Manejos de textos

Web

Implementando formato de texto con CSS

Selectores CSS

ActionScript 3.0 soporta tres selectores:

- **Tipo de elemento**: la relación con el estilo está dada por el nombre de la etiqueta, es decir, con solo utilizar una etiqueta determinada se reflejará el estilo en el texto. Por ejemplo: `<yodaStyle>`.
- **Clase**: este tipo de selector permite relacionar un estilo al atributo `class` dentro de cualquier etiqueta. Por ejemplo: `<t class:yodaStyle>`.
- **Pseudo-clase de vínculo**: solamente para el caso de la etiqueta `<a>`. Permite asociar un estilo para cuando el mouse se posa sobre el vínculo, se hace clic o el mouse está fuera del vínculo. No está habilitado el estado `visited` (visitado).

Pasos para asignar un formato CSS

En líneas generales, los pasos para aplicar un formato CSS son los siguientes:

Creación

1. Creamos un objeto `StyleSheet`.
2. Opcionalmente, creamos un objeto del tipo `Object` con ciertas propiedades de estilo determinados. Podríamos o bien pasar un objeto directamente al método `setStyle`, o bien pasar un `String` al método: `parseCSS`.
3. Creamos el objeto `TextField`.

Asignación

4. Mediante el método: `setStyle`, se añade un estilo. Este método admite dos parámetros, el nombre del estilo y un objeto con las propiedades del mismo. Si el estilo no existe se añadirá, si existe se sustituirá, si el parámetro del objeto es `null` se eliminará.
5. Se asigna la instancia de `StyleSheet` a la propiedad `styleSheet` del `TextField`.
6. Se asigna un texto HTML al `TextField`.

Para que el cambio de estilo tenga efecto, primero hay que asignar `styleSheet` del `TextField`; luego, asignar el texto.

Implementemos ejemplos:

Tipo de elemento como selector

Si queremos asociar un estilo determinado a una etiqueta, es decir, que cada vez que se utilice esa etiqueta se represente un mismo estilo, entonces tendríamos que

implementar algo como lo que se muestra a continuación:

```
package
{
  import flash.display.Sprite;
  import flash.text.StyleSheet;
  import flash.text.TextField;
  import flash.text.TextFieldAutoSize;

  public class Main extends Sprite
  {
    public function Main():void
    {
      initialize();
    }

    private function initialize():void
    {
      //Instanciamos el TextField
      var textField:TextField = new TextField();
      //Hacemos ajustes al TextField.
      textField.multiline = true;
      textField.wordWrap = false;
      textField.autoSize = TextFieldAutoSize.LEFT;

      //Creamos un objeto del tipo Object y le asignamos algunas de
las
      //propiedades.
      var yodaStyle:Object = new Object();
      yodaStyle.fontFamily = "Tahoma";
      yodaStyle.fontSize = 20;
      yodaStyle.fontWeight = "bold";
      yodaStyle.color = "#00CC00";

      //Creamos un objeto del tipo StyleSheet.
      var stylesSheet:StyleSheet = new StyleSheet();
```

```
      //Agregamos la regla. En este caso asociamos una etiqueta con
un estilo.
      stylesSheet.setStyle("p", yodaStyle);

      //Aplicamos el conjunto de estilos al TextField.
      textField.styleSheet = stylesSheet;

      //Asignamos un texto HTML
      textField.htmlText = "<p>Hazlo, o no lo hagas, pero no lo in-
tentes.</p>"

      //Le asignamos una posición:
      textField.x = 100;
      textField.y = 100;

      //Lo agregamos a display list
      addChild(textField);
    }
  }
}
```

El resultado que obtendríamos es el siguiente:

Fig. 1 Resultado de aplicar el formato CSS.

Para este `TextField`, todo texto contenido en la etiqueta `<p>` será representado según el estilo que se haya definido en la regla. Esto permite además generar etiquetas personalizadas. También podemos pasar el objeto de la forma abreviada:

```
//Instanciamos el TextField
var textField:TextField = new TextField();
//Hacemos ajustes al TextField.
textField.multiline = true;
textField.wordWrap = false;
textField.autoSize = TextFieldAutoSize.LEFT;

//Creamos un objeto del tipo StyleSheet.
var stylesSheet:StyleSheet = new StyleSheet();

//Creamos un objeto del tipo StyleSheet.
var stylesSheet:StyleSheet = new StyleSheet();

//Asociamos el tipo de elemento al Object en un solo paso.
stylesSheet.setStyle("p", {
                            fontFamily:"Tahoma",
                            fontSize:20,
                            fontWeight:"bold",
                            color:"#00CC00"
                          });

//Aplicamos el conjunto de estilos al TextField.
textField.styleSheet = stylesSheet;

//Asignamos un texto HTML
textField.htmlText = "<p>Hazlo, o no lo hagas, pero no lo inten-
tes.</p>"
```

Etiqueta personalizada

Podríamos también hacer lo que se muestra a continuación, donde el selector y la etiqueta están resaltados en negrita.

```
//Asociamos el tipo personalizado al Object en un solo paso.
stylesSheet.setStyle("yodaStyle", {
                            fontFamily:"Tahoma",
                            fontSize:20,
                            fontWeight:"bold",
                            color:"#00CC00"
                           });

//Aplicamos el conjunto de estilos al TextField.
textField.styleSheet = stylesSheet;

//Asignamos un texto HTML
textField.htmlText = "<yodaStyle>Hazlo, o no lo hagas, pero no lo
intentes.</yodaStyle>"
```

Clase como selector

En ActionScript 3.0, existe otra variante para asignar un estilo independientemente de la etiqueta, que permite aplicar diferentes estilos a una misma etiqueta. Para esto, se define una clase como selector, se le asigna el estilo en el objeto `StyleSheet` y se lo invoca en la etiqueta mediante el atributo `class`:

```
//Creamos un objeto del tipo StyleSheet.

var stylesSheet:StyleSheet = new StyleSheet();

//Asociamos una clase al Object en un solo paso.
stylesSheet.setStyle(".yodaStyle", {
                                  fontFamily:"Tahoma",
                                  fontSize:20,
                                  fontWeight:"bold",
                                  color:"#00CC00"
                                 });
```

```
//Aplicamos el conjunto de estilos al TextField.
textField.styleSheet = stylesSheet;


//Asignamos un texto HTML
textField.htmlText = "<p class=yodastyle>Hazlo, o no lo hagas, pero
no lo intentes. </yodaStyle>"
```

Pseudoclase de vínculo como selector

El único caso en ActionScript 3.0 en donde se puede utilizar la pseudoclase es con la etiqueta <a>. Esto permite asignar estilo a tres estados del vínculo:

- **a**: en reposo.
- **a:hover**: cuando se coloca el mouse sobre el vínculo.
- **a:activate**: cuando se hace clic sobre el vínculo.

Implementamos este caso sobre el ejemplo que estamos trabajando:

Aplicando el estilo CSS desde un string

Como habíamos mencionado, se puede asignar un estilo desde un texto.

