainer skinClass="CoolSkin">
  <s:layout>
    <s:HorizontalLayout/>
  </s:layout>
  <s:Button label="Button 1"/>
  <s:Button label="Button 2"/>
</s:SkinnableContainer>
```

Skin specified

Como se puede ver es un trabajo sencillo, es interesante destacar que las pieles pueden ser usadas en varios componentes. Podemos ver que la propiedad id es instanciada a contentGroup, esto es debido a que Flex buscará por un contenedor llamado contentGroup para saber a dónde destinar los componentes visuales asociados a la piel.

En el siguiente ejemplo podremos ver como el contenedor ahora presenta una piel con forma rectangular para contener a los botones.



Figure 4.18
SkinnableContainer with a skin
class that defines a square background

4.6.4 Panel container

El favorito de las masas, el contenedor Panel, es usualmente usado como un contenedor top-level para la aplicación en su totalidad (se pueden contener Paneles dentro de otros paneles también). Lo bueno de este envase es que añade una barra de título y de estado en la parte superior de la ventana y por defecto dibuja un borde alrededor de sus objetos secundarios.

El contenedor Panel usa por defecto la clase BasicLayout, y como es hijo del contenedor SkinnableContainer hereda todas sus propiedades y habilidades. En el siguiente cuadro agregaremos un HGroup al contenedor dentro del Panel de modo de agregar restricciones para añadir algo de relleno desde el borde del panel, de lo contrario los botones van hasta el borde y no quedan tan agradables estéticamente.

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

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

← Constraints
for padding

A continuación un ejemplo de cómo se vería:



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

4.6.5 ApplicationControlBar container

Tal como su nombre lo indica, el contenedor ApplicationControlBar agrega un área de control a nuestra aplicación. El contenedor ApplicationControlBar crea un área en la parte superior de la aplicación que es similar al menú de archivo usado en la mayoría de aplicaciones de escritorio. Para usar este contenedor lo combinamos con el contenedor de la aplicación tal como podemos ver en el siguiente ejemplo:

Listing 4.17 ApplicationControlBar provides easy access to common functionality.

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

Podemos de esta forma brindar a los usuarios un acceso instantáneo a las características frecuentes de la aplicación. Esta característica es basada en los componentes Halo/MX, y no tiene un equivalente en Spark.

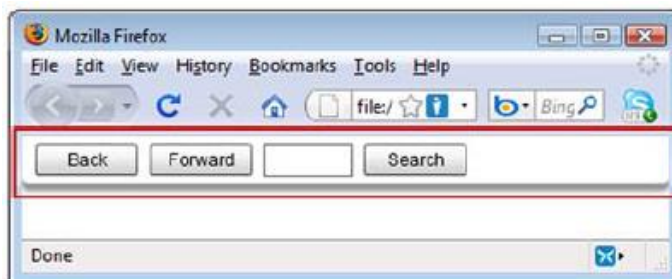


Figure 4.20
The ApplicationControlBar container enables quick access to frequently used features.

Hay un contenedor similar en Halo/MX llamado ControlBar, el cual combinado con el contenedor Panel agrega un borde inferior al panel para poner botones del estilo aceptar, cancelar. Sin embargo no es compatible con el contenedor Panel del Spark.

4.6.6 DataGroup and SkinnableDataContainer

A simple vista estos contenedores son similares a sus contrapartes Group y SkinnableContainer, pero cuando analizamos más a fondo vemos que agregan una característica que es la de capturar datos y enlazarlos con un renderer para visualizar la información.

Un renderer es uno de elementos incluidos en Flex (como ser botones, campos de texto, o los componentes personalizados que podamos crear), el propósito de los mimos es el de capturar datos en tiempo de ejecución.

Para poder utilizar uno de estos contenedores necesitamos datos, los cuales luego serán enviados al contenedor DataGroup o SkinnableDataContainer. Esto sería un array ser en forma de colección (por ejemplo ArrayCollection) y puede contener cualquier cosa, como ser strings, botones o gráficos.

En Flex disponemos de dos renderers de elementos que para la visualización de información:

- 1- spark.skins.default.DefaultItemRenderer— Muestra datos como texto simple.
- 2- spark.skins.default.DefaultComplexItemRenderer— Muestra datos como componentes dentro de un contenedor de grupo. Esto es útil únicamente si los datos traen restricciones de componentes visuales (botones, imágenes, etc.)

Se verá mejor en un ejemplo en el cual creamos una variable ArrayCollection con un array de strings en el, como se puede ver a continuación:

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

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

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

Por defecto el DataGroup usa la clase BasicLayout, la cual si la dejamos así sobrepondrá cualquier ítem de texto sobre otros, en su lugar se usa la clase HorizontalLayout para que quede mejor presentado.


De modo similar se puede modificar el código para crear un array de botones y usar el componente DefaultComplexItemRenderer para mostrarlo tal como es, ejemplo:

Listing 4.19 Using an array of components to display

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

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

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

 **1** Declare all non-visual tags first

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

Al estar utilizando la clase TileLayout, se despliegan los botones de modo de seguir un patrón de grilla como se ve a continuación:



Figure 4.21 A tiled layout of buttons using the DataGroup container

Utilizando el contenedor SkinnableDataContainer, tendremos el mismo proceso que el que se usa para ir desde Group al contenedor Skinnable-Container, donde reemplazamos los componentes en el código y aplicamos la piel deseada al control en la vista.

4.6.7 DividedBox, HDividedBox, and VDividedBox containers

A pesar de ser contenedores Halo/MX, vale la pena mencionarlos, ya que nos proveen de características extra al dar la capacidad de redimensionar al usuario. Esto es similar a los frames de HTML, donde el redimensionado está habilitado. El contenedor DividedBox soporta ambos diseños horizontal y vertical (usado por defecto) usando el atributo de dirección. Es posible de un modo más directo utilizar tanto HDividedBox (diseño horizontal) o VDividedBox (vertical).

Usando el puntero del mouse, podremos encontrar el punto para redimensionar el panel. Esta opción permite a los usuarios personalizar el área de trabajo. Adicionalmente, se pueden anidar para crear diseños muy útiles e interesantes.

Listing 4.20 Making a dashboard with DividedBoxes

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

Este tipo de contenedor es especialmente útil al momento de crear recuadros. Si se tiene mucha información para presentar en los recuadros de texto (como se ve en la siguiente imagen), este mecanismo es el más conveniente, pues permite al usuario ajustar como ver cada sección según la información que más le importe

Figure 4.22 Divided boxes allow the user to customize the dimensions of the application.

4.6.8 Form container

El propósito de este contenedor es facilitar la creación de formularios en la aplicación. Usar el contenedor es similar a crear formularios en HTML, pero en Flex el contenedor tiene los mecanismos para situar las etiquetas junto con los cuadros de ingreso de texto. A diferencia de HTML, quien requiere que los elementos del formulario estén contenidos dentro del tag HTML, el contenedor de Flex es únicamente para diseño y no es necesario tener ítems del formulario dentro del contenedor.

Un formulario HTML indica al navegador que todos los elementos de entrada dentro de los tags del formulario son parte de una colección de datos que sean enviadas al servidor al momento de hacer el submit.

Pero con Flex la comunicación del lado del servidor es un proceso totalmente distinto, por lo que no hay necesidad de una funcionalidad equivalente a la de los formularios HTML o del botón de submit.

Un contenedor de formulario de Flex es como cualquier otro contenedor en el sentido que lo único que hace es parte del diseño.

El mismo tiene la siguiente colección de tres tags:

Form— El contenedor principal.

FormHeader— Un componente adicional para agregar los cabezales de secciones.

FormItem— Un componente que nos permute asociar texto con cada componente de ingreso de datos.

Listing 4.21 Making use of form-related containers

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

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

En el siguiente recuadro se puede ver una ventana básica de logueo de usuario, un ejemplo de lo expuesto más arriba:



Figure 4.23 Form containers make it easy to lay out input components and give each component a label.

4.6.9 Grid container

El contenedor Grid, es similar a una tabla en HTML, en el sentido que se dispone de un tag grid para dar señal del comienzo de una grilla, existe el tag GridRow para la entrada de columnas y GridItem para el ingreso de datos en cada celda. A continuación un ejemplo de un control para manejo de MP3:

Listing 4.22 Making use of the Grid container

```
<mx:Grid>
  <mx:GridRow>
    <mx:GridItem>
      <s:Button label="Rewind"/>
    </mx:GridItem>
    <mx:GridItem>
      <s:Button label="Play"/>
    </mx:GridItem>
    <mx:GridItem>
      <s:Button label="Forward"/>
    </mx:GridItem>
  </mx:GridRow>
  <mx:GridRow>
    <mx:GridItem colSpan="3">
      <s:Button label="STOP" width="100%"/>
    </mx:GridItem>
  </mx:GridRow>
</mx:Grid>
```

← Create the first row

← Create the second row

Si se desea que una celda abarque más de una columna (como se puede ver en la siguiente imagen), se puede utilizar el atributo colSpan en cualquier ítem del Grid.

El contenedor es sumamente simple, bastante parecido a una tabla en HTML, lo que facilita el ordenado de los elementos.

4.7 Summary

La visualización de Flex está pensada desde el concepto del diseño, y a gran nivel tenemos tres acercamientos: posicionamiento absoluto, basado en restricciones o automático.

Con el posicionamiento absoluto se especifican las coordenadas en las que los componentes deben situarse. Con el posicionamiento restrictivo, necesitamos puntos de anclaje, por ejemplo el centro del navegador. Por último el posicionamiento automático tiene el trabajo pesado, puesto que debe emplear algoritmos de posicionamiento para que el diseño quede presentable.

Nosotros dispondremos de los componentes en contenedores, y las clases de Spark soportaran estos contenedores. Aún quedan algunos componentes Halo/MX que no tienen equivalentes en Spark, podemos nombrar a los que proveen habilidades de diseño únicos, como por ejemplo el contenedor de formulario.

De este modo aprendimos como poner cosas dentro de contenedores, el tipo de cosas que ayudan a mejorar el aspecto visual de una página o formulario, como ser botones, barras deslizantes, cajas de texto, menús desplegables, etc.

5. Desplegando Formularios y capturando entradas de Usuario

Como se mencionó en el capítulo 4, a pesar de que Flex ofrece un componente de forma, su uso es opcional y usted lo encontrará mejor como una herramienta de diseño. En la tierra de Flex hay componentes visuales (por lo general referido como controles) que muestran información y aceptan la entrada del usuario. Además los controles son eventos y controladores de eventos que pueden reconocer y responder a las acciones del usuario, como hacer clic en un ratón. Cuando las funciones de controlador de eventos se ejecutan, acceden a los datos de cualquier fuente que han sido instruidos, no hay etiqueta de formulario principal que contiene todas las entradas como en HTML (véase el gráfico 5.1).

Recuerde, usted no está limitado a un conjunto de controles que viene con Flex.

A diferencia de HTML, lo que limita los controles de interfaz de usuario de la especificación de HTML y el navegador aplicación, Flex le anima a extender un control existente para agregar más funcionalidad o crear sus propios, completamente nuevos controles de interfaz de usuario desde cero.

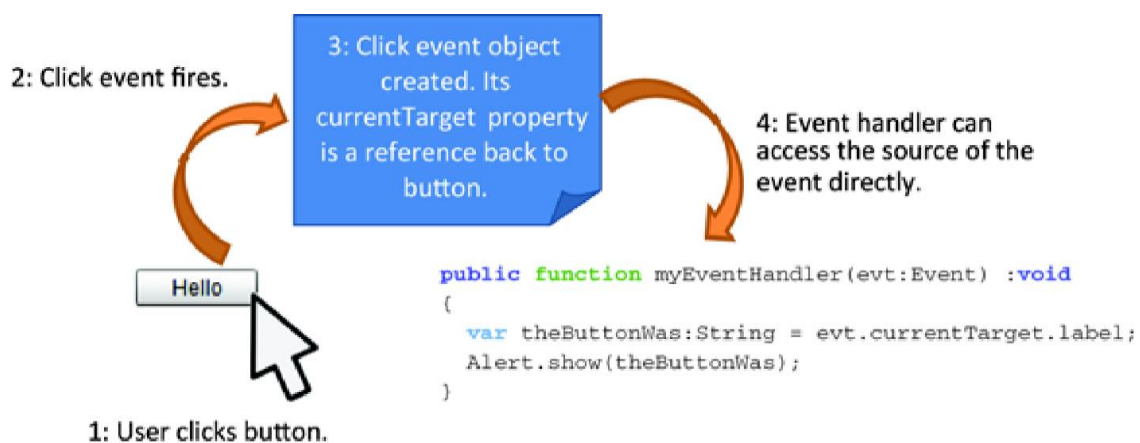


Figura 5.1 Cuando un usuario interactúa con un control, tales como proporcionar formularios relacionados a la entrada (en este caso, un botón), puede configurar un controlador de eventos (que es sólo una función) para controlar el evento. El objeto de evento proporciona una referencia de nuevo a la fuente del evento a través de la propiedad `currentTarget`, que permite que la función de controlador de eventos acceda a la fuente del evento directamente.

NOTA Diga adiós a la noción de formularios HTML. La etiqueta `<Form>` opcional

Flex no hace más que sentar en los componentes de interfaz de usuario denominados controles.

MIGRACIÓN TIP: Similar a lo que vimos en el capítulo anterior, Flex 4 viene con el nuevo paquete de componentes Spark que proporciona controles que obsoleta el anterior Halo. Y al

igual que los contenedores, no todos los controles de Halo tienen el equivalente en Spark. La principal diferencia es que los componentes Spark apoyan un enfoque más flexible para el peinado y el despellejamiento (style y skinning).

Una pieza clave para que el entendimiento es el atributo id invaluable, un identificador de un Componente MXML que le permite acceder a los valores contenidos en el atributo id.

A medida que avanzamos a través del catálogo de controles de entrada de usuario, varios ejemplos harán uso de este atributo id para acceder a sus valores.

5.1 El atributo id

El atributo id puede ser usado en cualquier componente, y se puede acceder de la misma manera que lo haría cualquier otra variable. Da un mecanismo para nombrar únicamente las instancias de un componente, que le permite consultar al componente utilizando explícitamente su identificador único.

A diferencia de HTML, MXML no requiere usar funciones JavaScript's

getElementById() to access an id

