

Tutorial 11 - Utilización de componentes

Paso 1 de 14



Un componente es un objeto reutilizable que puede incluir gráficos y código preconfigurados, y cuyos parámetros pueden ser modificados fácilmente.

Vamos a utilizar componentes para construir una minigalería de imágenes y un campo de texto.

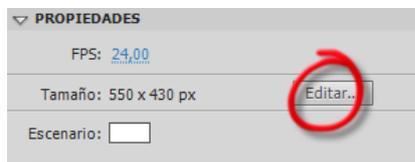
Las imágenes y el texto que vamos a utilizar en este tutorial se encuentran en el archivo [recursosTutorial11.zip](#).

Este archivo contiene una carpeta llamada *images* que deberemos colocar en la misma carpeta en la que tengamos nuestro archivo Flash.



Paso 2 de 14

Cambiamos el tamaño del escenario a 550 x 430 píxeles.



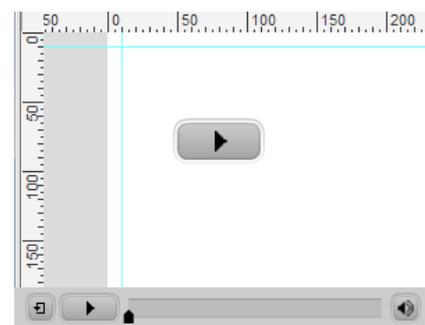
Vamos a dejar un margen de 10 píxeles entre los elementos que coloquemos en el escenario y los límites del documento.

Para facilitar el posicionamiento de objetos en puntos específicos del escenario podemos ayudarnos de guías.

Seleccionamos **Ver > Reglas** y nos aseguramos de que la casilla **Ver > Guías > Mostrar guías** esté seleccionada.

Hacemos clic sobre las reglas y arrastramos para posicionar cuatro guías en los lugares que necesitamos, que en este caso es a 10 y 420 píxeles en las guías horizontales, y a 10 y 540 en las verticales.

Para colocar las guías en estas posiciones podemos arrastrarlas hasta una posición aproximada, y después haciendo doble clic sobre ellas podemos introducir el valor exacto.

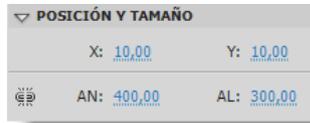


Paso 3 de 14

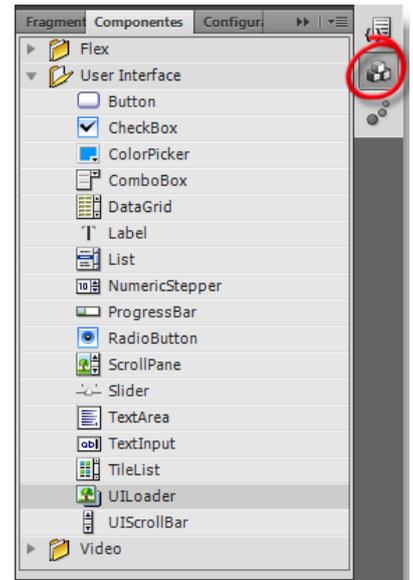
El primer componente que vamos a utilizar es **UILoader**, un contenedor que permite cargar con facilidad imágenes o archivos swf externos a la aplicación.

Abrimos el panel **Componentes**, y en la carpeta **User Interface** hacemos doble clic sobre el componente **UILoader**. Aparecerá en el centro del escenario una instancia de este componente.

Vamos a utilizar este componente para cargar la imagen principal de nuestra galería. En el inspector de Propiedades le asignamos un tamaño de **AN:400** y **AL:300**, que es el tamaño de nuestras imágenes.



Posicionamos el contenedor en **X:10** e **Y:10**. Podemos ayudarnos de las guías que hemos creado para ello, o bien utilizar el inspector de Propiedades.

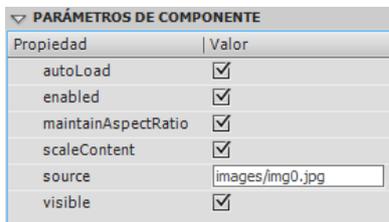


Paso 4 de 14

En el inspector de Propiedades podemos ver que aparece la sección **Parámetros de componente** en la que podemos modificar algunas de las propiedades de ese componente.