Se hace del siguiente modo:

```
//Creamos un objeto del tipo StyleSheet.
var stylesSheet:StyleSheet = new StyleSheet();


//Asociamos el tipo de elemento al Object en un solo paso.
var stringStyle:String = 'p {font-family: Arial; font-size:20px;
font-weight:bold; color: #00CC00}';


//Asociamos el String al selector
stylesSheet.parseCSS(stringStyle);
```

```
//Aplicamos el conjunto de estilos al TextField.
textField.styleSheet = stylesSheet;


//Asignamos un texto HTML
textField.htmlText = "<p>Hazlo, o no lo hagas, pero no lo inten-
tes.</p>"
```

La posibilidad de asignar un estilo desde un texto plano nos permitiría cargar un archivo plano, y luego desde los datos cargados, asignar el estilo. A través de esta operación se pueden hacer variaciones en nuestra aplicación sin necesidad de volver a compilar, solo sería necesario hacer los cambios en el archivo que sería cargado.

Qué restricciones suceden al aplicar un estilo CSS a un TextField y cómo evitarlas

Cuando se asigna la propiedad `styleSheet` de un `TextField`, no se podrán ejecutar los siguientes métodos:

- `replaceText()`
- `replaceSelectedText()`
- `defaultTextFormat`
- `setTextFormat()`

Es más, si utilizamos estos métodos luego de asignar `styleSheet`, obtendremos el siguiente error:

```
Error #2009: No se puede utilizar este método en un campo
de texto con una hoja de estilos.
```

Y a estas restricciones se suma que si tenemos un texto de entrada, luego de ser asignado `styleSheet`, el usuario no podrá seguir escribiendo en ese campo de texto.

Pero esto no debe desanimarnos en el uso de esta potente funcionalidad. Hay una alternativa, detallada a continuación, para evitar estas restricciones:

- Asignamos la propiedad: `styleSheet`.
- Asignamos el texto al `TextField`.

Y aquí viene el truco:

- Asignamos `styleSheet` con el valor: `null`.

Entonces, el formato permanece aplicado y desaparecen las restricciones.

```
//Aplicamos el conjunto de estilos al TextField.
textField.styleSheet = stylesSheet;

//Asignamos un texto HTML
textField.htmlText = "<p>Hazlo, o no lo hagas, pero no lo inten-
tes.</p>"

//para poder anular la restricción, asignamos null al styleSheet
textField.styleSheet = null;

//Agregamos algún texto.
textField.appenChild(" Maestro Yoda.");
```

Manejo de texto con Flash Text Engine

Uno de los aspectos más criticados de AS 3.0 es la limitada capacidad de manejo de texto. A partir de la versión de 10 del reproductor de Flash, se agregó un nuevo motor: Flash Text Engine, que permite un control más extenso y preciso sobre el manejo de textos, y es una mejora sobre la clase `TextField`. En el ambiente de desarrollo de Flash, ahora existen dos tipos de textos: el clásico, basado en la clase `TextField`; y el TLF, basado en un *framework* escrito en ActionScript 3.0.

El TLF (*Text Layout Framework*) funciona como un envoltorio del Flash Text Engine que simplifica y facilita su uso llamado. El Flash utiliza por defecto el TLF. A la clase `TextField` hay que conocerla para manejar así proyectos que se vienen realizando previamente a la versión 10 del reproductor. Será una cuestión de tiempo para que la clase `TextField` caiga en desuso, aunque es aún una opción válida. El Flash Text Engine es a los textos lo que las nuevas clases descendientes del DisplayObject son a los gráficos.

Tanto el Flash Text Engine como Text Layout Framework están disponibles únicamente a partir del Flash Player 10 y del Adobe AIR 1.5.

Esta nueva funcionalidad provee de amplias posibilidades esperadas durante mucho tiempo en el manejo de textos como:

- Dirección de texto de derecha a izquierda para idiomas como el árabe o el hebreo, y la combinación de textos en ambas direcciones. Se puede combinar texto en castellano y en hebreo en un mismo párrafo.
- Dirección de texto vertical y soporte de caracteres asiáticos para idiomas como el chino o el japonés.
- Distribución del texto en columnas o hacer fluir el texto entre distintos contenedores.
- Amplio control de tipografías.

Aspectos generales del Flash Text Engine

El FTE (*Flash Text Engine*) es un conjunto de clases que permite manejar contenido visual: texto y gráficos. La unidad de contenido según el FTE es la línea de texto. A continuación detallaremos las entidades que entran en juego a muy alto nivel, y luego las relacionaremos con las clases de ActionScript 3.0.

Entidades intervinientes

- **Unidades**: entidades que representan las unidades del FTE.
- **Contenido**: puede estar dado por un texto o un gráfico. Esta entidad, por lo tanto, es la clase base para los elementos de texto y gráficos. Es del estilo de clase abstracta.
- **Elemento texto:** desciende de la entidad contenido, y representa un texto con un formato determinado.
- **Elemento gráfico:** representa los gráficos, y por eso, desciende de la entidad contenido.
- **Grupo de elementos:** se trata de un `Vector` de entidades del tipo contenido, y también desciende de esta entidad. En ese sentido, trata a un grupo de contenido como si fuera una unidad de Elemento.
- **Formato:** contiene los datos necesarios para dar formato al contenido.
- **Manejadores:** son entidades responsables del manejo de otras entidades que contienen.
- **Bloque de texto:** es un contenedor y manejador de la línea de texto. Se encarga de hacer fluir, de acuerdo con las propiedades que le son configuradas, el contenido entre las líneas de texto. Es el orquestador de como se representa el contenido. El bloque de texto toma el contenido, su formato (fuente, color, forma de representar la fuente), y luego se encarga de distribuirlo en líneas de texto.
- **Línea de texto:** extiende a un `DisplayObjectContainer`, y es una clase que no puede instanciarse directamente, sino que se crea a través del bloque de texto. Esta clase "sabe" como armar estructuras de elementos, representarlos visualmente y manejar los eventos.

Veámoslo en un gráfico.

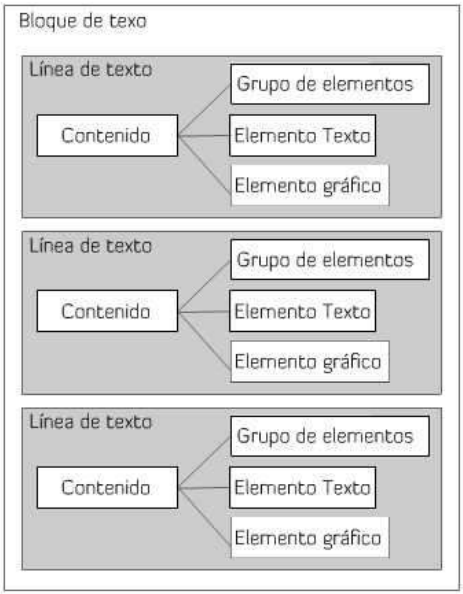


Fig. 2 Esquema de la estructura de entidades.

Ahora, hagamos el paralelo con las clases de ActionScript 3.0.

Contenido → `ContentElement`
Elemento Texto → `TextElement`
Elemento gráfico → `GraphicElement`
Grupo de elementos → `GroupElement`
Formato → `ElementFormat`
Bloque de texto → `TextBlock`
Línea de texto → `TextLine`

Implementando con Text Framework Engine

Y ahora, veamos cómo implementamos un código sencillo.