```
<fx:Script>

    public function showit()

    {

        mx.controls.Alert.show('idMe is ' + idMe.text);

    }

</fx:Script>

<s:Label id="idMe" text="Hi mom"/>
```

Cuando se trata de formas de construcción, la propiedad id es una manera conveniente de recuperar los valores de un componente de destino, después de haber introducido, vamos a utilizarlo en varios ejemplos.

5.2 Catálogo de controles de Flex

Entradas de texto, casillas de verificación, botones de opción y cuadros combinados desplegables. Sí, Flex tiene todos ellos, y lo mismo ocurre con cualquier otra tecnología de interfaz de usuario. Pero un montón de controles más ingeniosos vienen fuera de la caja, como el editor de texto enriquecido, deslizadores y las teclas numéricas paso a paso.

Comenzamos con los controles basados en texto.

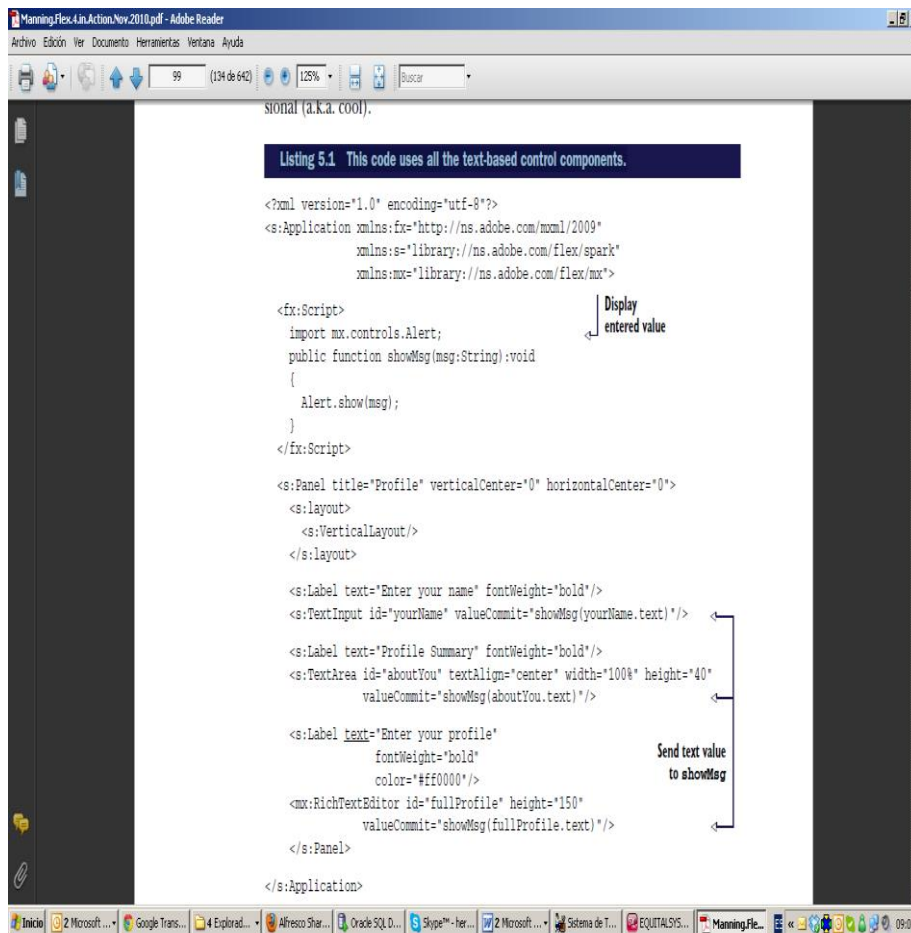
5.2.1 Controles de Texto

El propósito básico de la interfaz de usuario de cualquier aplicación es mostrar el texto. Flex ofrece una variedad de componentes que le permiten capturar y mostrar información textual.

Tabla 5.1 presenta estos controles y sus descripciones.

Control	Paquete	Descripción
Label	Spark	Un control simple que muestra información textual, donde el texto puede envolver. Reemplaza el texto de Halo y Label.
RichText	Spark	Similares a la etiqueta, que no tiene desplazamiento, selección, o capacidades de edición. Se le añade la posibilidad de representar el texto con formato enriquecido.
RichEditableTextSpark		Añade soporte para enlaces URL, desplazamiento, selección y edición. No hay fondo, frontera, o barras de desplazamiento.
TextInput	Spark	Presenta un campo de entrada de texto de una línea. Este control es similar a el <code>type="text"></code> etiqueta <code><input</code> en HTML, y aunque no tiene desplazamiento, tiene una frontera y es personalizable. Sustituye la versión de Halo.
TextArea	Spark	Presenta un campo de entrada de texto de varias filas mediante la ampliación en RichEditableText. Al igual que en la etiqueta <code><textarea></code> en HTML. Compatible con desplazamiento de texto, rico, y los gráficos, es personalizable, y sustituye a la versión de Halo.
RichTextEditor	Halo	Como el nombre implica, un editor robusto que permite al usuario dar formato a su texto. Esto incluye parámetros como el tamaño de color, tipo de letra y texto. HTML no ofrece una característica similar a RichTextEditor nativa.

Vamos a empezar con un caso sencillo de utilizar todos los controles mencionados en la tabla menos el RichText y RichEditableText. En el listado 5.1 tiene una simple aplicación de perfil de usuario que hace uso de varios controles de texto, y usted usa un Panel recipiente para envolver alrededor de estos componentes, lo que hará que se vea más profesional.



El componente Label se utiliza para mostrar una etiqueta descriptiva sobre cada control de entrada. Para que te hagas una idea de lo que Label puede hacer, hemos modificado algunas de sus propiedades de estilo. Cuando un usuario entra en un valor y lo confirma, por ejemplo, haciendo clic en Intro o Tab o un elemento, como un botón de enviar-Flex reconocerá este evento y ejecutará la función ShowMsg () instruida.

Pasando a RichText y RichEditableText (porque no usamos estos aún), estos componentes son parte de un paquete primitivo de Spark en Flex y son una pieza fundación para la construcción de las cosas más inteligentes. Para usarlos usted proporciona datos especificando el atributo de contenido, lo que se puede hacer como una propiedad del componente en sí o como etiqueta secundaria.

El atributo **content** es compatible con las siguientes etiquetas:

- <p> un nuevo párrafo
- para peinar un bloque de texto
-
 para una única línea de rotura

RichText tiene una serie de propiedades de estilo para el ajuste del tamaño de la fuente, el peso, la familia, el color, y así sucesivamente. Piense en la etiqueta span como una forma de modificar el que establece el RichText padre. Por ejemplo, en el listado 5.2 se establece que el color sea azul pero luego se cambia para que la palabra people y negrita.

Del mismo modo, se aumenta el tamaño de la letra en la palabra TIERRA!, Que es evidente en la figura 5.3.

El componente RichEditableText es lo mismo, excepto que permite al usuario seleccione el texto y haga clic en hipervínculos, además de que añade desplazamiento si es necesario y la capacidad de editar el texto. Es efectivamente una versión más simple de TextArea. Todos estos campos de texto se pueden utilizar para capturar una amplia gama de datos.

5.2.1 Controles de Fecha

A diferencia de html que requiere javascript, flex ofrece 2 controles basados en fecha.

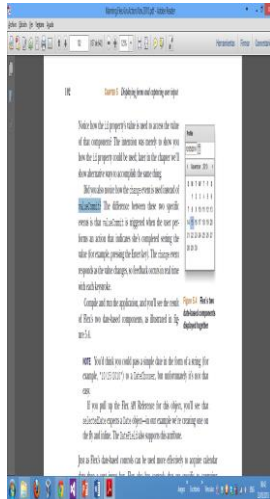
Tabla 5.2 Elementos basadas en fechas

Control	Paquete	Descripción
DateChooser	Halo	Muestra un pequeño calendario similar al de la mayoría de la selección natural basada en HTML Widgets. Una característica notable es que se puede controlar la visualización de rangos de fechas, así como determinar cuáles son seleccionables rangos. Esto es muy útil si desea bloquear fechas como días festivos.
DateField	Halo	El control de entrada que se parece a la entrada <code><type = "text"></code> tag / calendario combinación de imagen que se utiliza en HTML. Permite al usuario introducir una fecha, o, si se selecciona el icono, se muestra el control DateChooser.

Observe cómo se utiliza el valor de la propiedad ID

¿Usted también notará cómo se utiliza el evento de cambio de lugar de valueCommit? La diferencia entre estos dos específicos eventos es que valueCommit se activa cuando el usuario realiza una acción que indica que ha terminado de configurar el valor (por ejemplo, al pulsar la tecla Enter). El evento de cambio responde como los cambios de valor, así que la regeneración se produce en tiempo real con cada pulsación de tecla. Compilar y ejecutar la aplicación, y

verás el resultado de dos componentes basadas en fechas de Flex, como se ilustra en la figura



NOTA Se podría pensar que podría pasar una fecha simple en la forma de una cadena (por ejemplo, "10/15/2010") a un DateChooser, pero por desgracia no es tan fácil. Si acabas de llegar de la referencia API Flex para este objeto, verás que `selectedDate` espera un objeto `Date`-en nuestro ejemplo estamos creando uno de la marcha y en línea. El `DateField` también apoya esta atributo.

5.2.3 Controles Numéricos

Permite capturar números de la entrada del usuario. Pero además y pensando fuera de html también se puede usar para cambiar el tamaño una imagen en tiempo real, filtra un conjunto de datos, o navegar a una sección de la solicitud.

Control	Paquete	Descripción
NumericStepperSpark		Un control sencillo que permite el incremento del usuario y reducir valores. Puede especificar los valores mínimos y máximos permitidos, así como el tamaño de la unidad de la etapa de incremento / decremento.
Spinner	Spark	Casi el mismo que el NumericStepper, excepto que no muestra una entrada de texto al lado de él para mostrar el valor actual. Se supone que debe ser emparejado con otra cosa o extendida.
HSlider	Halo	Permite que el usuario deslice lo que se conoce como un pulgar horizontalmente a lo largo de un seguimiento. Puede controlar los valores mínimos y máximos permitidos, incrementos snap Interval (posiciones en la diapositiva a la que el el pulgar se romperá), y los incrementos visibles (llamados garrapatas). Usted también tiene la opción de crear más de un dedo pulgar, si desea permitir la usuario para especificar múltiples valores (por ejemplo, un intervalo). Un más primitivo Spark versión de este existe.

VSlider

Halo

Idéntica a la HSlider, excepto la orientación de la pista es vertical.

Una versión Spark más primitivo de esta existe.

Listado 5.4 usando control de entrada de numero para capturar valores.

