

## Ejemplo: Filter Workbench

Filter Workbench proporciona una interfaz de usuario con la que es posible aplicar distintos filtros a imágenes y otro contenido visual, y ver el código resultante que se puede utilizar para generar el mismo efecto en ActionScript. Además de proporcionar una herramienta para experimentar con filtros, en esta aplicación se muestran las siguientes técnicas:

- Creación de instancias de diversos filtros
- Aplicación de varios filtros a un objeto de visualización

Para obtener los archivos de aplicación de este ejemplo, vaya a [www.adobe.com/go/learn\\_programmingAS3samples\\_flash\\_es](http://www.adobe.com/go/learn_programmingAS3samples_flash_es). Los archivos de aplicación de Filter Workbench se encuentran en la carpeta Samples/FilterWorkbench. La aplicación consta de los siguientes archivos:

Archivo	Descripción
com/example/programmingas3/filterWorkbench/FilterWorkbenchController.as	Clase que proporciona la funcionalidad principal de la aplicación, como el cambio del contenido al que se aplican los filtros y la aplicación de filtros al contenido.
com/example/programmingas3/filterWorkbench/IFilterFactory.as	Interfaz que define los métodos comunes que implementa cada una de las clases de fábrica de filtros. Esta interfaz define la funcionalidad común que la clase FilterWorkbenchController utiliza para interactuar con cada una de las clases de fábrica de filtros.

Archivo	Descripción
<p>en la carpeta com/example/ programmingas3/filterWorkbench/ BevelFactory.as BlurFactory.as ColorMatrixFactory.as ConvolutionFactory.as DropShadowFactory.as GlowFactory.as GradientBevelFactory.as GradientGlowFactory.as</p>	<p>Conjunto de clases, cada una de las cuales implementa la interfaz IFilterFactory. Cada una de estas clases proporciona la funcionalidad de crear y establecer valores para un solo tipo de filtro. Los paneles de propiedades de los filtros de la aplicación utilizan estas clases de fábrica para crear instancias de sus filtros particulares, que la clase FilterWorkbenchController recupera y aplica al contenido de la imagen.</p>
<p>com/example/programmingas3/ filterWorkbench/ColorStringFormatter.as</p>	<p>Clase de utilidad que incluye un método para convertir un valor de color numérico en un formato de cadena hexadecimal.</p>
<p>com/example/programmingas3/ filterWorkbench/GradientColor.as</p>	<p>Clase que sirve como objeto de valor y que combina en un solo objeto los tres valores (color, alfa y proporción) que se asocian a cada color en GradientBevelFilter y GradientGlowFilter.</p>
<h3>Interfaz de usuario (Flash)</h3>	
<p>FilterWorkbench fla</p>	<p>Archivo principal que define la interfaz de usuario de la aplicación.</p>
<p>flashapp/FilterWorkbench.as</p>	<p>Clase que proporciona la funcionalidad de la interfaz de usuario de la aplicación principal; esta clase se utiliza como la clase de documento del archivo FLA de la aplicación.</p>
<p>En la carpeta flashapp/filterPanels: BevelPanel.as BlurPanel.as ColorMatrixPanel.as ConvolutionPanel.as DropShadowPanel.as GlowPanel.as GradientBevelPanel.as GradientGlowPanel.as</p>	<p>Conjunto de clases que proporcionan la funcionalidad de cada panel que se utiliza para establecer las opciones de un solo filtro. En cada clase, hay además un símbolo MovieClip asociado en la biblioteca del archivo FLA de la aplicación principal, cuyo nombre coincide con el nombre de la clase (por ejemplo, el símbolo "BlurPanel" está vinculado a la clase definida en BlurPanel.as). Los componentes que constituyen la interfaz de usuario se sitúan y denominan en estos símbolos.</p>
<p>flashapp/ImageContainer.as</p>	<p>Objeto de visualización que sirve como contenedor de la imagen cargada en la pantalla.</p>
<p>flashapp/ButtonCellRender.er.as</p>	<p>Procesador de celdas predeterminado que se utiliza para incluir un componente Button en una celda del componente DataGrid.</p>