Para eso, debemos configurar `MainTFE` como clase del documento:

```
package
{
  import flash.display.Sprite;
  import flash.text.engine.TextElement;
  import flash.text.engine.TextBlock;
  import flash.text.engine.TextLine;
  import flash.text.engine.ElementFormat;
  import flash.text.engine.FontDescription;

  public class MainTFE extends Sprite
  {
    public function MainTFE()
    {
      //Configuramos el contenido:
      var textContent:String = "";
      textContent += "Que es mi barco mi tesoro,\n";
      textContent += "que es mi dios la libertad,\n";
      textContent += "mi ley, la fuerza y el viento,\n";
      textContent += "mi única patria, la mar.";

      //Creamos el formato para aplicar al contenido.
      var fontDescription:FontDescription = new FontDescrip-
tion("Papyrus");
      var elementFormat:ElementFormat = new ElementFor-
mat(fontDescription, 30);

      //Creamos el Elemento texto.
      var content:TextElement = new TextElement(textContent);

      //Le asignamos el formato.
      content.elementFormat = elementFormat;

     //Creamos el bloque de texto.
     var textBlock:TextBlock = new TextBlock();
```

```
      //Le asignamos el contenido.
      textBlock.content = content;


      //A través del TextBlock creamos y distribuimos los TextLine.
      //El TextBlock creará líneas en la medida que quede contenido
por mostrar.
      var lastLine:TextLine = null;
      var positionY:Number = 50;
      var index:Number = 0;
      var leading:Number = 5;
      var textBlockWidth:Number = 350;
      while (lastLine = textBlock.createTextLine(lastLine, text-
BlockWidth))
      {
         lastLine.y = positionY;
         positionY += lastLine.textHeight + leading;
         addChild(lastLine);
       }
     }
   }
}
```

Y veremos el siguiente resultado:

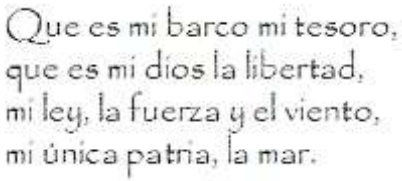


Fig. 3

A continuación, analizaremos cómo utilizando Text Layout Format se nos simplifica y reduce la cantidad de código necesario para lograr el mismo resultado.

Utilización de TLF (Text Layout Framework)

Como habíamos visto previamente, el TLF viene a simplificarnos el uso de del TFE. Las entidades serán las mismas. Uno de los aspectos que se simplifican es el despliegue de las líneas de texto. La lógica para desplegarlas ya no dependerá del programador. Tendremos a las entidades:

- **Contenido**: son los textos o contenedores de textos. También conocido como historia.
- **Formato**: el aspecto y estilo que tendrá la historia y su flujo.
- **Flujo**: todo lo relacionado con la forma en que el texto fluirá entre los contenedores.

Contenido

Las historias se estructuran de una manera similar a la que se utiliza en HTML.

Esto significa que tenemos una estructura de árbol y la raíz de una historia es siempre un `TextFlow`. Dentro de él pueden existir dos tipos de entidades: `FlowGroupElement` (que pueden contener otros elementos), y `FlowElement` (la unidad del árbol de jerarquía, es decir la hoja).

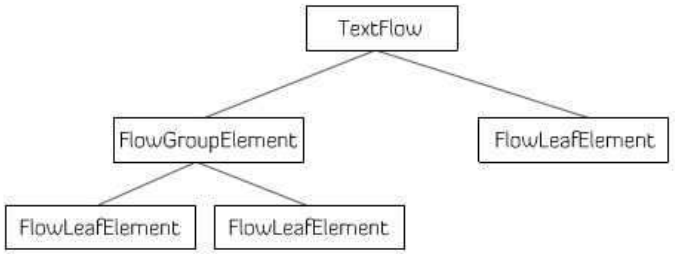


Fig. 4 Representa una historia a muy alto nivel.

Estas dos clases son del estilo abstracto. Las clases que se analizan a continuación descienden de la clase: `FlowElement`, y sus subclases son:

- **`FlowLeafElement`**
  - **`SpanElement`:** representa un bloque de texto.
  - **`InlineGraphicElement`:** representa un gráfico en una línea de texto.
- **`FlowGroupElement`**
  - **`ParagraphElement`:** es un conjunto de `FlowLeafElement`. No puede contener otro `ParagraphElement`.
  - **`DivElement`:** puede contener `ParagraphElement` o `DivElement`.
  - **`LinkElement`:** representa un hipervínculo y puede contener `FlowLeafElement`.
  - **`TCYElement`:** sirve para representar textos en dirección horizontal en un texto vertical. Por ejemplo, en la escritura japonesa, se puede escribir en dirección vertical y puede intercalarse texto en dirección horizontal sin alterar el texto que se encuentra en vertical. Las iniciales TCY se corresponden con los vocablos japoneses *tate-chu-yoko* (Horizontal dentro de vertical). Puede contener `FlowLeafElement`.

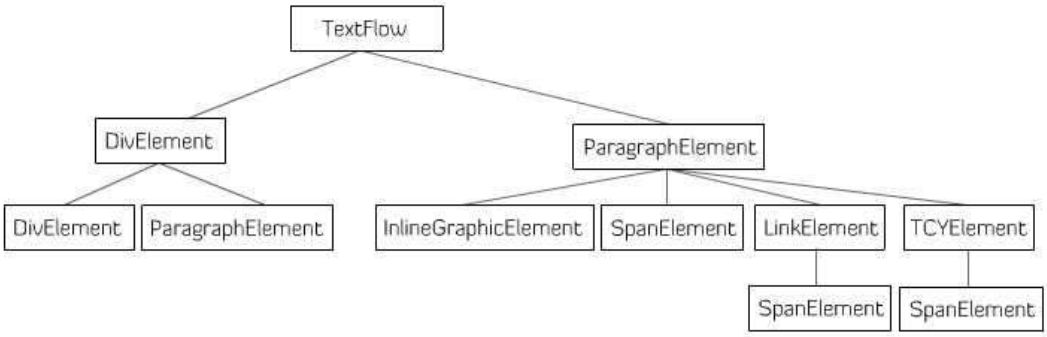


Fig. 5 Esquema de cómo se estructuran los `FlowElement`.

Formato

El formato está representado por su propia interfaz y clase: `ItextLayoutFormat` y `TextLayoutFormat`, respectivamente. La clase `FlowElement` implementa

`ItextLayoutFormat`, y además contiene una propiedad llamada: `format`, del tipo `TextLayoutFormat`. Esto significa que el modo de proporcionar un formato es o bien mediante la asignación de un valor a cada propiedad del objeto `FlowElement`, o bien asignando un objeto `TextLayoutFormat` a la propiedad `format`.

Esto puede ser útil para casos donde se tenga definido un estilo y sea necesario aplicar algún tipo de excepción en alguna palabra o frase. Al mejor estilo CSS, los objetos anidados heredan las propiedades de sus ancestros, pudiendo cada objeto sobrescribir el formato heredado.