```
<?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.controls.Alert;
public function showMsg(msg:String):void
{
Alert.show(msg);
}
]]>
</fx:Script>
<s:Panel title="Profile">
<s:layout>
<s:HorizontalLayout/>
</s:layout>
<s:VGroup>
<s:Label fontWeight="bold" text="How many kids do you have?"/>
<s:NumericStepper id="kids"
minimum="0"
maximum="10"
stepSize="1"
change="showMsg(kids.value.toString())"/>
<s:HGroup>
<s:Label fontWeight="bold" text="Kids in college?"/>
<s:Spinner minimum="0" maximum="10" id="collegeKids"/>
<s:Label text="{collegeKids.value} in college"/>
</s:HGroup>
<s:Label fontWeight="bold" text="How long is your commute (mins)?"/>
<mx:HSlider id="commuteTimeRange" minimum="0" maximum="180"
snapInterval="5"
tickInterval="15"

labels="[0 mins,180 mins]"

thumbCount="2"

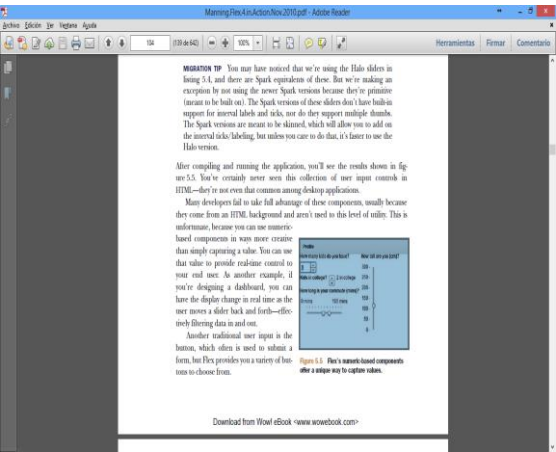
change="showMsg(commuteTimeRange.values.toString())"/>
</mx:HSlider>
</s:VGroup>
<s:VGroup>
<s:Label fontWeight="bold" text="How tall are you (cm)?"/>
<mx:VSlider id="yourHeight" minimum="0" maximum="300"
tickInterval="50" snapInterval="1"
labels="[0,50,100,150,200,250,300]"
change="showMsg(yourHeight.values.toString())"/>
</s:VGroup>
</s:Panel>
</s:Application>
```

Puede seleccionar en unidades de 5 -1. y marcadores pantalla cada 15 unidades -2. La pantalla es de dos etiquetas uniformemente espaciadas (una en cada extremo) -3. En el control deslizante vertical -4. los marcadores son más separados que en el control deslizante horizontal. Por último, se muestra siete espaciados uniformemente etiquetas -5.

Migration TIP:

Usted puede haber notado que estamos usando los controles deslizantes de Halo en listado 5.4, y hay equivalentes Spark de estos. Pero estamos haciendo una excepción al no utilizar las versiones más recientes de Spark porque son primitivos (destinado a ser construido sobre). Las

versiones Spark de estos controles no tienen una función de soporte para las etiquetas de intervalo y garrapatas (ticks), ni admiten varios pulgares.



Se pueden utilizar componentes de entrada de número, de formas más creativas que la simple captura de un valor. Usted puede usar el valor para proporcionar un control en tiempo real a el usuario final. Otro ejemplo, si usted está diseñando un cuadro de mandos, puede que el cambio de visualización en tiempo real, el usuario pasa el cursor hacia atrás y adelante -eficacia en filtrado de los datos de entrada y salida.

Otra entrada del usuario es el tradicional botón, que a menudo se utiliza para enviar un formulario, pero Flex le ofrece una variedad de botones.

5.2.1 Explorando Botones Flex

Flex va más allá del botón llanura de edad que ofrece HTML y agrega una colección de opciones interesantes (véase el cuadro 5.4). Los botones son uno de los baluartes de interfaces gráficas de usuario, pero pueden ser un poco aburrido. Flex los lleva a un nivel más alto con un número de variaciones.

Control	Paquete	Descripción
Button	Spark	El botón estándar, de uso múltiple para aceptar un clic del ratón interacción. Botones de Flex se han incorporado en el apoyo que le permite fácilmente añadir una imagen en el botón. En HTML se puede utilizar una imagen para crear un botón de pseudo-texto + imagen, pero el texto será estática. Con Flex, el texto sigue siendo dinámico.
ButtonBar	Spark	Dinámicamente genera una serie de botones en base a una matriz. usted podría hacer lo mismo, generando dinámicamente botones de su cuenta, pero no sólo es una barra de botones fáciles de implementar, añade un poco de estilo por el redondeo de los

bordes más exteriores de los botones para dar la apariencia de una única barra de menú, dividido en secciones.

LinkButton	Halo	<p>La versión de Flex de un vínculo HTML. Si acabas de llegar de la referencia de clase</p> <p>para este objeto (en el API Flex), verás que es un descendiente del botón, pero con estilo y extendida a comportarse como un Enlace HTML por tener un fondo transparente y sin bordes.</p>
LinkBar	Halo	<p>Similar a la relación entre un botón y un</p> <p>Barra de botones, la barra de enlaces es una manera fácil de crear una serie de</p> <p>LinkButtons.</p>
ToggleButtonBar	Halo	<p>Casi idéntica a una barra de botones, salvo que persista la selección-</p> <p>Si pulsa uno de los botones, se queda abajo (y resaltado) hasta que se cambie a otro botón. Piense en ello como una radio botón con una etiqueta en el interior.</p>
PopUpButton	Halo	<p>Un botón dual que combina la funcionalidad de dos botones-el lado izquierdo actúa como un botón normal, el lado derecho invoca otro objeto de interfaz de usuario (cualquier cosa derivada de la UIComponent objeto). Este se utiliza típicamente para crear una selección múltiple Botón que se parece a un botón y un cuadro combinado fusionado en uno.</p>
PopUpMenuButtonHalo		<p>Un descendiente del PopUpButton, se orienta específicamente a crear un menú desplegable.</p>

El Listing 6.1 muestra un ejemplo del componente Validator, mostrando su forma de acción más básica. Debemos ver que el componente se encuentra dentro de un tag Declarations porque es un tag no visual.

Listing 6.1 A basic validator in action

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

Nonvisuals to be declared

Una vez compilado y corrido el código se verá una aplicación similar a la que se muestra en la figura 6.2. Debemos clikea la caja de texto cuando se nos pida entrar el nombre. Debemos notar que al mover el mouse sobre el campo el foco cambia. Adicionalmente el borde del campo cambia y se vuelve rojo y aparece un pop-up que avisa que el campo es requerido. Lo siguiente es un sumario de lo que pasa:

- Un cambio de foco puede suceder al moverse de una parte del UI a otra. Esto puede darse por un click, moverse con el tabulador o presionar enter, y hace que el input comitee un valor.
- Lo que haya en l caja de testo es comiteado, en este caso, el campo está vacío.
- El valor por defecto de triggerEvent es valueCommit. Los validadores para el se aplicarán a la propiedad text del valueCommit.
- Cuando la propiedad text de un textInput esta vacia, esto hace que las validaciones fallen y se muestre el campo bordeado de rojo.

Como todos los validadores heredan de la clase Validator todas pueden usar este mismo proceso.

La diferencia esta en que podemos usar diferentes criterios para las validaciones.

Por ejemploel StringValidator puede el número de caracteres, devolviendo un booleano.



Figure 6.2 The Validator component highlights a field with a message when validation fails.

6.2.2 StringValidator

Este es un validador de propósitos generales, que realiza todo lo que hace su padre, y agrega las funciones de chequear si el String es muy corto o muy largo. Usando las referencias de Flex podemos encontrar propiedades como minLength, tooShort-Error, maxLength, tooLong-Error, que permiten configurarlos para mostrar los mensajes cuando se disparen.

El Listing 6.2, se usa un StringValidator para chequear que el input no tenga menos de 3 caracteres, ni más de 20.

Listing 6.2 Using StringValidator to check character count

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

Used to trigger validation

En la figura 6.3 se usa el click del botón para realizar el trigger de la validación. Debemos notar que hacemos 2 cosas de manera diferente.

- Por defecto el validador está esperando que el evento de trigger ocurra en la validación del componente (normalmente cuando el usuario comitea los valores del input). Esto debe sobrescribirse y setear la propiedad trigger en el botón en sí.
- Por defecto el evento trigger es valueCommit. Pero al usar un botón, se desea que el evento sea clickButton antes que la presión del Tab o el Enter.



Figure 6.3 This StringValidator warns the user that the minimum required number of characters hasn't been entered into the field.

Usar validadores es sencillo sin usar mucho dinamismo. Son orientados a la utilidad y tiene un propósito claro, haciéndolos un mecanismo fácil de usar. Sabiendo que existe este tipo de validadores, está bien asumir, que también hay validadores basados en números.

6.2.3 NumberValidator

Como lo dice su nombre, el NumberValidator evalúa información numérica.

Es muy versátil, permite saber si lo que se ingresa es muy grande, o muy pequeño, si solo es numérico, o que no contenga negativos. De la misma forma, es suficientemente inteligente para reconocer la separación en miles.

Por internacionalización, provee las propiedades thousand-separator y decimalSeparator

Con todas las propiedades disponibles, se convierte en un validador flexible. Lo que presenta el Listing 6.3, se muestra un código que hace que el rango de entrada esté entre 5 y 110, y debe ser un entero.

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

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

La figura 6.4 muestra lo que sucede si se ingresa un número negativo.

El atributo `allowNegative` es redundante en el ejemplo previo. Puede utilizarse igualmente sin perjuicio, esto solo indicará al usuario, como se muestra en la imagen 6.5, que no se admiten negativos.



Figure 6.4 NumberValidator used to check whether a user has typed in a realistic age

6.2.4 DateValidator

Esto nos ayuda a trabajar con inputs diseñados para recibir datos de fechas. Una de las posibilidades del componente es recibir tres entradas diferentes que guardan el mes, el día y el año, a diferencia de otros controladores, que solo pueden tomar 1 fuente.

Se puede usar una sola fuente standard, que guarde texto de forma mm/dd/yyyy o usando una colección de inputs para capturar cada parte, y configurar el DateValidator para entender cada parte.

Se muestra un ejemplo en el Listing 6.4

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

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

En la figura 6.5 podemos ver como el DataValidator responde ya que diciembre no puede tener 39 días

Incluso se señala donde está el error al usuario.

Este ejemplo será llevado al siguiente nivel en el Listing 6.5, usando algunas propiedades avanzadas.



Figure 6.5 DateValidator doesn't like the date entered.

Listing 6.5 Advanced properties example

```
<?xml version="1.0"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">
  <fx:Declarations>
    <mx:DateValidator
      monthSource="{month}" monthProperty="value"
      daySource="{day}" dayProperty="value"
      yearSource="{year}" yearProperty="text"
      trigger="{submitButton}" triggerEvent="click"/>
  </fx:Declarations>
  <s:HGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="Month:"/>
    <s:NumericStepper id="month" />
    <s:Label text="Day:"/>
    <s:NumericStepper id="day" />
    <s:Label text="Year:"/>
    <s:TextInput id="year" width="60"/>
    <s:Button label="Submit" id="submitButton"/>
  </s:HGroup>
</s:Application>
```

Identifying a date
split across
multiple inputs

Cuando testeamos el código superior nos muestra que podemos entrar 3 campos (mes, día, año).

En el ejemplo la entrada está mal, y podemos ver el error en el pop-up en la imagen 6.6

El DataValidator es una opción viable para chequear entradas desde 1 fuente o desde 3.

En una aplicación real, no usaremos un InputText para capturar la información, sino más bien una DateField o un Date-Choser.



Figure 6.6 A DateValidator can be used to validate the separate form inputs that make up the complete date.

6.2.5 EmailValidator

Verificar una dirección de internet es básico en cualquier operación de registro, y en la mayoría de las webs se hace desde el lado del servidor. El MailValidator es bastante competente en las verificaciones, asegurándose que tienes el @ antes del dominio.

Las únicas propiedades configurables son errores. Esto es porque un email siempre es igual en todos lados. Es limitado en su cuerpo y fácil de verificar en su estructura. No se necesita mucho más que eso.

Listing 6.6 Verifying that what is entered is a properly structured email address

```
<?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:EmailValidator source="{email}" property="text"
                      invalidCharError="You've got some funky characters"
                      trigger="{submitButton}" triggerEvent="click"/>
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="Email:"/>
    <s:TextInput id="email"/>
    <s:Button label="Submit" id="submitButton"/>
  </s:VGroup>
</s:Application>
```

Custom warning message

Una vez compilado y corrido el código superior, al entrar un email invalido se mostrará el mensaje de la figura 6.7

Validar un mail en tiempo real no es algo muy usado en el mundo web, y suele realizarse en el servidor de backEnd.



Figure 6.7 The EmailValidator flags this address, because there's an invalid space character within it.

6.2.6 CreditCardValidator

Ahora abordaremos algunos validadores interesantes, los cuales no están disponibles en muchos lenguajes de programación. Es el caso del CreditCardValidator que usa el algoritmo Luhn mod10 para verificar el número y el tipo de la tarjeta de crédito suministrada por el usuario.

El validador necesita 2 fuentes de datos

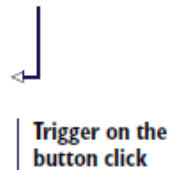
- El tipo de tarjeta (Flex soporta American E., Diners, Discover, MasterCard y VISA)
- El numero de la tarjeta de crédito.

Estas propiedades dan la habilidad de realizar los controles necesarios.

En el Listing 6.7 en lugar de hardcodear muchas de las propiedades, se le permite al usuario seleccionar el tipo de tarjeta. Entonces, usando un data binding se puede actualizar la propiedad deseada en el CreditCardValidator

Listing 6.7 Using data binding to know which type of credit card to validate