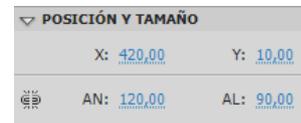
Desde aquí podemos indicar un valor para **source**, indicando la ruta del contenido que queremos que se muestre en ese contenedor.

Haremos que al abrir la aplicación se muestre la primera imagen, por lo que en **source** podemos indicar **images/img0.jpg**, ya que es la ruta relativa de la imagen respecto a nuestro archivo Flash.



Probamos la película con **Ctrl+Intro** para comprobar que la imagen se carga correctamente.

Para las miniaturas utilizaremos también el componente **UILoader**. Creamos otra instancia del componente y le damos los valores **AN:120** y **AL:90**. Con ayuda de las guías, colocamos esta instancia en el extremo superior derecho del escenario.

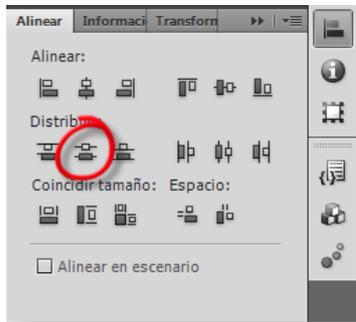


Para crear las otras tres miniaturas podemos partir de una copia de esta última, y así ya tendrán los valores de anchura y altura apropiados

Paso 5 de 14

Con ayuda de las guías, colocamos otra de las miniaturas en la parte inferior derecha del escenario. Quedará posicionada en AN:420 e Y:330.

Para alinear las dos miniaturas restantes podemos ayudarnos de las guías para posicionar el margen derecho, y del panel **Alinear** para posicionar la altura, seleccionando las cuatro miniaturas y **Distribuir verticalmente respecto al centro**.



Ahora que ya tenemos colocados todos los contenedores en el escenario, vamos a darles **nombres de instancia** a cada uno de ellos desde el inspector de Propiedades.

Llamaremos *main_ldr* al contenedor principal, y *thumb0_ldr*, *thumb1_ldr*, *thumb2_ldr* y *thumb3_ldr* a las miniaturas.

Llamamos *loaders* a la capa que contiene todos los contenedores.

Paso 6 de 14

Creamos una nueva capa llamada *as* para la programación.

Antes hemos comprobado que si escribimos la ruta de una imagen en el parámetro *source* del componente, la imagen se mostrará en el contenedor correspondiente.

Si definimos la propiedad *source* de una instancia mediante código, veremos que funciona de la misma manera.

Por ejemplo, si escribimos la línea:

```
thumb0_ldr.source = "images/thumbs/img0.jpg";
```

comprobaremos que se carga la imagen correspondiente en el primer contenedor de las miniaturas.

Podríamos repetir el mismo código para las cuatro instancias cambiando sólo los números correspondientes, pero si nuestra galería tuviera muchas imágenes podría ser poco práctico hacerlo de esa manera. Veamos una manera de optimizar el código.

En primer lugar definiremos una variable que almacene el número de imágenes de nuestra galería, que en este caso son cuatro.

```
var numImages:uint = 4;
```

En otros tutoriales utilizábamos `Number` como tipo genérico cuando tratábamos con números, pero para números enteros positivos es más correcto utilizar el tipo más específico `uint` (en el caso de enteros positivos y negativos se utilizaría el tipo `int`).

Vamos a crear un bucle `for` que se ejecute cuatro veces (el número de nuestras imágenes), y que vaya cargando la imagen correspondiente en cada miniatura.

```
for (var i:uint=0; i<numImages; i++)
{
    this["thumb"+i+"_ldr"].source = "images/thumbs/
img"+i+".jpg";
}
```

Paso 7 de 14

Al tener nuestra variable `numImages` el valor de 4, podemos ver que en la primera línea del código:

```
for (var i:uint=0; i<numImages; i++)
```

se define que el bucle se repetirá 4 veces, con los valores para `i` de 0, 1, 2 y 3.