Flujo

Está nuestro contenido con un formato asignado. Ahora, el desafío es despegarlo y que el contenido se acomode en líneas de textos, como habíamos hecho con el `while`, agregando y ubicando líneas. Esta complejidad se encuentra encapsulada en el paquete: `flashx.textLayout.factory`. En este paquete, encontramos las clases necesarias para el despliegue de las líneas de textos.

Implementando con Text Layout Framework

Al momento de esta edición, para poder trabajar con el Text Layout Framework con FlashDevelop, es necesario incorporar a nuestra carpeta de librería (lib) el SWC del Text Layout Framework, en donde se encuentran disponibles las correspondientes clases de ActionScript 3.0. Existe una manera de incorporar un *framework* entero a nuestro entorno de trabajo, conformado por varios paquetes, clases y eventualmente gráficos, utilizando un único archivo con extensión: SWC.

Al momento de esta edición, se pueden ver las novedades con respecto al desarrollo de este *framework* en: http://blogs.adobe.com/tlf/
Hay un documento de especificaciones en:
http://blogs.adobe.com/tlf/files/2010/12/TLF_Specification_1.11.pdf

Al momento de esta edición para utilizar las clases de Text Layout Framework con FlashDevelop, debemos copiar el archivo “textLayout.swc“, que se encuentra dentro de la carpeta del Flex SDK: frameworks\libs\, pegarlo en la carpeta lib de nuestro proyecto y vincular el SWC a nuestra librería para que esté disponible en nuestra aplicación. Desde FlashDevelop, abrimos nuestro proyecto en el panel *pro-*

ject (proyecto) y abrimos la carpeta lib. A continuación, hacemos un clic en el botón derecho del mouse al archivo: textLayout.swc, y tildamos: *Add to Library*

Implementando un párrafo simple

Vamos a desplegar el mismo párrafo que utilizamos con el TFE.

Ahora configuraremos como clase de documento a `MainTLF` con el siguiente código:

```
package
{
  import flashx.textLayout.elements.TextFlow;
  import flashx.textLayout.elements.ParagraphElement;
  import flashx.textLayout.elements.SpanElement;
  import flashx.textLayout.formats.TextLayoutFormat;
  import flashx.textLayout.factory.TextFlowTextLineFactory;
  import flash.geom.Rectangle;
  import flash.display.DisplayObject;
  import flash.display.Sprite;

  public class MainTLF extends Sprite
  {
    public function MainTLF()
    {
      //Creamos el contenido:
      var textContent:String = "";
      textContent += "Que es mi barco mi tesoro,\n";
      textContent += "que es mi dios la libertad,\n";
      textContent += "mi ley, la fuerza y el viento,\n";
      textContent += "mi única patria, la mar.";

      //Creamos la estructura del contenido.
      var textFlow:TextFlow = new TextFlow();
      var paragraph:ParagraphElement = new ParagraphElement();
      var span:SpanElement = new SpanElement();
```

```
      //Configuramos las estructura.
      paragraph.addChild(span);
      textFlow.addChild(paragraph);

      //Asignamos el contenido.
      span.text = textContent;

      //Configuramos el formato.
      var format:TextLayoutFormat = new TextLayoutFormat();
      format.fontFamily = "Papyrus";
      format.fontSize = 30;

      //Y lo asignamos al párrafo.
      paragraph.format = format;

      //Creamos un factory para desplegar las líneas.
      var factory:TextFlowTextLineFactory = new TextFlowText-
LineFactory();
      //Configuraremos el rectángulo dentro los límites donde se
distribuirán
      //las líneas de texto.
      var positionX:Number = 50;
      var positionY:Number = 50;
      var widthText:Number = 340;
      var heightText:Number = 100;
      factory.compositionBounds = new Rectangle(positionX, posi-
tionY,
                                                widthText,
heightText);
      //Y creamos el callback al método que agregará cada línea.
      factory.createTextLines(onTextLineAdded, textFlow);
    }

    //Callback que será llamado cada vez que una línea sea agregada
al
    //contenedor desde el factory.
    private function onTextLineAdded(textLine:DisplayObject):void
    {
      addChild(textLine);
```

```
    }
  }
}
```

En principio, verificamos que el despliegue de texto se puede dividir en estos pasos:

- Creamos y configuramos la estructura de los elementos.
- Creamos y asignamos el contenido.
- Creamos y asignamos el formato.
- Creamos y desplegamos el texto.

Para que nos sea más sencillo avanzar con nuestro ejemplo y ver otras opciones, dividimos las diferentes acciones que implementamos, en métodos. Esto nos permitirá focalizar nuestra atención en un método específico para abordar algún tema en particular. Por otro lado, es una buena práctica para tener nuestro código modularizado y fácil de mantener. Entonces, el código quedaría así:

```
package
{
  import flashx.textLayout.elements.TextFlow;
  import flashx.textLayout.elements.ParagraphElement;
  import flashx.textLayout.elements.SpanElement;
  import flashx.textLayout.formats.TextLayoutFormat;
  import flashx.textLayout.factory.TextFlowTextLineFactory;
  import flash.geom.Rectangle;
  import flash.display.DisplayObject;
  import flash.display.Sprite;

  public class MainTLF extends Sprite
  {
    private var _textFlow:TextFlow;
    private var _paragraph:ParagraphElement;
    private var _span:SpanElement;
    private var _format:TextLayoutFormat;
```

```
public function MainTLF()
{
  //Creamos y configuramos la estructura de los elementos.
  createAndSetStructure();

  //Creamos y asignamos el contenido.
  var contentText:String = getTextContent();
  setContent(contentText);

  //Creamos y asignamos el formato.
  var format:TextLayoutFormat = getDefaultFormat();
  setFormat(format);

  //Configuramos y desplegamos el texto.
  displayLines();
}

//Crea los elementos de TLF y les asignamos una estructura.
private function createAndSetStructure():void
{
  //Creamos la estructura del contenido.
  _textFlow = new TextFlow();
  _paragraph = new ParagraphElement();
  _span = new SpanElement();

  //Configuramos la estructura.
  _paragraph.addChild(_span);
  _textFlow.addChild(_paragraph);
}

//Devuelve un texto determinado.
private function getTextContent():String
{
  //Creamos el contenido.
  var textContent:String = "";
  textContent += "Que es mi barco mi tesoro,\n";
```

```
      textContent += "que es mi dios la libertad,\n";
      textContent += "mi ley, la fuerza y el viento,\n";
      textContent += "mi única patria, la mar.";
      return textContent;
    }

    private function setContent(contentText:String):void
    {
      _span.text = contentText;
    }

    private function getDefaultFormat():TextLayoutFormat
    {
      //Creamos y configuramos el formato.
      var format:TextLayoutFormat = new TextLayoutFormat();
      format.fontFamily = "Papyrus";
      format.fontSize = 30;
      return format;
    }

    //Creamos y asignamos el formato.
    private function setFormat(format:TextLayoutFormat):void
    {
      _paragraph.format = format;
    }

    //Creamos y configuramos el despliegue de texto.
    private function displayLines():void
    {
      //Creamos un factory para desplegar las líneas.
      var factory:TextFlowTextLineFactory = new TextFlowText-
LineFactory();
      //Configuraremos el rectángulo dentro los límites donde se
distribuirán
      //las líneas de texto.
      var positionX:Number = 50;
      var positionY:Number = 50;
```

```
      var widthText:Number = 340;
      var heightText:Number = 100;
      factory.compositionBounds = new Rectangle(positionX, posi-
tionY,
                                                widthText,
heightText);
      //Y creamos el callback al método que agregará cada línea.
      factory.createTextLines(addChild, textFlow);
    }
  }
}
```

Haremos un pequeño ajuste en el método: `displayLines`, reemplazando esta línea: `factory.createTextLines(onTextLineAdded, textFlow)` por esta: `factory.createTextLines(addChild, textFlow);`

Veamos otro detalle más. Para el despliegue de texto se ha utilizado la factoría: TextFlowTextLineFactory, que viene a representar a un texto estático que no permite operaciones de selección ni dinámicas. Este tipo de operación es conveniente cuando solo queremos desplegar un texto que no tendrá ningún tipo de interacción ni cambiará. Para estos casos, se debe usar la clase: TextFlowTextLineFactory, puesto que permite un ahorro de memoria.