```
<?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:CreditCardValidator
      cardNumberSource="{cardNumber}"
      cardNumberProperty="text"
      cardTypeSource="{cardType}"
      cardTypeProperty="selectedItem"
      trigger="{submitButton}"
      triggerEvent="click"/>
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:DropDownList id="cardType" width="150">
      <mx:ArrayCollection>
        <fx:String>American Express</fx:String>
        <fx:String>Visa</fx:String>
        <fx:String>Diners Club</fx:String>
        <fx:String>Discover</fx:String>
        <fx:String>MasterCard</fx:String>
      </mx:ArrayCollection>
    </s:DropDownList>
    <s:Label text="Card Number:"/>
    <s:TextInput id="cardNumber"/>
    <s:Button label="Submit" id="submitButton"/>
  </s:VGroup>
</s:Application>
```



Al correr el código superior e insertar 4 bloques de dígitos para representar un nro de tarjeta de crédito. A menos que se escriba un número de Visa se obtendrá un warning como el de la figura 6.8

A screenshot of the application interface. At the top is a dropdown menu with "Visa" selected. Below it is a label "Card Number:". Under the label is a text input field containing "1234-4568-1232-1231". At the bottom is a "Submit" button.

The credit card number is invalid.

Figure 6.8
CreditCardValidator
used to pair up a credit card
type and credit card number

Cuando realizamos transacciones online, debemos poder validar los tipos de moneda. Si el sistema acepta ordenes internacionales, el siguiente paso es verificar el tipo de moneda y su tipo de cambio.

6.2.7 CurrencyValidator

Si estamos creando una aplicación internacional, debemos contar con el esfuerzo extra de saber como se expresan las mismas cosas alrededor del mundo.

El CurrencyValidator provee una considerable ayuda porque está diseñado para validar si la expresión contiene un tipo de moneda. Es similar al NumberValidator porque admite chequear por expresiones decimales, valores minimos y máximos, separación de decimales. Pero también agrega el reconocimiento de tipo de moneda y su posición en el input.

En el Listing 6.8, usamos el CurrencyValidator para verificar que solo existen 2 lugares decimales.

Listing 6.8 Using a CurrencyValidator to make sure money values are correct

```
<?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:CurrencyValidator
      source="{income}" property="text" allowNegative="false"
      negativeError="You pay your employer?"
      precision="2" precisionError="Just 2 decimals please."
      trigger="{submitButton}" triggerEvent="click" />
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="How much do you make?" />
    <s:TextInput id="income" />
    <s:Button label="Submit" id="submitButton" />
  </s:VGroup>
</s:Application>
```

Accept only two decimal places

En la figura 6.9 podemos ver un caso de error. La cantidad de dinero ingresada es incorrecta, ya que contiene 3 lugares decimales. Esto dispara un mensaje en el pop-up

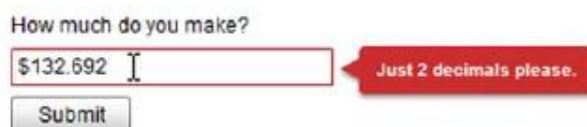


Figure 6.9 This example fails validation because the CurrencyValidator is configured to check for two decimal places only.

Similar al CurrencyValidator es el PhoneNumber-Validator. Permite facilidades para la comunicación global, validando números de teléfono internacionales.

6.2.8 PhoneNumberValidator

Este es un validador simple. Se ingresa un numero válido o no. Elte validador puede reconocer números originados internacionalmente, asi como en America del Norte.

Solo debemos fijarnos en 2 cosas:

- El numero debe tener al menos 10 dígitos.
- Cada carácter que se use debe ser valido(-,+,())

El Listing 6.9 nos muestra un ejemplo simple de chequeo sobre si el número es un número valido de la zona tropical.

Listing 6.9 Checking whether the phone number entered is a valid format

```
<?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:PhoneNumberValidator
      source="{phone}" property="text"
      trigger="{submitButton}" triggerEvent="click" />
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="What number can we reach you at?" />
    <s:TextInput id="phone" />
    <s:Button label="Submit" id="submitButton" />
  </s:VGroup>
</s:Application>
```

Compilado el código anterior y corrida la aplicación, debemos probar formas válidas e invalidas de ingresar teléfonos. En la figura 6.10 se muestra uno de los casos de error.



Tratar de escribir nuestro propio Validador de números puede ser engorroso, por esta razón es que los números de teléfono están estructurados de varias maneras. Si no podemos dar con la validación correcta, siempre podemos usar un validador de Expresiones regulares.

6.2.9 RegExpValidator

RegEx (regular Expressions) es un lenguaje más bien antiguo, pero poderoso, prácticamente sin límites. El validador en este caso compara una expresión RegEx contra un valor, para determinar cualquier cosa que machee con la expresión.

Usando las expresiones apropiadamente con las banderas RegEx de manera correcta(Pueden ser usadas para evitar ciertos comportamientos, búsqueda global o like), podemos definir un patrón de búsqueda.

Por ejemplo si deseáramos validar el número de seguridad social (SSN), podríamos buscar si algo de lo que ha ingresado coincide con el patrón del SSN, como se muestra en el Listing 6.10.

Listing 6.10 RegExpValidator matches text against a RegEx pattern

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

Global, multiline,
ignore case

Mirando en la imagen 6.1, podemos ver que el resultado es igual a cualquier otro validador, salvo que detrás de cámaras el RegExValidator trabaja un poco diferente. Esto es porque las expresiones regulares trabajan en base a patrones.

Social Security Number:

Your SSN is unrecognized.

Figure 6.11 A regular expression used to validate an SSN

Se puede usar la siguiente habilidad para ver en cuantas ocasiones se encontraron patrones que concuerdan y donde están ubicados en el String. Para esto, se debe usar un poco de Action-Script para crear un handler de eventos de validación como se muestra a continuación.

Listing 6.11 Using RegExpValidator to find all the matches on the pattern

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

  <fx:Script>
    <![CDATA[
      import mx.events.ValidationResultEvent;
      import mx.validators.RegExpValidationResult;
      import mx.controls.Alert;
      private function handleValidation(event:ValidationResultEvent):void

      {
        var oneResult:RegExpValidationResult;
        for (var i:int = 0; i < event.results.length; i++)
        {
          oneResult = event.results[i];
          Alert.show("Found a match at Index:" +
                    oneResult.matchedIndex +
                    "\non the characters of:" + oneResult.matchedString
                    , "RegExp Results",Alert.NONMODAL);
        }
      }
    ]]>
  </fx:Script>
  <fx:Declarations>
    <mx:RegExpValidator source="{test}" property="text" flags="gmi"
                       valid="handleValidation(event)"
                       expression="m[ai]n" noMatchError="I don't like it!"
                       trigger="{submitButton}" triggerEvent="click"/>
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="Try me:"/>
    <s:TextInput id="test" />
    <s:Button label="Submit" id="submitButton"/>
  </s:VGroup>
</s:Application>
```

Import necessary libraries

Evaluate each match

Loop over each pattern match

Display information about match

En el ejemplo previo se retornan un par de patrones coincidentes se retornan (figura 6.12) y el Action-Script realiza el recorrido entre ellos.

Las Expresiones regulares son poderosas, al punto de que si se pudiera usar solo un validador, deberíamos elegir este.

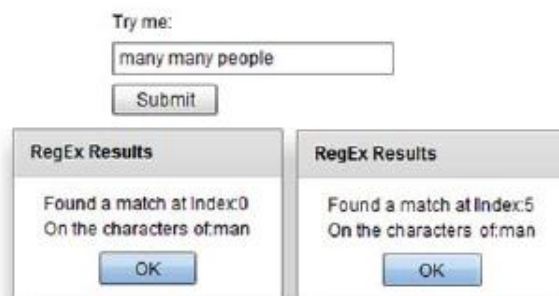


Figure 6.12 The submitted text pattern results in two matches found by the regular expression.

6.2.10 SocialSecurityValidator

En el ejemplo anterior se usó un RegExValidator para validar un SSN, pero Flex tiene incorporado un validador para eso.

Funciona esperando una serie de dígitos y un separador de cada bloque en el formato 'xxx-xx-xxxx'. Adicionalmente el SecuritySocialValidator permite definir el separador por defecto, y posee una regla implícita de que los 3 primeros dígitos no pueden ser 000.

Poemos comparar el Listing 6.11 que usa RegExValidator con este, el 6.12 que usa SocialSecurityValidator, para ver las diferencias

Listing 6.12 SocialSecurityValidator alerts user if SSN is improperly formatted.

```
<?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:SocialSecurityValidator
      source="{ssn}" property="text"
      trigger="{submitButton}" triggerEvent="click" />
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="Try me:" />
    <s:TextInput id="ssn" />
    <s:Button label="Submit" id="submitButton"/>
  </s:VGroup>
</s:Application>
```

Una vez compilado y corrido el código, nos disparara un erro ante cualquier input que no tenga el formato adecuado. En la figura 6.13 vemos un ejemplo.

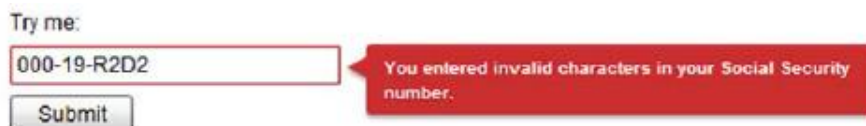


Figure 6.13 Busted! The SocialSecurityValidator detected that the value entered isn't a valid SSN.

6.2.11 ZipCodeValidator

El último de los validadores incluidos es el ZipCodeValidator. Sirve para ver si un código postal Norteamericano o canadiense está correctamente construido.

Para un código postal acepta el formato de 5 dígitos o el de 5+4 dígitos. Para saber que esta validando puede usar la propiedad dominio. El resto de las propiedades solo sirven para sobrescribir el mensaje de error.

El Listing 6.13 nos muestra un ZipCodeValidator.

Listing 6.13 Trying out the ZipCodeValidator to test for U.S. ZIP Codes

```
<?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:ZipCodeValidator
      source="{zipcode}" property="text" domain="US Only"
      trigger="{submitButton}" triggerEvent="click"/>
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="What's your Zip Code (US Only)?" />
    <s:TextInput id="zipcode" />
    <s:Button label="Submit" id="submitButton" />
  </s:VGroup>
</s:Application>
```

Este es otro de los validadores que querremos experimentar. Al entrar un código incorrecto se desplegará un mensaje como en la figura inferior.



Figure 6.14 The ZipCodeValidator has been configured to check only for ZIP Codes, so this Canadian postal code fails.

Como el ZipCodeValidator está configurado para aceptar códigos postales de Norteamérica, al ingresar uno canadiense se dispara ese error.

Esto encasilla todos los validadores fuera de caja de Flex.

Ahora, nos acercaremos a otro tipo de validaciones.

6.3 Validaciones en tiempo real

Para validar en tiempo real, o para captar errores cuando están sucediendo, requiere que el componente sea “escuchado” (listen) a cada cambio que ocurra cuando los datos van entrando.

Para hacer esto, podemos monitorear los cambios con el evento CHANGE como trigger.

En el siguiente Listen se da un ejemplo de lo anterior.

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

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

Use change event to trigger validation

En el Listing 6.14, al momento que el usuario comienza a tipear, el campo queda en rojo, hasta que el usuario ingrese por lo menos 5 caracteres. Es una forma efectiva de mantener al usuario tipeando hasta que complete el mínimo requerido de dígitos, aunque puede ser molesto.

Es menos agresivo con el usuario, realizar las validaciones cuando comitea la información.

Para poder lograr esto, se ha de cambiar el efecto trigger de CHANGE a VALUE-COMMIT.

```
<mx:StringValidator source="{address}" minLength="5" property="text"
                  trigger="{address}" triggerEvent="valueCommit" />
```

Esto añade una interactividad deseada con el usuario. En algunos casos deseamos que se valide si ya se han llenado ciertos campos solamente, en ese caso, debemos usar una validación PASSTHROUGH

6.5 Pass-through validation

Las validaciones passThrough son las más pasivas. Se espera a tener todo el formulario completo para enviarlo y después realizar las validaciones. Esta es una práctica bastante común en el mundo web.

El modo más normal de dispararlos es con un botón. Por lo tanto, se han de cambiar los eventos trigger (trigger y triggerEvent) por el click del botón en la interfaz deseada.

En un form actual, podemos tener varios input configurados para validarse con el click del botón.

El Listing 6.15 muestra cuántos múltiples validadores pueden ser unidos al click del botón para chequear las cajas de texto necesarias.