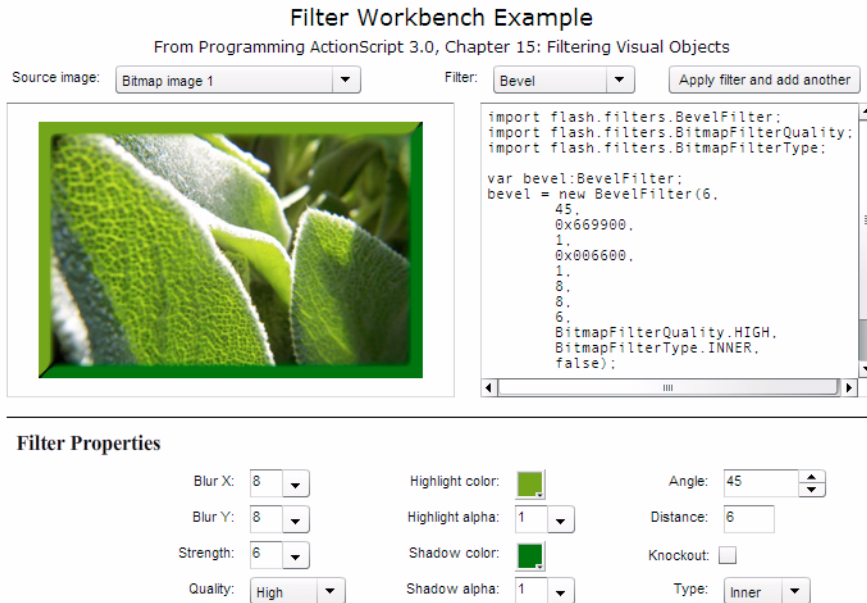
Archivo	Descripción
<b>Contenido de imagen filtrado</b>	
com/example/programmingas3/ filterWorkbench/ImageType.as	Esta clase sirve como objeto de valor que contiene el tipo y el URL de un solo archivo de imagen en el que la aplicación puede cargar y aplicar filtros. La clase también incluye un conjunto de constantes que representan los archivos de imágenes actuales disponibles.
images/sampleAnimation.swf, images/sampleImage1.jpg, images/sampleImage2.jpg	Imágenes y otro contenido visual a los que se aplican los filtros en la aplicación.

## Experimentación con filtros de ActionScript

La aplicación Filter Workbench se ha diseñado para permitir que el usuario experimente con diversos efectos de filtro y genere el código ActionScript apropiado para ese fin. La aplicación permite seleccionar tres archivos distintos con contenido visual, como imágenes de mapa de bits y una animación de Flash, y aplicar ocho filtros de ActionScript distintos a la imagen seleccionada, ya sea de forma individual o en combinación con otros filtros. La aplicación incluye los siguientes filtros:

- Bisel (`flash.filters.BevelFilter`)
- Desenfoque (`flash.filters.BlurFilter`)
- Matriz de colores (`flash.filters.ColorMatrixFilter`)
- Convolución (`flash.filters.ConvolutionFilter`)
- Sombra (`flash.filters.DropShadowFilter`)
- Iluminado (`flash.filters.GlowFilter`)
- Bisel degradado (`flash.filters.GradientBevelFilter`)
- Iluminado degradado (`flash.filters.GradientGlowFilter`)

Cuando un usuario ha seleccionado una imagen y un filtro para aplicarlo a dicha imagen, la aplicación muestra un panel con controles para establecer propiedades específicas del filtro seleccionado. Por ejemplo, en la siguiente imagen se muestra la aplicación con el filtro de bisel seleccionado:



A medida que el usuario ajusta las propiedades del filtro, la vista previa se actualiza en tiempo real. El usuario también puede aplicar varios filtros; para ello, debe personalizar un filtro, hacer clic en el botón Aplicar, personalizar otro filtro, hacer clic en el botón Aplicar y así sucesivamente.

Los paneles de filtros de la aplicación presentan algunas funciones y limitaciones:

- El filtro de matriz de colores incluye un conjunto de controles para manipular directamente las propiedades de imagen más comunes, como el brillo, el contraste, la saturación y el matiz. Además, se pueden especificar valores personalizados de la matriz de colores.
- El filtro de convolución, que sólo está disponible si se utiliza ActionScript, incluye un conjunto de valores de matriz de convolución de uso común y permite especificar valores personalizados. Sin embargo, si bien la clase ConvolutionFilter acepta una matriz de cualquier tamaño, la aplicación Filter Workbench utiliza una matriz fija de 3 x 3, que es el tamaño de filtro más utilizado.

- El filtro de mapa de desplazamiento, que sólo está disponible en ActionScript, no está disponible en la aplicación Filter Workbench. Un filtro de mapa de desplazamiento requiere una imagen del mapa además del contenido de imagen filtrado. La imagen del mapa es la principal entrada que determina el resultado del filtro, de modo que si no es posible cargar o crear una imagen del mapa, se reduce enormemente la posibilidad de experimentar con el filtro de mapa de desplazamiento.