Ahora que hemos visto lo básico, comprobemos las opciones disponibles con TLF, como el manejo de columnas, que no están disponibles con `TextField`.

Implementando un texto en columnas

Ahora veamos una implementación en donde necesitemos desplegar un texto en una cierta cantidad columnas. Podemos simplificar aún más el despliegue de texto que en el ejemplo anterior con TextFlowTextLineFactory, incluso generar un texto en una única columna. De esa forma, nos ahorra la creación y configuración de: TextFlow, ParagraphElement y SpanElement.

Colocaremos en la clase `TextContents` el contenido de textos obtenido a través de métodos estáticos. Así, mantenemos más limpia la clase principal y separamos la generación de contenido de su representación. Entonces, tendríamos clase al mismo nivel que `MainTFL`:

Clase: `TextContents`

```
package
{
  public class TextContents
  {
    public function TextContents()
    {
    }

    public static function get paragraph1():String
    {
      var textContent:String = "";
      textContent += 'Para quienes no crean la noticia, simplemente
tienen que '
      textContent += 'confirmarlo entrando a la web de Iron Maiden.
'
      textContent += 'La banda se va a presentar en Buenos Aires el
8 de abril '
      textContent += 'en el estadio Vélez Sarsfield. El Final Fron-
tier World Tour'
      textContent += 'tendrá un total de 29 shows y  uno de ellos
será en Argentina. '
      textContent += 'La banda se subirá una vez más al Ed Force
One en el mes '
      textContent += 'de febrero y el punto de partida para el res-
to de la gira '
      textContent += 'será Rusia.\n\n'

      textContent += '"Nos sorprendió la reacción del público ante
el Ed Force One '
      textContent += 'durante el último tour. Como todos nuestros
admiradores '
      textContent += 'saben de la película Flight 666 documenta to-
das las pruebas'
      textContent += ' y los problemas que sufrimos para conseguir
el proyecto '
      textContent += 'pudiera despegar, literalmente. El resultado
final hizo '
      textContent += 'que valiera la pena tanto esfuerzo", comentó
Bruce Dickinson.'
      textContent += ' Y continuó: Tanto la banda y como el equipo
disfrutaron'
      textContent += '" viajando de esta manera, así que nos pare-
ció lo más lógico'
```

```
      textContent += ' armar esta etapa del Final Frontier Tour de
la misma manera, '
      textContent += 'así podríamos ver a una mayor cantidad de
fans en todo el'
      textContent += 'mundo. Pero además, esta estamos corriendo
los límites y '
      textContent += 'vamos a llegar más lejos de lo que Maiden
llegó hasta el '
      textContent += 'momento". De acuerdo con la página de la ban-
da, '
      textContent += 'las entradas comenzarían a venderse a partir
del 27 de noviembre.'
      textContent += ' Además, también corre la versión por ahí que
las fechas '
      textContent += 'anunciadas hasta hoy no son las únicas. Hay
que estar atento'
      textContent += ' para ver dónde se sumarán nuevos shows. ';
      return textContent;
    }
  }
}
```

Y ahora que tenemos el contenido separado, nos focalizaremos en su representación. Creamos una nueva clase y la configuramos como clase de documento, bajo el nombre: `MulticolumnTest`. En ella, solo nos hará instanciar la clase: `flashx.textLayout.factory.StringTextLineFactory`. A la cual le asignaremos un formato en el cual estará establecida la cantidad de columnas y el rectángulo dentro del cual se desplegará el texto y su contenido.

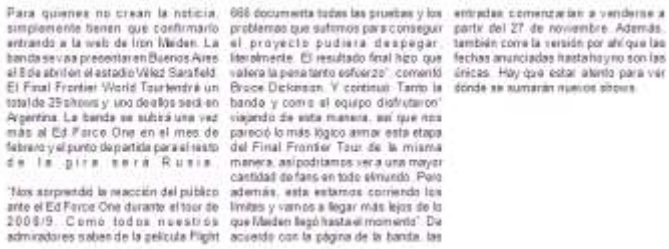
```
package
{
  import flashx.textLayout.formats.TextLayoutFormat;
  import flashx.textLayout.formats.TextAlign;
  import flashx.textLayout.formats.TextJustify;
  import flashx.textLayout.factory.StringTextLineFactory
  import flash.geom.Rectangle;
  import flash.display.DisplayObject;
  import flash.display.Sprite;
  public class MulticolumnTest extends Sprite
```

```
  {
    public function MulticolumnTest()
    {
      //Creamos un factory para desplegar las líneas.
      var factory:StringTextLineFactory = new StringTextLineFacto-
ry();

      //Configuraremos el rectángulo dentro los límites donde se
distribuirán
      //las líneas de texto: posición x, posición y, ancho y alto.
      factory.compositionBounds = new Rectangle(50, 50, 800, 300);

      //Creamos el formato.
      var format:TextLayoutFormat = new TextLayoutFormat();
      format.fontFamily = "Arial";
      format.fontSize = 15;
      format.columnCount = 3;
      format.columnGap = 15;
      //Configuramos el texto con alineación justificada.
      format.textAlign = TextAlign.JUSTIFY;
      //Establecemos como se comportará el justificado, en
      //este caso la separación se hará entre caracteres.
      format.textJustify = TextJustify.DISTRIBUTE
      //Asignamos el formato.
      factory.textFlowFormat = format;
      //Asignamos el texto.
      factory.text = TextContents.paragraph1;
      //Creamos las líneas.
      factory.createTextLines(addChild);
    }
  }
}
```

Y quedaría de la siguiente manera:



Para quienes no crean la noticia, simplemente tienen que confirmarlo entrando a la web de Iron Maiden. La banda se va a presentar en Buenos Aires el 8 de abril en el estadio Vélez Sarsfield. El Final Frontier World Tour tendrá un total de 25 shows y uno de ellos será en Argentina. La banda se subirá una vez más al Ed Force One en el mes de febrero y el punto de partida para el resto de la gira será Rusia.