Listing 6.15 Using a submit button to validate form fields

```
<?xml version="1.0"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">
  <fx:Declarations>
    <mx:StringValidator source="{username}"
                       property="text"
                       minLength="6"
                       trigger="{submitButton}"
                       triggerEvent="click"/>
    <mx:EmailValidator source="{email}"
                      property="text"
                      trigger="{submitButton}"
                      triggerEvent="click"/>
  </fx:Declarations>
  <s:VGroup horizontalCenter="0" verticalCenter="0">
    <s:Label text="Enter your email:"/>
    <s:TextInput id="email"/>
    <s:Label text="Enter your username:"/>
    <s:TextInput id="username" />
    <s:Button label="Submit" id="submitButton"/>
  </s:VGroup>
</s:Application>
```

Validators set up for form fields

Validate on button click

Input form fields have validators

6.6 Scripted validation

Todos los validadores vistos hasta aquí poseen un ActionScript equivalente para proveer máxima conveniencia y control.

Se usa una versión ActionScript cuando se desea reusar un control sobre varios componentes., o incluso valores que no sean inputs.

Listing 6.16 Using ActionScript to validate values at will

```

<?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.validators.EmailValidator;
        import mx.validators.StringValidator;
        import mx.controls.Alert;
        import mx.events.ValidationResultEvent;
        public var emailVal:EmailValidator = new EmailValidator();
        public var stringVal:StringValidator = new StringValidator();
        public function validateForm():void
        {
            var valResult:ValidationResultEvent;
            stringVal.source = username;
            stringVal.property = "text";
            stringVal.minLength=6;
            emailVal.source = email;
            emailVal.property = "text";

            valResult = emailVal.validate();
            if(valResult.type == "invalid")
            {
                Alert.show("Please fix your Email address.");
            }
            else
            {
                valResult = stringVal.validate();
                if(valResult.type == "invalid")
                {
                    Alert.show("Please fix your Username.");
                }
            }
        }
    ]]>
    </fx:Script>
    <s:VGroup horizontalCenter="0" verticalCenter="0">
        <s:Label text="Email:"/>
        <s:TextInput id="email"/>
        <s:Label text="Enter your username:"/>
        <s:TextInput id="username" />
        <s:Button label="Submit" id="submitButton" click="validateForm()" />
    </s:VGroup>
</s:Application>

```

Import needed ActionScript classes
1 Use two validator
2 Use two TextInputs
3 Invoke validateForm() function
4 Validate email address
5 Check username

En el Listing superior tenemos 2 validators (1) apuntando a 2 textInputs (2). Cuando se presiona el botón de submit, la función validateForm() es invocada (3). La dirección de mail es validada primero (4), si esta correcta, el validateForm() se mueve a siguiente inputField y chequea el userName(5)

Hasta ahora hemos cubierto los tipos de validaciones y como se usan. Ahora veremos unos tips antes de cerrar el capítulo.

6.7 Validation tidbits

6.7.1 Un validador siempre chequea todos los criterios?

Si se da el caso en que más de un requerimiento falle en la misma comprobación pasará lo siguiente:

Flex chequeará 1 a 1 los criterios hasta que encuentre un error. En caso de encontrarlo desplegará el mensaje correspondiente.

Aunque nosotros vimos el uso básico de los criterios, debemos tener en cuenta que pueden usarse para cubrir una gran cantidad de otros criterios, por ejemplo, que una fecha cae en un rango específico.

6.7.2 Controlar validación via trigger.

Para mantenerlo simple, hemos etiquetado algunos de las formas de validación (por ejemplo la validación en tiempo real), pero usando las propiedades Trigger y TriggerEvent podemos comenzar la validación cuando deseemos.

En nuestro ejemplo usamos el id de un botón para el trigger, y el TriggerEvent será el click.

Como sabemos si un evento esta listo para dispararse? En FlexBuilder se pueden ver los componentes en los cuales estemos interesados. Para esto debemos usar el API Reference.

Esta documentación nos permitirá saber que propiedades tiene cada uno de los validadores en particular. Además de saber a que componentes puede afectar y como puede ser disparado. Toda esta información está disponible en la API Reference.

6.8 Summary

Validation es un mecanismo apreciable y positivo que reduce la interacción obvia entre el usuario y el servidor. Validando en el lado del cliente agregamos usabilidad y reducimos tiempos al usuario.

También nos ofrece ayuda para liberar trabajo de nuestro backEnd, ya que le damos la información lista desde el frontEnd.

Flex cuenta con una enorme cantidad de validadores incorporados, desde básicos para texto a mas específico como el de SSN.

Se pueden usar muchos tipos de validación, en tiempo real, o disparando tod junto cuando se pulsa un botón.

También vimos que se puede usar ActionScript para crear validaciones y usarlas contra datos que deseemos chequear.

7 Formatters

- Formatear datos “crudos” con los Built-in Formatters de Flex
- Tipos de Formatters
- Real-Time Formatting y Scripted Formatting
- Manejar errores de Formateo

Los Formatters son una clase de objetos que toman “datos crudos” y la transforman en datos con formato visual para Presentación. Desde una perspectiva de uso, se pueden comparar con los Validators, ya que están implementados de forma similar.

Los Formatters se pueden configurar de dos formas:

- Tiempo-Real: el formateo es conducido en la ejecución automáticamente por medio de “data binding”(vinculación de datos).
 - Encriptado(Scripted): Mediante ActionScript se pueden crear instancias de Formatters cuando se precisan y formatear los datos de forma acorde
-
- ✓ Muy fáciles de Usar: Se ingresan datos crudos y se estructuran. Obtenemos contenido legible
 - ✓ Los Formatters solo funcionan con “datos crudos” (sin formato). Se debe poner especial énfasis en evitar formatear datos ya formateados.
 - ✓ Los Formatters pueden usarse de forma independiente y se pueden utilizar en varios escenarios. Un caso común de uso es recuperar información de una base de datos y enviarla a un Formatter para presentarla de una forma cómoda. Esto se suele hacer mediante Internet, para para simplificar en los ejemplos se utilizarán XML estáticos para simular los datos.

7.1 Built-in Formatters(Formatters Constructores)

Flex provee ciertos Built-in Formatters que pueden ser usados rápidamente. Heredan de Formatter y por defecto vienen con un conjunto común de funciones, eventos y propiedades.

Estas clases están enfocadas para formatear datos en contextos específicos, y poseen propiedades únicas para dar dirección a estos escenarios.

Para conocer estas propiedades debemos utilizar la Referencia del Lenguaje de la API.

7.1.1 Formatters, empezando por el Padre

Formatter: es la clase padre y sirve como template para el resto de Formatters. Como clase base, no resulta de gran utilidad por sí misma, aunque presenta una función clave:

- `Format()` : toma un objeto que debe ser formateado y retorna un String como resultado.

También presenta una propiedad importante, ya que determina la fuente de los errores de formateo.

- `String error`: Se usa esta propiedad para visualizar los errores de Formateo. Se puede saber cuándo ocurrió un error de formateo ya que en ese caso `format()` devuelve un String vacío.

Number Formatter: Es de los Formatter más utilizados. Maneja detalles en los valores numéricos, como el punto de precisión para decimales, o el caracter utilizado para separar milenios.

<< Tabla con propiedades >>

Ejemplo:

```
<?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>
<fx:XML id="myData">
<root>
<forsale>
<item name="weight" value="32.5698" />
</forsale>
</root>
</fx:XML>
<mx:NumberFormatter id="fmtNumber" precision="2"/>
</fx:Declarations>
<s:Label
text="Weight {fmtNumber.format(myData.forsale.item.@value)}lbs"/>
</s:Application>
```

<< Mensajes de Error >>

CurrencyFormatter: Dado un valor numérico crudo, este Formatter organiza la data para mostrarla como una expresión monetaria conocida.

CurrencyFormatter contiene todas las propiedades de NumberFormatter pero agrega dos propiedades específicas para manejar valores monetarios:

<<Funciones extra>>

Ejemplo:

```
<?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>
<fx:XML id="myData">
<root>
<forsale>
<item name="Laptop" price="599.99" />
</forsale>
</root>
</fx:XML>
<mx:CurrencyFormatter id="fmtCurrency" precision="2"/>
</fx:Declarations>
<s:Label text="Laptop Price
{fmtCurrency.format(myData.forsale.item.@price)}"/>
</s:Application>
```

Este Formatter usa los mismos mensajes de error que NumberFormatter.

DateFormatter: como su nombre lo indica, este Formatter provee control sobre cómo presentar fechas.

Este Formatter tiene una propiedad particular:

- String formatString: es una máscara patrón para el formateo de fechas.

La clave en este Formatter son los caracteres especiales para éste, y la forma en que se combinen.

<<Tabla de caracteres >>

Como en el resto de Formatter 's, este presenta errores:

- InvalidValue – DateFormatter no reconoció los datos como una fecha. Este Formatter acepta o bien un objeto Date o un String que contenga una fecha.
- InvalidFormat – Se ingresó como formatString un formato inválido.

Ejemplo:

```
<?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[
[Bindable]
public var sDate:String = "12/01/08 12:42";
[Bindable]
public var dDate:Date = new Date("12/01/08 12:42");
]]>
</fx:Script>
<fx:Declarations>
<mx DateFormatter id="fmtDate" formatString="MM.DD.YY"/>
</fx:Declarations>
```

```

<s:VGroup verticalCenter="0" horizontalCenter="0" horizontalAlign="center">
<s:Label text="Formatting the Date as a String:
{fmtDate.format(sDate)}" />
<s:Label text="Formatting the Date as a Date object:
{fmtDate.format(dDate)}" />
</s:VGroup>
</s:Application>

```

El Formatter del ejemplo no está asociado directamente con un tipo en particular. Puede usarse tanto para Date como para String.

Se puede usar data en formato XML pero se debe ser cuidadoso ya que puede llevar a errores, como en el siguiente caso:

(Esto da error)

```

<fx:Declarations>
<fx:XML id="myData">
<root>
<info>
<item lastvisit="12/01/08 12:42"/>
</info>
</root>
</fx:XML>
<mx:DateFormatter id="fmtDate" formatString="MM.DD.YY"/>
</fx:Declarations>
<s:Label
text="{fmtDate.format(myData.info.item.@lastvisit)}" />

```

Esto no funciona, ya que internamente Flex transforma la data XML en una colección de Objetos pero no específicamente String ni Date, que son los tipos para los cuales funciona DateFormatter. Sin embargo se puede castear el valor a formatear de la siguiente forma:

```

<s:Label text="{fmtDate.format(String(myData.info.item.@lastvisit))}" />

```

PhoneFormatter: La principal utilidad que presenta este Formatter es la de poder tomar , por ej, números de teléfono almacenados en una base de datos como simples dígitos , y transformarlos a un formato más familiar en cuanto a números telefónicos.

Este Formatter se comporta de forma similar al DateFormatter, ya que tiene su propio *formatString* para especificar el patrón que se aplica al número, pero también porque acepta dos tipos de datos en particular: String y Number.

<< Propiedades de PhoneFormatter >>

Ejemplo de patrones de formato y sus efectos:

<<Ejemplos de patrones>>

Ejemplo de uso:

```

<?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>
<fx:XML id="myData">

```

```

<root>
<contactlist>
<item name="contact" phone="2016679872" />
</contactlist>
</root>
</fx:XML>
<mx:PhoneFormatter id="fmtNumber" formatString="(###) ###-####"/>
</fx:Declarations>
<s:Label text="Contact Phone:
{fmtNumber.format(myData.contactlist.item.@phone) }"/>
</s:Application>

```

Los errores de este Formatter son:

- **InvalidValue:** Este Formatter acepta sólo String o Number. Cualquier otro tipo ingresado genera este error. Si se ingresa un String, este debe tener igual cantidad de caracteres que formatString.
- **InvalidFormat:** Indica un error en formatString. Puede ser causado por tener caracteres inválidos o porque el código de área no contiene 3 dígitos.

ZipCodeFormatter: Este Formatter es útil cuando se quiere formatear códigos postales americanos o canadienses.

Tiene su propio formatString:

- **String formatString:** la máscara patrón que se va a aplicar para el código postal. El patrón aplicado debe ser un formato de código postal americano o canadiense conocido. Por defecto es ##### .

El código postal americano puede ser de 5 o 9 dígitos. Si se elige un patrón de 9 dígitos pero solo se ingresan 5, se autocompleta el código con “-0000”. En el caso inverso, los últimos 4 dígitos se truncan.

Ejemplo:

```

<?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>
<fx:XML id="myData">
<root>
<contacts>
<item name="John Doe" zipcode="953763233"/>
</contacts>
</root>
</fx:XML>
<mx:ZipCodeFormatter id="fmtZip" formatString="#####-####"/>
</fx:Declarations>
<s:Label text="Zip Code
{fmtZip.format(myData.contacts.item.@zipcode) }"/>
</s:Application>

```

Salida:

Zip Code	95376-3233
----------	------------

Los errores de ZipCodeFormatter son:

- InvalidValue: pasa cuando la entrada no contiene la misma cantidad de caracteres que los esperados. Tener en cuenta que para los códigos americanos sólo se aceptan números y para el canadiense debe ser siempre 6 caracteres.
- InvalidFormat: ocurrió un error con formatString. Esto ocurre por caracteres inválidos o por haberse especificado un formato no reconocido.