## Creación de instancias de filtros

La aplicación Filter Workbench incluye un conjunto de clases, una por cada uno de los filtros disponibles, que utilizan los paneles individuales para crear los filtros. Cuando un usuario selecciona un filtro, el código ActionScript asociado al panel de filtros crea una instancia de la clase de fábrica de filtro correspondiente. (Estas clases se denominan *clases de fábrica* porque su propósito es crear instancias de otros objetos, de forma similar a una fábrica real que crea productos individuales.)

Cuando el usuario cambia un valor de propiedad en el panel, el código del panel llama al método apropiado en la clase de fábrica. Cada una de las clases de fábrica incluye métodos específicos que el panel utiliza para crear la instancia de filtro adecuada. Por ejemplo, si el usuario selecciona el filtro de desenfoque, la aplicación crea una instancia de BlurFactory. La clase BlurFactory incluye un método `modifyFilter()` que acepta tres parámetros: `blurX`, `blurY` y `quality`, que se utilizan conjuntamente para crear la instancia de BlurFilter deseada:

```
private var _filter:BlurFilter;

public function modifyFilter(blurX:Number = 4, blurY:Number = 4,
    quality:int = 1):void
{
    _filter = new BlurFilter(blurX, blurY, quality);
    dispatchEvent(new Event(Event.CHANGE));
}
```

Por otro lado, si el usuario selecciona el filtro de convolución, tendrá una mayor flexibilidad y podrá controlar un mayor número de propiedades. En la clase `ConvolutionFactory`, se llama al siguiente código cuando el usuario seleccione un valor distinto en el panel de filtros:

```
private var _filter:ConvolutionFilter;

public function modifyFilter(matrixX:Number = 0,
                             matrixY:Number = 0,
                             matrix:Array = null,
                             divisor:Number = 1.0,
                             bias:Number = 0.0,
                             preserveAlpha:Boolean = true,
                             clamp:Boolean = true,
                             color:uint = 0,
                             alpha:Number = 0.0):void
{
    _filter = new ConvolutionFilter(matrixX, matrixY, matrix, divisor, bias,
    preserveAlpha, clamp, color, alpha);
    dispatchEvent(new Event(Event.CHANGE));
}
```

Hay que tener en cuenta que, en cada caso, cuando se cambian los valores del filtro, el objeto de fábrica distribuye un evento `Event.CHANGE` para notificar a los detectores que los valores del filtro han cambiado. La clase `FilterWorkbenchController`, que es la que realmente aplica los filtros al contenido filtrado, detecta dicho evento para determinar cuándo necesita recuperar una nueva copia del filtro y volver a aplicarlo al contenido filtrado.

La clase `FilterWorkbenchController` no necesita tener detalles específicos de cada una de las clases de fábrica de filtros, sino que simplemente debe saber que el filtro ha cambiado y tener acceso a una copia del filtro. Para que esto sea posible, la aplicación incluye una interfaz, `IFilterFactory`, que define el comportamiento que debe ofrecer una clase de fábrica de filtro para que la instancia de `FilterWorkbenchController` de la aplicación pueda realizar su trabajo. La interfaz `IFilterFactory` define el método `getFilter()` que se utiliza en la clase `FilterWorkbenchController`:

```
function getFilter():BitmapFilter;
```

Hay que tener en cuenta que la definición del método de interfaz `getFilter()` especifica que devuelve una instancia de `BitmapFilter` en lugar de un tipo de filtro específico. La clase `BitmapFilter` no define un tipo de filtro específico. En lugar de eso, `BitmapFilter` es la clase base a partir de la cual se crean todas las clases de filtro. Cada clase de fábrica de filtro define una implementación específica del método `getFilter()` en el que devuelve una referencia al objeto de filtro que ha creado. Por ejemplo, a continuación se ofrece una versión abreviada del código fuente de la clase `ConvolutionFactory`:

```
public class ConvolutionFactory extends EventDispatcher implements
    IFilterFactory
{
    // ----- Variables privadas -----
    private var _filter:ConvolutionFilter;
    ...
    // ----- Implementación de IFilterFactory -----
    public function getFilter():BitmapFilter
    {
        return _filter;
    }
    ...
}
```

En la implementación de la clase `ConvolutionFactory` del método `getFilter()`, devuelve una instancia de `ConvolutionFilter`, aunque cualquier objeto que llame a `getFilter()` no tendrá por qué saber que, según la definición del método `getFilter()` que sigue `ConvolutionFactory`, debe devolver cualquier instancia de `BitmapFilter`, que puede ser una instancia de cualquiera de las clases de filtros de `ActionScript`.