Si tienes dudas sobre el uso del bucle `for` puedes consultar el [paso 7 del tutorial 10](#).

Continuando con el análisis del código, en el fragmento:

```
this["thumb"+i+"_ldr"]
```

se sustituirá a `i` por los diferentes valores que va adoptando en las repeticiones, creando con ello las cadenas `this["thumb0_ldr"]`, `this["thumb1_ldr"]`, etc.

El `this` buscará una instancia cuyo nombre coincida con la cadena que se encuentra entre los corchetes, paso necesario al escribir los nombres de instancia de forma dinámica, lo que equivaldría a haber escrito directamente el nombre de la instancia.

En el fragmento:

```
"images/thumbs/img"+i+".jpg"
```

se sustituirá igualmente la `i` por su valor correspondiente en el bucle.

Por tanto, este bucle equivaldrá a haber escrito:

```
thumb0_ldr.source = "images/thumbs/img0.jpg";  
thumb1_ldr.source = "images/thumbs/img1.jpg";  
thumb2_ldr.source = "images/thumbs/img2.jpg";  
thumb3_ldr.source = "images/thumbs/img3.jpg";
```

lo cual mostrará las cuatro imágenes en su contenedor correspondiente.

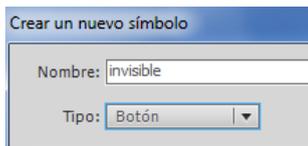
Paso 8 de 14

Lo siguiente que necesitamos para nuestra galería es que al hacer clic sobre las miniaturas se muestre en el contenedor principal la imagen correspondiente.

Para que se reconozca con más facilidad que las miniaturas tienen la función de botones, podemos variar ligeramente su aspecto al situarnos al hacer clic sobre ellas, por ejemplo iluminándolas u oscureciéndolas según nuestra interacción.

Una forma eficaz de crear esta funcionalidad y cambiar el aspecto de todas las miniaturas, es crear un botón invisible y colocar una instancia del botón sobre cada miniatura.

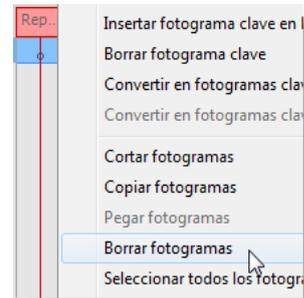
Seleccionamos **Insertar > Nuevo símbolo**, tipo **Botón**, y le damos el nombre *invisible*.



Dibujamos un rectángulo sin trazo, con relleno negro con un alfa del 50%, y con tamaño de 120x90 (el mismo que las miniaturas). Lo alineamos en la posición 0x0 de su escenario.

Pulsamos **F6** sobre los fotogramas *sobre* y *presionado* para crear fotogramas clave con el mismo contenido que en el fotograma *reposo*.

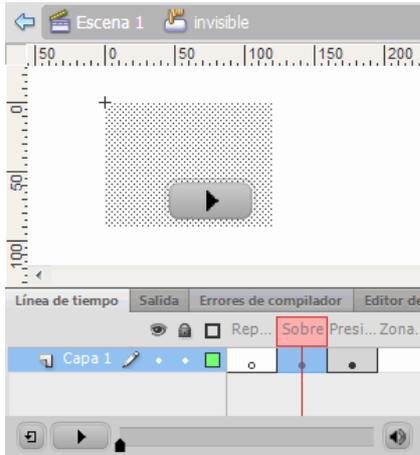
Borramos el contenido del fotograma *reposo* haciendo clic con el **botón derecho** del ratón sobre el fotograma *reposo* y seleccionando **Borrar fotogramas**.



Cambiamos el color del fotograma *Sobre* de negro a blanco, y con una transparencia del 20%.

Paso 9 de 14

De esta forma habremos construido un botón que no se ve en su estado de reposo (pero que es igualmente funcional). Si estamos sobre él mostrará un recuadro blanco semitransparente (parecerá iluminar lo que se encuentre por debajo de él) y, de la misma forma, oscurecerá lo que esté detrás del botón mientras estemos haciendo clic sobre él.