SwitchSymbolFormatter: Es un dispositivo genérico para la presentación de datos que no es reconocido netamente como un Formatter predefinido de Flex. Es distinto del resto de los Formatter porque no tiene un contexto tan específico para el cual ser usado.

El SwitchSymbolFormatter puede usarse para crear formatos propios que extiendan de éste.

Funciones de este Formatter:

<<Tabla de Funciones>>

He aquí un ejemplo de su funcionamiento (Tener en cuenta que no existe una versión de SwitchSymbolFormatter en MXML, se usa la versión de ActionScript de modo ilustrativo):

```
<?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.formatters.SwitchSymbolFormatter;
      public var fmtSymbol:SwitchSymbolFormatter
      = new SwitchSymbolFormatter("#");
      public function formatMe(rawData:String):String
      {
        return fmtSymbol.formatValue("####-####",rawData);
      }
    ]]>
  </fx:Script>
  <fx:Declarations>
    <fx:XML id="myData">
      <root>
        <workorders>
          <item name="Fix something" id="99818382" />
        </workorders>
      </root>
    </fx:XML>
  </fx:Declarations>
  <s:Label verticalCenter="0" horizontalCenter="0"
    text="Work Order: {formatMe(myData.workorders.item.@id)}"/>
</s:Application>
```

Al ser este un Formatter diseñado para utilizarlo como padre, no existen para el mismo mensajes de error específicos.

Ahora veremos las diferentes formas de utilizar los Formatter's vistos:

7.2 Real-Time Formatting (Formateo en tiempo real)

No es el término oficial para llamar a este tipo de formato, pero se acerca al que se muestra en los ejemplos. (Excepto para el SwitchSymbolFormatter) para los cuales se ha llamado a la función Format() en el MXML.

Ejemplo de PhoneFormatter en tiempo real:

```
<mx:PhoneFormatter id="fmtPhone" formatString="###-###-####"/>
<s:Label text="Contact Phone: {fmtNumber.format("1112223333")}" />
```

No es algo muy elegante, pero es fácil de implementar y efectivo.

Como defecto principal se puede decir que usando los Formatters de esta forma no se puede involucrar lógica de negocio. Si queremos involucrar lógica debemos usar Scripted Formatting (Formateo encriptado o por script).

Scripted Formatting: (Formateo por script o dinámico)

Cuando hablamos de este tipo de formateo debemos involucrar ActionScript para procesar los datos de entrada de una forma más elaborada. Aunque hay algunas variantes para este tipo de formateo, todas implican utilizar una función al menos.

Usando una Función con un componente Formatter:

En este caso pasamos un valor a formatear a la función y recibimos el resultado formateado. Esta función utiliza un MXML Formatter que había sido creado previamente.

- ✓ Se generan funciones reusables.
- ✓ Son fáciles de usar.

Ejemplo:

```
<?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[
[Bindable]
public var rawPhone:String = "2223333";
public function formatThis(plainText:String):String
{
//We can add some extra business logic if we want.
//For example, changing the formatString on the fly
//depending on the size of the text.
if(plainText.length == 7)
fmtPhone.formatString = "###-####";
```

```

else
fmtPhone.formatString = "###-###-####";
return (fmtPhone.format (plainText));
}
]]>
</fx:Script>
<fx:Declarations>
<mx:PhoneFormatter id="fmtPhone"/>
</fx:Declarations>
<s:Label text="{formatThis (rawPhone)}"/>
</s:Application>

```

La desventaja que presenta este modo de operar es que el correcto funcionamiento de la Función dependerá de tener el elemento Formatter adecuadamente declarado. Una alternativa puede ser usar una instancia de un Formatter de ActionScript según se necesite.

Usando una Función con una clase Formatter:

Cada Formatter tiene una clase de ActionScript equivalente. Esto te da la opción de no tener que utilizar un componente MXML para hacer funcionar la función (valga la redundancia). De esta forma creamos una función más independiente y autosustentable.

Ejemplo:

```

<?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.formatters.*
[Bindable]
public var rawPhone:String = "2223333";
public function formatThis (plainText:String):String
{
var fmtPhone:PhoneFormatter = new PhoneFormatter();
if(plainText.length == 7)
fmtPhone.formatString = "###-####";
else
fmtPhone.formatString = "###-###-####";
return (fmtPhone.format (plainText));
}
]}>
</fx:Script>
<s:Label text="{formatThis (rawPhone)}"/>
</s:Application>

```

En el anterior ejemplo se creaba una instancia del Formatter cada vez que se llamaba a la función. Esto puede generar procesamiento de mas.

Para evitarlo se puede declarar un Formatter fuera de la función de la siguiente forma:

```

import mx.formatters.*
public var fmtPhone:PhoneFormatter = new PhoneFormatter();

```

Con esto la desventaja es que se va a crear una instancia de un Formatter aunque no se use.

Por último, nos queda ver el manejo de errores de formateo:

Para esto nos beneficiamos con el uso de ActionScript. Dentro de un script se puede identificar los posibles errores que puedan suceder y actuar en consecuencia. Notar

que en caso de ocurrir un error, el retorno va a ser un String vacío (es decir, se pierde el valor ingresado).

Ejemplo:

```
<?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.formatters.*
import mx.controls.Alert;
public function formatThis(plainText:String):String
{
var fmtPhone:PhoneFormatter = new PhoneFormatter();
var formattedString = fmtPhone.format(plainText);
if(fmtPhone.error == "Invalid value")
{
Alert.show('The value you entered is invalid.');
```

Obviamente es recomendable el controlar errores después de que un formateo se realiza.

Listing 5.5 Una colección de componentes basados en Botones

```
<?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.controls.Alert;
import mx.collections.ArrayCollection;
[Bindable]
public var myArray:ArrayCollection = new
ArrayCollection(['One', 'Two', 'Three']);
public function showMsg(msg:String):void
{
Alert.show('You just clicked on ' + msg);
}
]]>
</fx:Script>
<s:Panel title="Profile" width="360" height="240"
horizontalCenter="0"
verticalCenter="0">
<s:layout>
<s:HorizontalLayout/>
</s:layout>
<s:VGroup>
<s:Button id="thisBtn" label="Button" click="showMsg('button')"/>
<mx:LinkButton id="thisLinkBtn" label="LinkButton"
click="showMsg('linkbutton')"/>
</s:VGroup>
<s:VGroup>
<s:ButtonBar id="thisBtnBar"
dataProvider="{myArray}"
click="showMsg(ButtonBar(event.currentTarget).selectedItem)" />
<mx:LinkBar id="thisLinkBar"
```

```
dataProvider="{myArray}"
itemClick="showMsg(event.label)" />
<mx:ToggleBarButton id="thisToggleBar"
dataProvider="{myArray}"
itemClick="showMsg(event.label)" />
</s:VGroup>
</s:Panel>
</s:Application>
```

Al ser un lenguaje OOP, Flex tiende a acumularse sobre sí mismo, el uso de algunos componentes como fundación.

Barra de botones, barra de enlaces y ToggleBarButton (BarButton, LinkBar, and ToggleBarButton) son una familia tan cerca desde un punto de vista mecánico, al margen de las diferencias de presentación menores, son la misma cosa.



Usan dataProvider, que es una propiedad que acepta datos en matrices como para generar su pantalla.

También se puede sacar ventaja de ArrayCollection. En el listado 5.5 que utilizó ActionScript para crear este ArrayCollection, lo que le permite luego enlazar los resultados a los distintos controles (tenga en cuenta que una serie regular no admite el enlace). La ventaja de crear una variable de ArrayCollection es la capacidad de desacoplar el lógica que contiene a partir del componente que lo muestra

NOTA Cuál es la diferencia entre el clic e itemClick? No mucho; itemClick añade propiedades de la etiqueta y el índice para que sea conveniente saber en cual botón se hizo clic. Click refiere al botón del ratón que se está presionando (el gatillo) en un componente botón. ItemClick es específicamente qué botón (si está utilizando un componente que tiene más de uno). El itemClick sólo es compatible con los controles de Halo, mientras que el Spark ToggleBarButton utiliza el evento click genérico, por lo que tiene que utilizar la propiedad currentTarget para acceder al botón que se presionó.

Los ejemplos anteriores cubren todos los botones, excepto dos: el PopUpButton y el PopUpMenuButton

EL POPUPBUTTON Y POPUPMENUBUTTON

Son considerados casos especiales, y, como tal, que no necesitan una explicación más detallada.

Ejemplo de como usar PopUpMenuButton

```
<?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.controls.Alert;
public function showMsg(msg:String):void
{
Alert.show('You just clicked on ' + msg);
}
]]>
</fx:Script>

<mx:PopUpMenuButton id="menuBtn"
dataProvider="{['One', 'Two', 'Three']}"
click="showMsg('left side')"
itemClick="showMsg('right side with ' + event.label)"/>
</s:Application>

```



Aca se usan los 2 eventos, click y itemClick . click a la derecha despliega el menu cuando seleccionas, se dispara el trigger de itemClick pero si clickeo el lado izquierdo dispara ambos itemClick y click, uno después del otro.

Esto es útil cuando es necesario diferenciar entre estas dos interacciones del usuario.

Por ejemplo, supongamos que el menú desplegable definida en el listado 5.6 contiene una lista de marcas de tarjetas de crédito (por ejemplo, Visa, MasterCard y American Express). Cuando el usuario cambia la selección (itemClick), el tipo de tarjeta de crédito se almacena en una variable, pero cuando se pulsa el botón principal (click), Flex envía el formulario.

El PopUp-Button puede realizar las mismas tareas que PopUpMenuButton pero posee capacidades más amplias, que naturalmente, lleva más código para implementar.

PopUpMenuButton utiliza un elemento llamado menú para generar el menú desplegable (otro objeto estándar con Flex). Pero Menu es el único que puede PopUpMenuButton mostrar.

PopUpButton puede mostrar una mayor variedad de elementos, pero por defecto no cualquiera en particular .

Listing 5.7 Usando PopUpButton para mostrar un menu opcional.

```

<?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.controls.Alert;
import mx.events.*;
import mx.controls.Menu;
public var menuItems:Object =
[{'label':'One'}, {'label':'Two'}, {'label':'Three'}];
public var thisMenu:Menu = Menu.createMenu(null, menuItems, false);
public function handleItemClick(event:MenuEvent):void
{
menuBtn.label = event.label;
}
]]>
</fx:Script>
</s:Application>

```

```

]]>
</fx:Script>
<mx:PopUpButton id="menuBtn"
creationComplete=
"thisMenu.addEventListener('itemClick',handleItemClick);"
popUp="{thisMenu}"/>
</s:Application>

```

Aquí hay tres puntos a tener en cuenta sobre listing 5.7:

- En la mayoría de los ejemplos, la variable que contiene la información a ser poblada es una matriz. En el ejemplo anterior, se utilizó un objeto. Si usted recuerda de capítulo 3, no hay mucha diferencia entre ellos. En este contexto se podía haber declarado MenuItems como matriz en vez de objeto y habría funcionado.
- Introdujimos creationComplete. Este evento indica al PopUpButton para que llame código adicional cuando se termina su creación.
- Este código adicional le dice a nuestro objeto Menu (thisMenu) para escuchar el evento itemClick, y si eso ocurre, llamar a nuestro gestor de eventos (handleItemClick) para gestionarla.

La principal diferencia entre el uso PopUpButton y PopUpMenuButton es que con PopUpButton usted no está limitado a su uso de los menús, se puede utilizar cualquier elemento visual para interactuar con el usuario. Si lo que quieres es el efecto del menú desplegable, ir con el PopUpMenuButton. Si necesita más control sobre las opciones que desee presentar para el usuario, aprovechar la flexibilidad de PopUpButton.