## Aplicación de filtros a objetos de visualización

Tal y como se ha explicado en la sección anterior, la aplicación Filter Workbench utiliza una instancia de la clase FilterWorkbenchController (en lo sucesivo denominada “instancia de controlador”), que ejecuta la tarea real de aplicar filtros al objeto visual seleccionado. Para que la instancia de controlador pueda aplicar un filtro, necesita saber a qué imagen o contenido visual debe aplicarse el filtro. Cuando el usuario selecciona una imagen, la aplicación llama al método `setFilterTarget()` en la clase FilterWorkbenchController y pasa una de las constantes definidas en la clase ImageType:

```
public function setFilterTarget(targetType:ImageType):void
{
    ...
    _loader = new Loader();
    ...
    _loader.contentLoaderInfo.addEventListener(Event.COMPLETE,
        targetLoadComplete);
    ...
}
```

A partir de esta información, la instancia de controlador carga el archivo designado y, una vez cargado, lo almacena en una variable de instancia denominada `_currentTarget`:

```
private var _currentTarget:DisplayObject;

private function targetLoadComplete(event:Event):void
{
    ...
    _currentTarget = _loader.content;
    ...
}
```

Cuando el usuario selecciona un filtro, la aplicación llama al método `setFilter()` de la instancia de controlador, lo cual proporciona al controlador una referencia al objeto de fábrica de filtro relevante, que almacena en una variable de instancia denominada `_filterFactory`.

```
private var _filterFactory:IFilterFactory;

public function setFilter(factory:IFilterFactory):void
{
    ...

    _filterFactory = factory;
    _filterFactory.addEventListener(Event.CHANGE, filterChange);
}
```

Hay que tener en cuenta que, tal y como se ha descrito, la instancia de controlador no sabe cuál es el tipo de datos específico de la instancia de fábrica de filtro que se le asigna; sólo sabe que el objeto implementa la instancia `IFilterFactory`, lo que significa que tiene un método `getFilter()` y que distribuye un evento `change` (`Event.CHANGE`) cuando cambia el filtro.

Cuando el usuario cambia las propiedades de un filtro en el panel del filtro, la instancia de controlador detecta que el filtro ha cambiado a través del evento `change` de la fábrica del filtro, que llama al método `filterChange()` de la instancia de controlador. Dicho método, a su vez, llama al método `applyTemporaryFilter()`:

```
private function filterChange(event:Event):void
{
    applyTemporaryFilter();
}

private function applyTemporaryFilter():void
{
    var currentFilter:BitmapFilter = _filterFactory.getFilter();

    // Añadir el filtro actual al conjunto temporalmente
    _currentFilters.push(currentFilter);

    // Actualizar el conjunto de filtros del destino de filtro
    _currentTarget.filters = _currentFilters;

    // Quitar el filtro actual del conjunto
    // (no se elimina del destino de filtro, ya que
    // el destino usa una copia de la matriz de filtros internamente.)
    _currentFilters.pop();
}
```

La aplicación del filtro al objeto de visualización se produce en el método `applyTemporaryFilter()`. En primer lugar, el controlador recupera una referencia al objeto de filtro mediante una llamada al método `getFilter()` de la fábrica del filtro.

```
var currentFilter:BitmapFilter = _filterFactory.getFilter();
```

La instancia de controlador tiene una variable de instancia `Array` denominada `_currentFilters`, en la que almacena todos los filtros que se han aplicado al objeto de visualización. El siguiente paso es añadir el filtro recién actualizado a dicha matriz:

```
_currentFilters.push(currentFilter);
```

A continuación, el código asigna la matriz de filtros a la propiedad `filters` del objeto de visualización, que realmente aplica los filtros a la imagen:

```
_currentTarget.filters = _currentFilters;
```

Por último, dado que el filtro recién añadido sigue siendo un filtro “activo”, no debería aplicarse de forma permanente al objeto de visualización, de forma que se elimina de la matriz

```
_currentFilters:
```

```
_currentFilters.pop();
```

La eliminación de este filtro de la matriz no afecta al objeto de visualización filtrado, ya que un objeto de visualización realiza una copia de la matriz de filtros cuando se asigna a la propiedad `filters` y utiliza dicha matriz interna en lugar de la original. Por este motivo, cualquier cambio que se realice en la matriz de filtros no afectará al objeto de visualización hasta que se asigne nuevamente la matriz a la propiedad `filters` del objeto de visualización.