'Nos sorprendió la reacción del público ante el Ed Force One durante el tour de 2008/9. Como todos nuestros admiradores saben de la película Flight 666 documenta todas las pruebas y los problemas que sufrimos para conseguir el proyecto pudiera despegar literalmente. El resultado final hizo que valiera la pena tanto esfuerzo', comentó Bruce Dickinson. Y continuó: 'Tanto la banda y como el equipo disfrutaron viajando de esta manera, así que nos pareció lo más lógico armar esta etapa del Final Frontier Tour de la misma manera, así podríamos ver a una mayor cantidad de fans en todo el mundo. Pero además, esta estamos corriendo los límites y vamos a llegar más lejos de lo que Maiden llegó hasta el momento'. De acuerdo con la página de la banda, las entradas comenzarían a venderse a partir del 27 de noviembre. Además, también corre la versión por ahí que las fechas anunciadas hasta hoy no son las únicas. Hay que estar atento para ver donde se sumarán nuevos shows.

Fig. 6

Distribuyendo un texto en los contenedores

Ya hemos estudiado la implementación de texto estático, donde hemos aplicado fácilmente la distribución en columnas, incluso hemos determinado la forma en la que se verá el justificado. En las versiones del reproductor Flash 10, el alineado justificado no está disponible ni tampoco lo está en la clase `TextField`.

Existen muchos ajustes para el formato de texto, que están fuera del área de interés de un desarrollador, por lo que la orientación de un diseñador es crucial. También está disponible una extensa y detallada documentación en la "Referencia del lenguaje ActionScript 3.0", de Adobe.

La distribución de texto entre contenedores amplía las posibilidades de manejo del diseño y el formato. Esto permite el flujo del texto entre contenedores de distintos tamaños y posiciones, rodear imágenes con texto, trabajar con selección de texto, generar interactividad, entre otras actividades.

Entidades que intervienen

Para entender cómo es la estructura y el manejo de este tipo de técnica, haremos un paralelo con el MVC (Modelo, Vista, Controlador). Hay una explicación y un ejemplo de implementación detallada del MVC al abordar el tema de eventos.

Veamos una explicación rápida del MVC de una arquitectura que consiste en tres capas:

- **Modelo:** maneja todo lo referido a los datos. En nuestro ejemplo, es el `TextFlow` texto y formatos.
- **Vista:** es la representación visual de los textos y formatos. En nuestro ejercicio, son las líneas de texto desplegadas dentro de los `Sprite` que actúan como contenedores.
- **Controlador:** es el que mantiene comunicados a la Vista y al Modelo. Toma los eventos y cambios procedentes de la Vista y se los traduce al Modelo, y toma los cambios del Modelo y se lo transmite a la Vista para que lo represente. En nuestro ejemplo, existe un controlador por cada contenedor, encargado de representar los datos línea a línea. Y un Compositor que calcula, en función del tamaño del contenedor y el formato del contenido, cuantas líneas entran, así se las da al Controlador para que las represente. Luego, si le queda contenido por desplegar, el Compositor distribuye entre sus Controladores el texto que tiene para desplegar, aplicando los mismos pasos.

Veamos una figura:

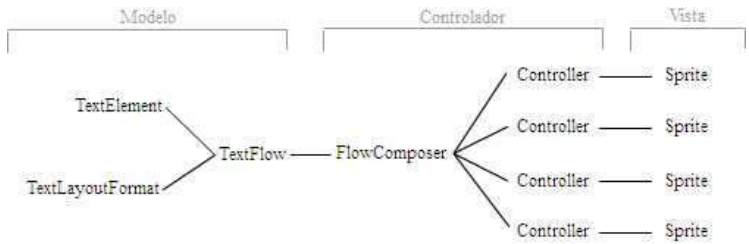


Fig. 7 Relaciones entre las entidades en el despliegue de contenido en contenedores.

Entonces, para implementarlo debemos seguir, a grandes rasgos, estas etapas.

En primer lugar, creamos:

- El `TextFlow` y sus elementos
- Los formatos
- El compositor (manejador de los controladores)
- Los contenedores
- Los controladores pasándoles sus respectivos contenedores

En segundo lugar, realizamos las siguientes asignaciones:

- La jerarquía de los elementos dentro del `TextFlow`
- Formato al contenido
- A las diferentes jerarquías: texto, imágenes, etcétera
- El Compositor al `TextFlow`
- Los Controladores al compositor

En tercer lugar, iniciamos la representación:

- Ejecutamos en el Compositor la orden de representar el contenido.

Por último, habilitamos la selección:

- Permitimos que los textos queden seleccionados, lo que nos permite resaltar texto al seleccionarlo, también copiar y pegar.

En nuestra implementación, agregaremos dos imágenes (bruceEDforce1.jpg y edForceOne.jpg). Estas imágenes las hemos colocado en una carpeta llamada *Images*, al mismo nivel donde se publica el swf.

Hacemos un ajuste en la clase `TextContents`. Se fracciona el texto que se ha probado en varias variables, ya que no todas llevarán el mismo formato.

Clase `TextContents`

```
package
{
  public class TextContents
  {
    public function TextContents()
    {
    }

    public static function get paragraph1():String
    {
```

```
      var textContent:String = "";
      textContent += 'Para quienes no crean la noticia, simplemente
tienen que '
      textContent += 'confirmarlo entrando a la web de Iron Maiden.
'
      textContent += 'La banda se va a presentar en Buenos Aires el
8 de abril '
      textContent += 'en el estadio Vélez Sarsfield. El Final Fron-
tier World Tour'
      textContent += 'tendrá un total de 29 shows y uno de ellos
será en Argentina. '
      textContent += 'La banda se subirá una vez más al Ed Force
One en el mes '
      textContent += 'de febrero y el punto de partida para el res-
to de la gira será Rusia.\n\n'
      return textContent;
    }

    public static function get paragraph2():String
    {
      var textContent:String = "";
      textContent += '"Nos sorprendió la reacción del público ante
el Ed Force One '
      textContent += 'durante el último tour. Como todos nuestros
admiradores '
      textContent += 'saben de la película Flight 666 documenta to-
das las pruebas'
      textContent += ' y los problemas que sufrimos para conseguir
que el proyecto '
      textContent += 'pudiera despegar, literalmente. El resultado
final hizo '
      textContent += 'que valiera la pena tanto esfuerzo"';
      return textContent;
    }

    public static function get paragraph3():String
    {
      var textContent:String = "";
      textContent += ', comentó Bruce Dickinson. Y continuó: ';
      return textContent;
    }
```

```
    public static function get paragraph4():String
    {
      var textContent:String = "";
      textContent += '"Tanto la banda y como el equipo disfrutaron'
      textContent += ' viajando de esta manera, así que nos pareció
lo más lógico'
      textContent += ' armar esta etapa del Final Frontier Tour de
la misma manera, '
      textContent += 'así podríamos ver a una mayor cantidad de
fans en todo el'
      textContent += ' mundo. Pero además, esta estamos corriendo
los límites y '
      textContent += 'vamos a llegar más lejos de lo que Iron Mai-
den llegó hasta el '
      textContent += 'momento".     '
      return textContent;
    }

    public static function get paragraph5():String
    {
      var textContent:String = "";
      textContent += '. De acuerdo con la página de la banda, '
      textContent += 'las entradas comenzarían a venderse a partir
del 27 de noviembre.'
      textContent += ' Además, también corre la versión por ahí que
las fechas '
      textContent += 'anunciadas hasta hoy no son las únicas. Hay
que estar atento'
      textContent += ' para ver dónde se sumarán nuevos shows. ';
      return textContent;
    }
  }
}
```

A continuación, la clase que hará el trabajo de crear y desplegar texto e imágenes.