5.2.1 Controles Lista de Selección

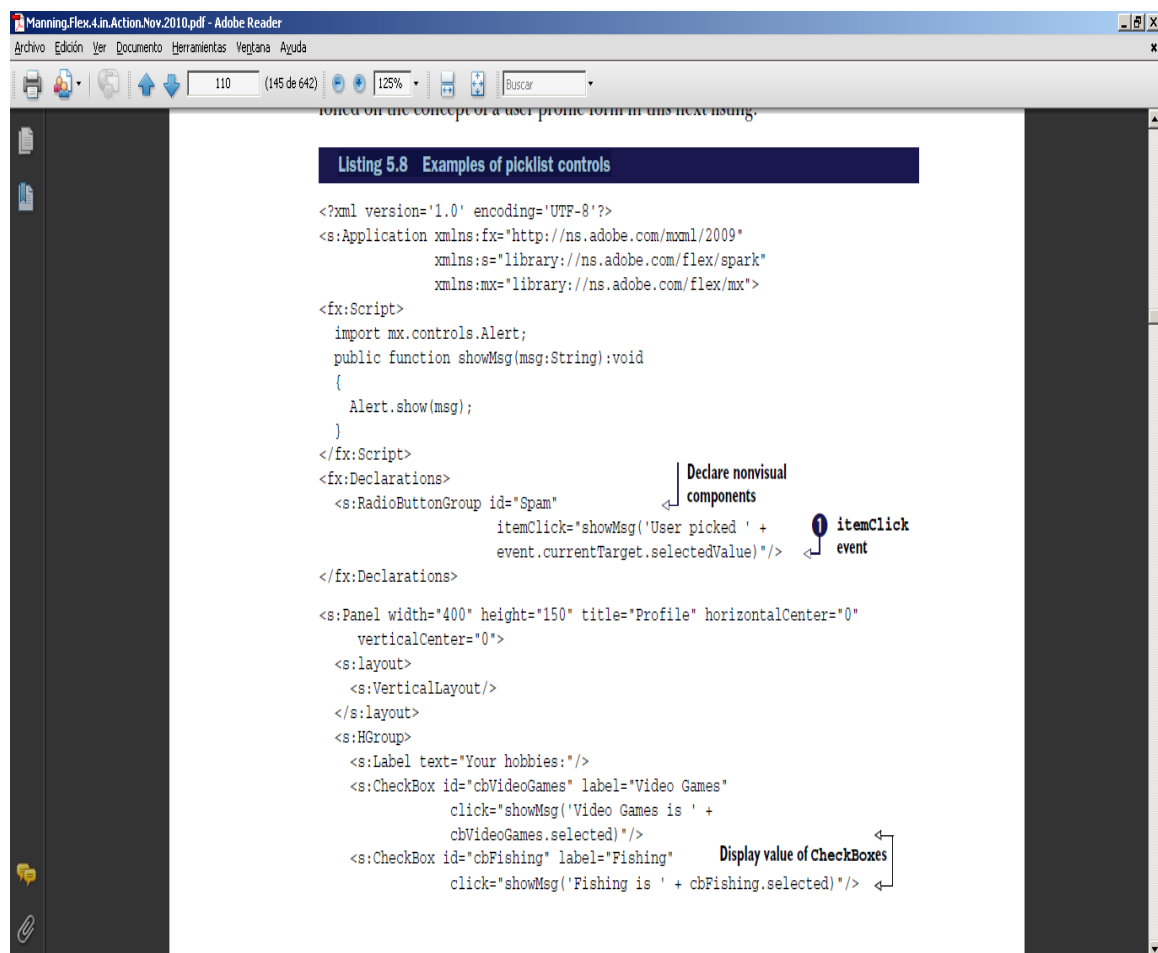
Listas de selección son los controles que representan todo lo demás que, presentan listas de opciones, se muestran en la siguiente tabla 5.5

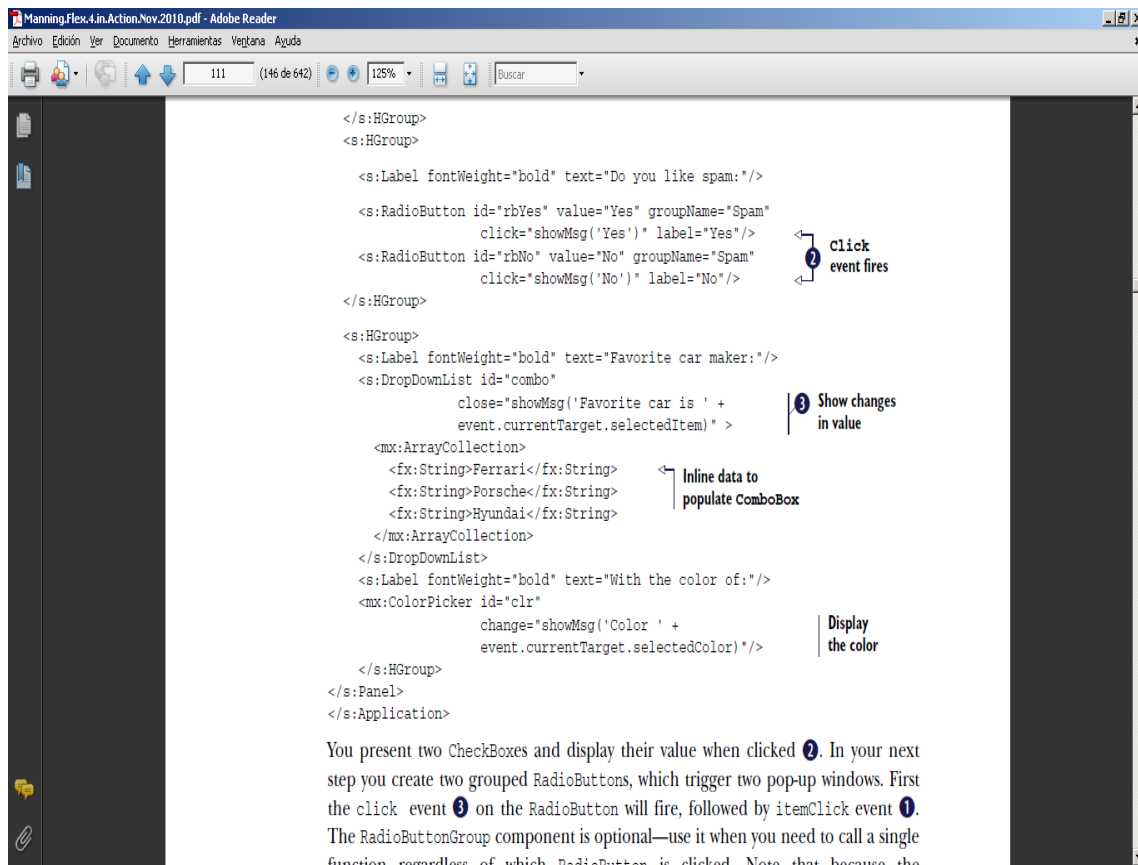
Control	Paquete	Descripción
CheckBox	Spark	El control CheckBox genérico (idéntico comportamiento a un HTML CheckBox). Es un descendiente de los botones, excepto que está diseñado específicamente con el fin de ser activada y desactivada. El uso más común de un CheckBox es ofrecer una serie de opciones desde el que el usuario puede comprobar (seleccionar) una o más de estas opciones.
RadioButton	Spark	Similar a la casilla de verificación, excepto cuando se agrupan como una gama de opciones, el usuario puede seleccionar sólo una. El RadioButton también un descendiente de Button.
DropDownListSpark		Este control del menú desplegable muestra la opción seleccionada en ese momento, pero cuando se hace clic se deja caer una lista de visualización de opciones a elegir. ComboBox es lo mismo que un cuadro de <select> en HTML, excepto a diferencia de su equivalente HTML, puede permitir que el usuario de forma libre editar el texto seleccionado. A pesar de que parece basarse en Botón, no es un descendiente. Anteriormente conocido como el ComboBox en el

paquete de Halo, excepto que no es compatible con capacidad de edición en línea.

ColorPicker **Halo** Un hermano de ComboBox. Se despliega un menú de color, lo que permite al usuario seleccionar entre una paleta de colores. Usted puede usar esto para permitir a los usuarios personalizar su escritorio o como parte del flujo de trabajo de la aplicación (por ejemplo, un selector de color con el que los visitantes del sitio web de un fabricante de automóviles puede ver las opciones de color de un vehículo en tiempo real).

Listing 5.8 con un ejemplo de programa con lo mas nuevo, en un concepto de fomulario de perfil de usuario en la siguiente lista.

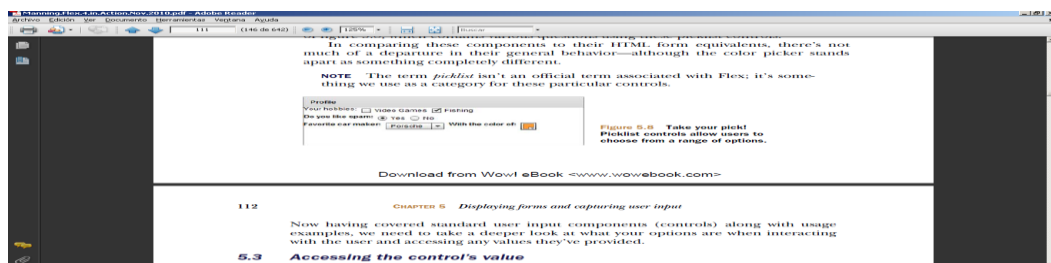




Click en 2 presenta dos checkbox y muestra el valor. En el próximo paso crearas dos grupos RadioButtons que disparan dos ventanas pop-up. Primero, el click en evento 3 del RadioButton se disparará, seguido del itemClick, evento 1. El RadioButtonGroup, es opcional, usarlo cuando tenes que llamar una unica funcion independientemente del RadioButton clickeado.

Tener en cuenta que debido a que la etiqueta `<s:RadioButtonGroup/>` no es visual necesitan ser agrupadas dentro de un `<fx:Declarations/>`

NOTA: el termino picklist no es oficial, se usa asociado como categoría para este control en particular.



5.3 Accediendo a valores del control

Tenga en cuenta que como en todo lenguaje, hay muchas formas de implementar lo mismo. Los ejemplos anteriores son fáciles de hacerlo, pero pueden haber mejores formas.

5.3.1 Pasando valores a una función

Aunque nuestra intención era mantener a los ejemplos ágiles y fáciles de entender, pasar valores a las funciones independientes es una buena técnica a emplear, ya que alivia la lógica que rocea los datos al no tener necesidad de saber de dónde provienen. Permite reutilización. El cambio de un `RadioButton` a una `CheckBox` no perturba la función del todo.

Esto es lo que ha observado con la función `ShowMsg`:

```
public function showMsg(msg:String):void
{
    Alert.show(msg);
}
```

Cuando llamo una función es en respuesta a un disparador que causó un evento. Cada evento tiene un objeto del evento para que viaje con el, por medio de la propiedad `CurrentTarget` que tiene todo tipo de artículos, incluido el objeto que viaja con el.

En el ejemplo extraigo el valor seleccionado

```
<s:RadioButtonGroup itemClick="showMsg(event.currentTarget.selectedValue)"/>
```

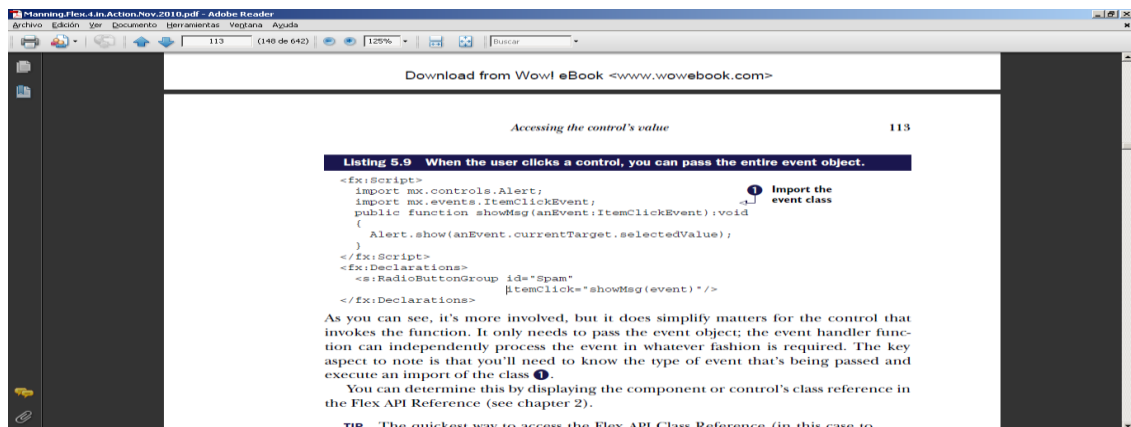
Pero puedo asignar un id

```
<s:RadioButtonGroup id="Spam"
itemClick="showMsg(Spam.selectedValue.toString())"/>
```

Flex aprovecha el evento en sí.

5.3.2 Pasando eventos a una función

Más responsabilidad a la función, aumento la conciencia del origen de los datos, paso todo el objeto evento como en el siguiente listado 5.9



Uso el manejador de eventos, lo importante es saber el tipo del evento que estoy pasando y hacer import de esa clase. (Flex API Class Referente) , clickear el control y presionar Shift+F2.

En el listing 5.9 sería:

En su código, haga clic en `RadioButtonGroup` y pulse Shift + F2. La referencia API

Para este componente se mostrará.