Con el botón ya creado, volvemos a la escena principal.

Creamos una nueva capa llamada *buttons* por encima de la capa *loaders*.

Arrastramos al escenario cuatro instancias del botón *invisible*, y colocamos cada una de ellas sobre cada contenedor de las miniaturas.

Como el botón *invisible* no tiene ningún gráfico en su estado de reposo, veremos un recuadro azul para indicar la posición y tamaño de la instancia del botón en el escenario, y facilitar de este modo su posicionamiento.

Aunque estos botones todavía no tengan funcionalidad, podemos probar nuestra película para ver el efecto de iluminar y oscurecer que hemos conseguido con nuestros botones.

Damos a estos botones los nombres de instancia *thumb0_btn*, *thumb1_btn*, *thumb2_btn* y *thumb3_btn*.

El siguiente paso será añadir el código necesario para que esos botones que hemos creado permitan cargar las imágenes correspondientes a cada miniatura.

Paso 10 de 14

Al haber dado a cada botón un nombre de instancia con un número que corresponde también al de cada imagen, podemos añadir su funcionalidad en el mismo bucle en el que hemos cargado las imágenes de las miniaturas.

Añadimos en el bucle `for` la línea:

```
this["thumb"+i+"_btn"].addEventListener(MouseEvent.CLICK, loadImage);
```

Con esa línea habremos añadido un listener a cada botón, llamando todos ellos a la función `loadImage`.

Como todos los botones llaman a la misma función, ésta deberá recuperar el nombre del botón que la lanzó para saber qué imagen cargar en el contenedor principal. Como vimos en otros tutoriales, podemos identificar el nombre de la instancia que desencadena el evento con `e.target.name`.

Probemos con la siguiente definición de la función `loadImage`:

```
function loadImage(e:MouseEvent):void
{
    trace(e.target.name);
}
```

Si probamos nuestra película y pulsamos sobre los diferentes botones, comprobaremos que, efectivamente, aparecen los nombres de las instancias en el panel Salida.

Nuestro contenedor principal (`main_ldr`) deberá cargar una imagen que se encuentra en la carpeta `images`, y cuyo nombre es `img` seguido del mismo número que contiene el nombre de la instancia del botón que hemos pulsado, y seguido de `.jpg` (por ejemplo `images/img2.jpg`).

En las cadenas de texto se considera que el primer carácter ocupa la posición 0, el segundo la posición 1, etc. Con el método `substr` podemos extraer caracteres de una cadena, siendo el primer parámetro la posición desde donde se extraerán los caracteres y el segundo el número de caracteres a extraer.

En el nombre de las instancias de los botones, el número ocuparía la posición 5. Por ello, nos interesa extraer un carácter que se encuentra en la posición 5 del nombre. Para ello deberemos escribir `substr(5,1)`.

Nuestra función `loadImage` quedará por tanto como sigue:

```
function loadImage(e:MouseEvent):void
{
    main_ldr.source =
    "images/img"+e.target.name.substr(5,1)+".jpg";
}
```

Probamos de nuevo nuestra película para comprobar la carga de las diferentes imágenes según la miniatura que pulsemos.

Paso 11 de 14

Por último vamos a incluir una caja de texto en nuestra minigalería, que colocaremos en una nueva capa llamada `text`.

Para esta caja de texto utilizaremos otro componente llamado **TextArea**, que se encuentra dentro de la carpeta **User Interfaces** del panel **Componentes**.

Colocamos en el escenario una instancia de este componente. Le damos unas dimensiones de *AN:400* y *AL:100*, y lo situamos en la parte inferior izquierda del escenario con ayuda de las guías (*X:10* e *Y:320*).

Podemos ver los parámetros asociados a este componente en el área **Parámetros de Componente** del inspector de Propiedades.