Clase: `TextContainerTest`

```
package
{
  import flashx.textLayout.formats.TextLayoutFormat;
  import flashx.textLayout.formats.TextAlign;
  import flashx.textLayout.formats.TextJustify;

  import flashx.textLayout.elements.InlineGraphicElement;
  import flashx.textLayout.elements.TextFlow;
  import flashx.textLayout.elements.ParagraphElement;
  import flashx.textLayout.elements.SpanElement;

  import flashx.textLayout.compose.StandardFlowComposer;

  import flashx.textLayout.container.ContainerController;

  import flashx.textLayout.edit.SelectionManager;

  import flash.text.FontStyle;

  import flash.geom.Rectangle;
  import flash.display.DisplayObject;
  import flash.display.Sprite;

  public class TextContainerTest extends Sprite
  {
    public function TextContainerTest()
    {
      //========================================
      //Creamos los objetos
      //========================================

      // TextFlow y sus elementos
      var textFlow:TextFlow = new TextFlow();
```

```
      var paragraph1:ParagraphElement = new ParagraphElement();
      var paragraph2:ParagraphElement = new ParagraphElement();

      var span1:SpanElement = new SpanElement();
      var span2:SpanElement = new SpanElement();
      var span3:SpanElement = new SpanElement();
      var span4:SpanElement = new SpanElement();

      var image1:InlineGraphicElement = new InlineGraphicElement();
      var image2:InlineGraphicElement = new InlineGraphicElement();

      //Los formatos.
      var formatNormal:TextLayoutFormat = new TextLayoutFormat();
      formatNormal.fontFamily = "Garamount";
      formatNormal.fontSize = 14;
      formatNormal.textAlign = TextAlign.JUSTIFY;
      formatNormal.textJustify = TextJustify.INTER_WORD;

      var formatBoldItalic:TextLayoutFormat = new TextLayoutFor-
mat();
      formatBoldItalic.fontFamily = "Times New Roman";
      formatBoldItalic.fontStyle = FontStyle.ITALIC;
      formatBoldItalic.fontWeight = FontStyle.BOLD;

      //El compositor.
      var composer:StandardFlowComposer = new StandardFlowCompos-
er();

      //Los contenedores.
      var container1:Sprite = new Sprite();
      var container2:Sprite = new Sprite();

      //Los controladores.
      var controler1:ContainerController = new ContainerControl-
ler(container1,

500, 332);
      var controler2:ContainerController = new ContainerControl-
ler(container2,
```

```
500, 300);

      //=======================================
      //Realizamos las asignaciones
      //=======================================

      //La jerarquía de los elementos dentro del TextFlow
      textFlow.addChild(paragraph1);
      textFlow.addChild(paragraph2);

      paragraph1.addChild(image1);

      paragraph2.addChild(span1);
      paragraph2.addChild(span2);
      paragraph2.addChild(span3);
      paragraph2.addChild(span4);
      paragraph2.addChild(image2);



      //Los formatos
      paragraph2.format = formatNormal;

      span2.format = formatBoldItalic;
      span4.format = formatBoldItalic;

      controler2.columnCount = 3;
      controler2.columnGap = 20;

      //Los textos.
      span1.text = TextContents.paragraph1;
      span2.text = TextContents.paragraph2;
      span3.text = TextContents.paragraph3;
      span4.text = TextContents.paragraph4;

      //Las rutas de las imágenes:
```

```
      image1.source = "./images/bruceEDforce1.jpg";
      image1.width = 500;
      image1.height = 330;

      image2.source = "./images/edForceOne.jpg";
      image2.width = (500/3)-15;
      image2.height = 110 - 20;

      //Ubicamos al segundo container para que esté por debajo del
primero.
      container2.y = 340;

      //El compositor al TextFlow.
      textFlow.flowComposer = composer;

      //Los controladores al compositor.
      composer.addController(controler1);
      composer.addController(controler2);

      //Agregamos los contenedores al display list
      addChild(container1);
      addChild(container2);

      //=======================================
      //Representamos el contenido
      //=======================================

      composer.updateAllControllers();

      //=======================================
      //Habilitamos la selección de texto.
      //=======================================

      textFlow.interactionManager = new SelectionManager();
    }
  }
}
```

Y el resultado aparecería del siguiente modo:

Fig. 8 Vemos el resultado de la clase: TextContainerTest.

Implementando Text Layout Markup

Si analizamos la estructura de los elementos con los que estamos trabajando: `TextLayout`, `DivElement`, `ParagraphElement`, `SpanElement`, inmediatamente los podemos relacionar con el formato CSS, y sino tenemos una noción de esto es fácil vincular esta estructura con una estructura XML. De todas maneras, la buena noticia es que podemos generar el contenido y la estructura de texto desde un XML.

Esto nos brinda mucha más flexibilidad a la hora de generar el contenido. Ya que podemos cambiar la estructura, el texto, la imagen y el formato desde un XML sin necesidad de compilar un swf. Incluso, utilizando el XML o un `string` que lo

represente directamente en nuestro código, nos resultará mucho más sencillo manejar los cambios y mantener nuestra aplicación. Además, tenemos la ventaja de "importar", es decir, transformar un `string` en un formato y contenido TLF; o "exportar", convertir un formato TFL a `string`.

Sigamos con nuestro ejemplo y transformemos nuestro contenido, construido y estructurado desde ActionScript 3.0, a uno tomado desde un `String`. Entonces, veamos primero cómo se representan las estructuras de TLF con Text Layout Markup.

Paralelo entre Text Layout Markup y los objetos TLF

El elemento raíz de todo TLF es el `TextLayout`, que puede contener a `DivElement` y `ParagraphElement` así como al resto de los objetos. Ahora bien, las etiquetas definen el tipo de elemento y los atributos sus propiedades.

Repasemos la estructura del ejercicio anterior:

```
//La jerarquía de los elementos dentro del TextFlow
textFlow.addChild(paragraph1);
textFlow.addChild(paragraph2);

paragraph1.addChild(image1);

paragraph2.addChild(span1);
paragraph2.addChild(span2);
paragraph2.addChild(span3);
paragraph2.addChild(span4);
paragraph2.addChild(image2);
```

Lo cual se refleja en forma de XML de la siguiente manera:

```
<TextFlow>
  <p>
    <img/>
  </p>
  <p>
    <span/>
```

```
    <span/>
    <span/>

 <span/>
    <img/>
  </p>
</TextFlow>
```

Si bien no son muchas las etiquetas que se utilizan, existe un método que genera a partir de un `TextFlow` un `String`, donde dependiendo de los valores que optemos, podrá ser un `String` con el texto sin formato del `TextFlow`, o bien un XML o un HTML, que describe la estructura, texto y formato.