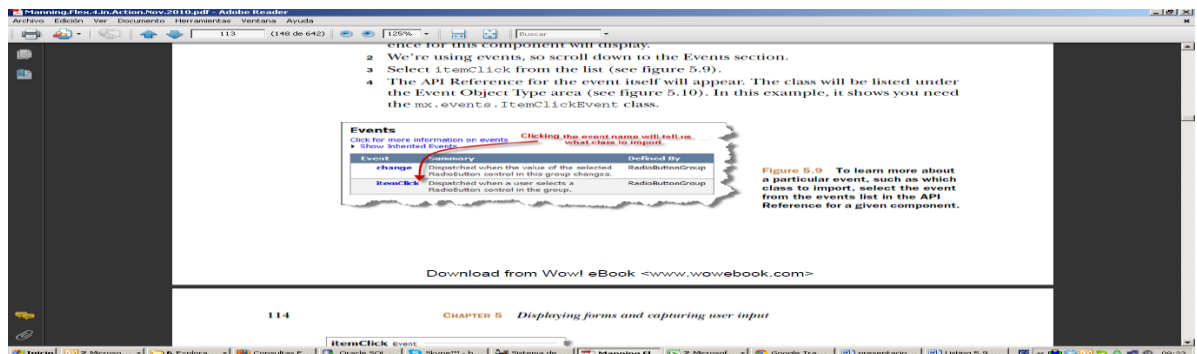
2 Estamos utilizando eventos, así que baje hasta la sección de Eventos.

3 Seleccione itemClick de la lista (véase el gráfico 5.9).

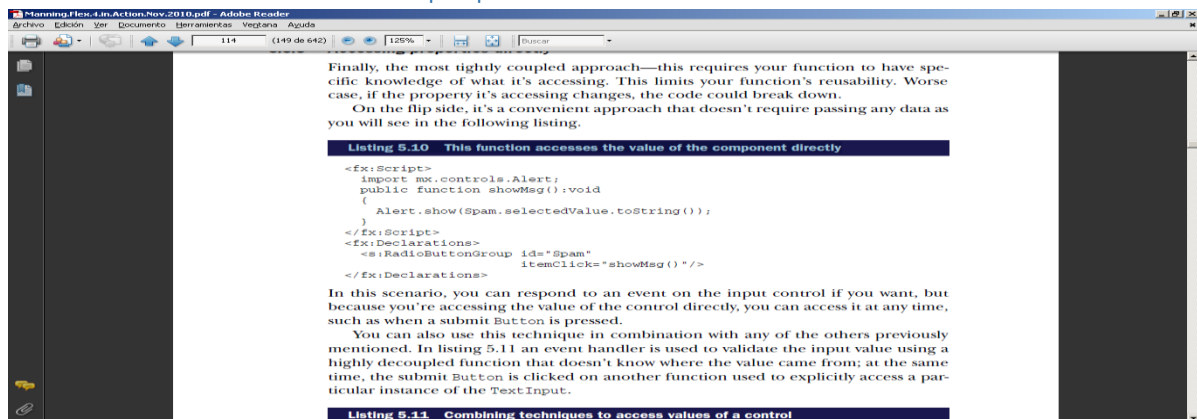
4 Aparecerá la referencia de API para el evento en sí. La clase aparecerá en la lista

Caso área Tipo de objeto (ver figura 5.10). En este ejemplo, se muestra lo que necesita

la clase `mx.events.ItemClickEvent`



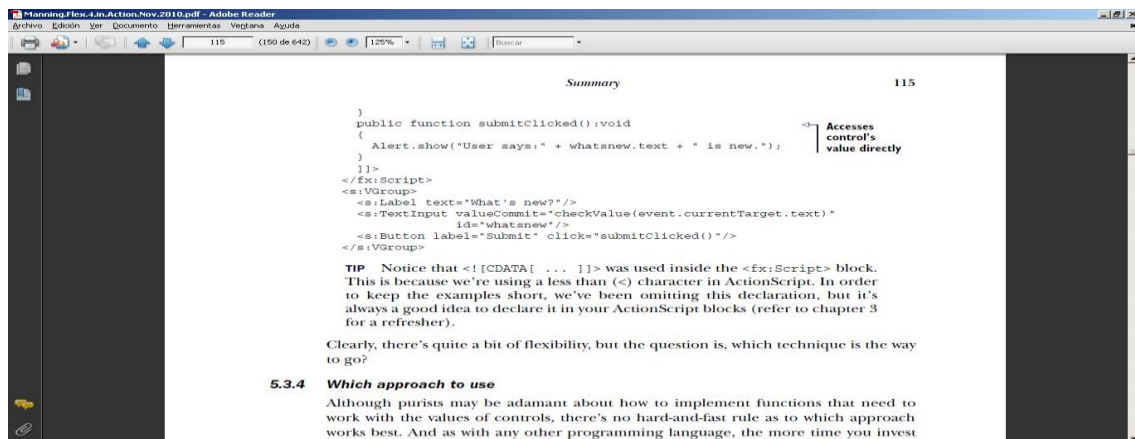
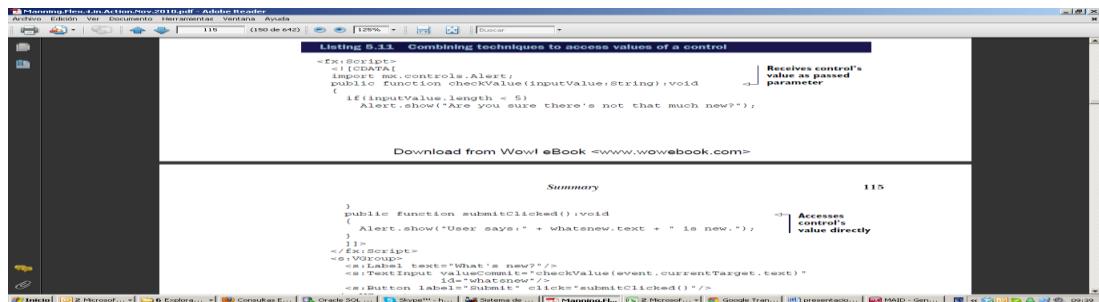
5.3.3 Acceder directamente a propiedades



Puedo responder a un evento en el control de entrada si lo deseo, pero como accedo al valor del control, puedo hacerlo cuando quiera, por ejemplo al presionar el submit Button.

Se puede combinar esto con otras técnicas antes mencionadas

En el listado 5.11 se usa el manejador de eventos para validar el valor del input usando una muy desacoplada función que no conoce de donde viene el valor.



5.3.4 Que enfoque utilizar

Los mas puristas pueden inclinarse por una u otra forma de trabajar con los valores de los controles. Invertir tiempo en que una función sea reutilizable tiene su beneficio a largo plazo. La realidad es que no hay una forma correcta o incorrecta, hay que evaluar tamaño de la aplicación, su esperanza de vida, y otros temas. Si tenemos por objetivo un proyecto o prueba de concepto rápido no se justifica invertir mucho tiempo.

5.4 Resumen

Flex viene con un número de componentes de la interfaz conocidas como controles. Algunos controles tienen equivalentes en HTML, y muchos van mucho más allá de sus equivalentes en HTML. Por ejemplo, el ColorPicker, sliders, y la gran variedad de botones ofrecen una gran libertad de ser creativo en la construcción de interfaces y formas. Tenga en cuenta que, a diferencia de HTML, donde un `<form/>` encapsula un conjunto de entradas de formulario, en Flex una forma no hace nada más que ayudar diseñar componentes. Aquellos componentes no necesariamente tienen que estar dentro de un recipiente `<s:Form/>`.

Hay un número de maneras que usted puede recuperar el valor de las selecciones del usuario, se puede hacerlo en tiempo real, mediante el aprovechamiento de los acontecimientos, o se puede acceder al valor según sea necesario utilizando la propiedad `id` del componente. Ahora tiene las herramientas para crear formas de capturar la entrada del usuario y agregar la interactividad para acceder a los valores de siempre.

Captura de entradas del usuario es una cosa, pero ¿qué pasa con validarlo? Por ejemplo, puede ser necesario para construir una aplicación que requiere que el usuario introduzca una contraseña formada por al menos cinco caracteres. En el próximo capítulo vamos a echar un vistazo a cómo Flex validadores pueden ser utilizado para llevar a cabo esto, y mucho más.

6. Validar la información del Usuario

Este capítulo trata de

- Validación contra la aplicación
- Traspaso validación
- Validación valor comprometido (Committed-value)
- Validación de secuencias de comandos en tiempo real

La realización de la validación del lado del cliente es una fortaleza clave de Flex. Desde un punto de usabilidad. En ambiente html, javascript resuelve algo de la validación.

Flex proporciona un mecanismo de validación en tiempo real que lleva a cabo discretamente ese objetivo. Maneja varios tipos de validaciones, como las que se verifique el formato correcto de los números de teléfono y de la estructura de las direcciones de correo electrónico.

6.1 Resumen de la validación

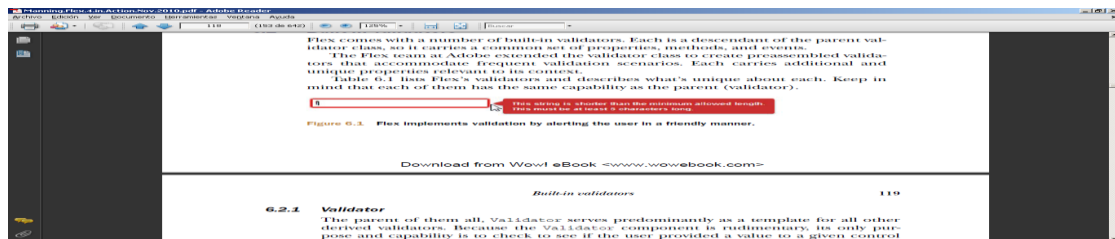
Flex permite validar fácil con componentes integrados para hacer el trabajo duro para usted y ofrece una serie de enfoques para invocarlos. La siguiente lista explica común formas de validar la entrada del usuario:

- **Validación en tiempo real:** todas las interacciones de teclas o el ratón, la aplicación comprueba si los datos de entrada son como se esperaban o requerían.
- **Validación del valor comprometido(committed-value):** similar al tiempo real, excepto que en lugar de evaluar cada golpe de teclado, la aplicación espera hasta que el usuario haya rellenado el campo completo y confirme la entrada (por ejemplo, pulsando Intro o pulsar Tab para cambiar los campos).
- **Traspaso de validación:** Por lo general, el resultado de golpear un botón de envío, ste tipo de validación pasa a través de todas las entradas de formulario para asegurarse de que todo está validado a la vez.
- **Scripted validation:** usando ActionScript puedes crear validadores dinámicamente, incluso reutilizar el mismo validador en múltiples elementos.

La validación es alertando sobre el usuario y el desarrollador que no se cumplen ciertos criterios. El desarrollador tiene que determinar la forma de aplicar la validación que solicita al usuario en cuanto a lo que hay que hacer para cumplir con los requisitos de la aplicación.

6.2 Validadores empotrados

Flex viene con un número de validadores incorporados. Cada uno es un descendiente del padre validador clase, por lo que lleva a un conjunto común de propiedades, métodos y eventos. El equipo Flex de Adobe amplió la clase de validador para crear validadores premontados, cada uno lleva propiedades relevantes a su contexto.



6.2 Validador

El validador padre sirve como template para los demás validadores derivados de este. Sirve para corroborar si el usuario le dio un valor a un control que es apuntado por el validador. Usando la api, se puede ver la cantidad de propiedades que el validador soporta y como es padre, todos los hijos las heredan.

Propiedad	Tipo	Descripción
enabled	Boolean	verdadero o falso. Permite alternar si la validación es activa
required	Boolean	verdadero o falso. ¿Es este un campo en el que el usuario es requerida para proporcionar la entrada?
requiredFieldErrorString	String	Si requerido es cierto, este es el mensaje que se muestra para el usuario. Si no se especifica, el valor predeterminado es un Mensaje general.
source	Object	El objeto (por ejemplo, un TextInput) que desea validación.
property	String	La propiedad (por ejemplo, la propiedad de texto del TextInput) del objeto de origen que se comprobará para validar si el valor proporcionado es compatible.
listener	Object	Por defecto, este es lo que la fuente se establece. cuando la propiedad de la fuente no valida, Flex destaca el objeto de origen, pero si quieres un objeto diferente resaltado, puedes utilizar el oyente.
valid	Function	The name of a function you want to call if validation passes.
invalid	Function	El nombre de una función que desea llamar en caso de validación falla.

trigger	Object	<p>Por defecto, este es lo que la fuente se establece. Es la</p> <p>nombre del objeto que hará que el disparador que se produzca</p> <p>(por ejemplo, una instancia de un botón que el usuario</p> <p>usaría para enviar un formulario).</p>
triggerEvent	String	<p>El nombre del evento que desea hacer la validación</p> <p>para ejecutar basado en el gatillo (por ejemplo, un Enviar</p> <p>evento de clic de botón). Por defecto se busca la</p> <p>caso valueCommit (que por lo general es causada por el</p> <p>usuario desplazarse a otro campo, presione la tecla Tab o</p> <p>haciendo clic en otros lugares).</p>