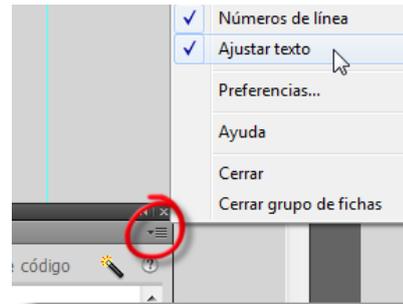
Para conocer la definición o funcionalidad de los distintos parámetros y otras propiedades o métodos asociados a ese componente, podemos buscar en la ayuda de flash la información sobre su clase asociada, cuyo nombre aparece en el área de vinculación del panel Biblioteca. En este caso la clase es [fl.controls.TextArea](#).

Damos a nuestra instancia del componente TextArea el nombre `textBox`.

Para añadir texto sin formato en nuestra caja de texto utilizaremos la propiedad `text`. Añadimos en el código la siguiente línea:

```
textBox.text="Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Donec vitae dui nulla,
et gravida justo.";
```

Para poder ver el texto con comodidad mientras programamos, sin necesidad de utilizar el scroll horizontal, seleccionamos **Ajustar texto** en las opciones del panel Acciones.



Paso 12 de 14

Probamos nuestra película. Aparecerá el texto con el formato predeterminado dentro de la caja de texto. No aparecerá ningún scroll, dado que en este caso no es necesario por la extensión del texto.

Probemos a añadir más texto a `textBox.text`, por ejemplo:

```
"Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Donec vitae dui nulla, et gravida justo.
```

```
Maecenas quam risus, suscipit ut congue ac, ultricies
eget odio. Etiam porta ultrices ante. Phasellus auctor
pulvinar auctor. Integer eget vestibulum sem.
```

```
Sed semper tortor vel justo pharetra volutpat. Ut
posuere arcu at felis convallis laoreet. In hac
habitasse platea dictumst."
```

Si copiamos este texto con saltos de línea incluidos y probamos nuestra película, el panel Errores de compilador mostrará un error que indica que las cadenas no pueden contener saltos de línea.

Podemos añadir retornos de carro dentro de un campo de texto mediante el carácter de retorno de carro `\r`, sustituyendo este carácter todos los saltos de línea que teníamos. De esta forma el texto quedaría como sigue:

```
"Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Donec vitae dui nulla, et gravida justo.\r
\rMaecenas quam risus, suscipit ut congue ac, ultricies
eget odio. Etiam porta ultrices ante. Phasellus auctor
pulvinar auctor. Integer eget vestibulum sem.\r\rSed
semper tortor vel justo pharetra volutpat. Ut posuere
arcu at felis convallis laoreet. In hac habitasse
platea dictumst."
```

Probamos de nuevo la película y comprobamos que ya no aparece el error anterior. Se habrá generado un scroll vertical de forma automática, ya que la extensión del texto es ahora mayor que el tamaño de la caja.

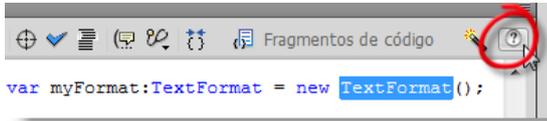
Paso 13 de 14

Vamos a formatear nuestra caja de texto mediante código. Para ello añadimos la línea:

```
var myFormat:TextFormat = new TextFormat();
```

En esta línea hemos creado una instancia de la clase `TextFormat` a la que hemos llamado `myFormat`

Para conocer las propiedades de la clase `TextFormat` podemos seleccionar su nombre en el panel Acciones y pulsar sobre el botón de ayuda del panel.



Especificamos la fuente, el color y el tamaño que aplicaremos a nuestro texto, por ejemplo:

```
myFormat.font = "Georgia";  
myFormat.color = "0x003366";  
myFormat.size = 15;
```

Por último, aplicamos este formato a nuestro texto:

```
textBox.setStyle("textFormat", myFormat);
```

Con esto ya habremos terminado nuestra minigalería de imágenes con un campo de texto.

Paso 14 de 14

Para complementar los conceptos desarrollados en este tutorial, se recomienda hacer las siguientes actividades:

1. Muestra en el campo de texto diferente contenido dependiendo de la imagen cargada.
2. Añade un componente `Label` para el título de cada imagen.