Podemos, a su vez, recuperar este `String` para que venga con las etiquetas y atributos correspondientes a un `TextFlow`. Entonces, en un solo paso, crearlo, ahorrándose la creación de los elementos de texto, la asignación del formato, texto o imagen, y la generación de la estructura. Veamos qué opciones existen para exportar e importar.

Supongamos que queremos generar el texto necesario para representar el contenido, formato y estructura, pero no sabemos exactamente cómo nombrar las etiquetas o propiedades. La forma más directa y sencilla es armando la estructura por código y luego utilizando la clase: `flashx.textLayout.conversion.TextConverter`. Y de ella utilizar el método estático: `export`.

La firma de este método es el siguiente:

```
public static function export(source:TextFlow,
                              conversionFormat:String,
                              conversionType:String):Object
```

A continuación, detallamos los parámetros que acepta:

- **`source:TextFlow`** → el objeto `TextFlow` que queremos exportar.
- **`conversionFormat:String`** → es el formato de exportación, que incluye:
  - **`TextConverter.PLAIN_TEXT_FORMAT`**: convierte el texto con formato a texto plano, es decir, en un `String` liso y llano.
  - **`TextConverter.TEXT_FIELD_HTML_FORMAT`**: convierte el contenido, formato y estructura al formato HTML que soporta la clase `TextField`.
  - **`TextConverter.TEXT_LAYOUT_FORMAT`**: traduce el formato de este objeto a un XML en donde se encuentra la estructura, contenido y formato, que luego puede ser recreado a través de este XML.
- **`conversionType:String`** → es el tipo de conversión que esperamos obtener. Pueden ser: `ConversionType.STRING_TYPE` para texto plano sin formato o `ConversionType.XML_TYPE`.

Nos puede surgir la pregunta: ¿qué pasa si con el valor de conversión de formato pasamos alguno que no es de texto plano, pero en el tipo de conversión elegimos texto plano? ¿Qué tipo de devolución tendremos haciendo combinaciones no compatibles de texto plano con texto con marcación?

Como aporte invaluable a la ciencia y a la humanidad estas son las combinaciones y los resultados que obtendríamos si combinamos:

conversionFormat: `TextConverter.TEXT_FIELD_HTML_FORMAT` ó `TextConverter.TEXT_LAYOUT_FORMAT`

con

conversionType: `ConversionType.STRING_TYPE`

Obtendremos: Texto con marcación.

---

conversionFormat: `TextConverter.PLAIN_TEXT_FORMAT`

con

conversionType: `ConversionType.XML_TYPE`

Obtendremos: null

Y el resultado del XML con el contenido, estructura y formato es el siguiente, parte del contenido fue suprimido por razones de legibilidad al mejor estilo *Seinfield*:

```
<TextFlow columnCount="inherit" columnGap="inherit"
columnWidth="inherit" lineBreak="inherit" paddingBottom="inherit"
paddingLeft="inherit" paddingRight="inherit" paddingTop="inherit"
verticalAlign="inherit" whiteSpaceCollapse="preserve"
xmlns="http://ns.adobe.com/textLayout/2008">
  <p>
    <img height="330" width="500"
source="./images/bruceEDforce1.jpg"/>
    <span></span>
  </p>
  <p fontFamily="Garamount" fontSize="14" textAlign="justify"
textJustify="interWord">
    <span>Para quienes no crean la noticia y yara, yara, ya-
ra...</span>
    <span fontFamily="Times New Roman" fontStyle="italic"
fontWeight="bold">"Nos sorprendió la reacción y yara, yara, ya-
ra..."</span>
    <span>, comentó Bruce Dickinson. Y continuó:</span>
    <span fontFamily="Times New Roman" fontStyle="italic" font-
Weight="bold">"Tanto la banda y yara, yara, yara...</span>
    <img height="90" width="151.66666666666666"
source="./images/edForceOne.jpg"/>
    <span></span>
  </p>
</TextFlow>
```

El resto de los datos están tal cual los ha generado el método. Una vez que tenemos el XML generado, por supuesto que podríamos agregar una capa que se encargue de tomar este XML, y persistirlo para luego recuperarlo, plasmando esa estructura en un objeto vivo, en ActionScript. Al referirmos a un objeto vivo queremos denotar un objeto instanciado en memoria. Veamos ahora cómo tomar los datos.

Al tomar los datos de un XML y convertirlos en un objeto vivo en ActionScript, este método procesa y devuelve un `TextFlow`. Esta es la firma del método:

```
public function TextConverter.importToFlow(sourceContent:String,
                                            conversionFor-
mat:String):TextFlow;
```

Utilizando un `String` con formato `TextConverter.TEXT_LAYOUT_FORMAT`, nos ahorramos de importar varias clases, y tenemos la posibilidad de cambiar formato, estructura y contenido desde un texto plano, desplegándolo en un solo paso.

En la clase `TextContents`, agregamos un *getter* para la propiedad: `htmlTxt` y le asignamos el `String` que obtuvimos, es decir, tenemos un `String` que contiene el contenido y formato.

Cuando queremos utilizar el `String` que nos devuelve el método `export`, hay que tener en cuenta que se deben colocar las etiquetas una a continuación de la otra, al estilo: `<p><span>yara, yara, yara...</span></p>`.

Esta propiedad será cargada luego desde nuestra clase. Es interesante resaltar que ha quedado mucho más simplificada:

```
package
{
  import flashx.textLayout.conversion.TextConverter;
  import flashx.textLayout.conversion.ConversionType;
  import flashx.textLayout.elements.TextFlow;
  import flashx.textLayout.compose.StandardFlowComposer;
```

```
  import flashx.textLayout.container.ContainerController;
  import flashx.textLayout.edit.SelectionManager;

  import flash.geom.Rectangle;
  import flash.display.DisplayObject;
  import flash.display.Sprite;

  public class TextContainerTest extends Sprite
  {
    public function TextContainerTest()
    {
      //========================================
      //Creamos los objetos
      //========================================

      // TextFlow y sus elementos
      var textFlow:TextFlow = new TextFlow();

      //El compositor.
      var composer:StandardFlowComposer = new StandardFlowCompos-
er();

      //Los contenedores.
      var container1:Sprite = new Sprite();
      var container2:Sprite = new Sprite();

      //Los controladores.
      var controler1:ContainerController = new ContainerControl-
ler(container1,

500, 332);
      var controler2:ContainerController = new ContainerControl-
ler(container2,

500, 300);
      controler2.columnCount = 3;
      controler2.columnGap = 20;

      var sourceContent:String = TextContents.htmlString;
```

```
      var conversionFormat:String = TextConvert-
er.TEXT_LAYOUT_FORMAT;
      textFlow = TextConverter.importToFlow(sourceContent, conver-
sionFormat);

      //Ubicamos al segundo container para que esté por debajo del
primero.
      container2.y = 340;

      //El compositor al TextFlow.
      textFlow.flowComposer = composer;

      //Los controladores al compositor.
      composer.addController(controler1);
      composer.addController(controler2);

      //Agregamos los contenedores al display list
      addChild(container1);
      addChild(container2);

      //=======================================
      //Representamos el contenido
      //=======================================

      composer.updateAllControllers();

      //=======================================
      //Habilitamos la selección de texto.
      //=======================================

      textFlow.interactionManager = new SelectionManager();
    }
  }
}
```