

Exemple : Filter Workbench

L'exemple Filter Workbench comporte une interface utilisateur qui permet d'appliquer divers filtres à des images et différents types de contenu visuel, et de voir le code résultant, qui peut être utilisé pour générer le même effet en code ActionScript. Cette application offre non seulement un outil permettant d'expérimenter des filtres, cette application démontre les techniques suivantes :

- Création d'occurrences de différents filtres
- Application de plusieurs filtres à un objet d'affichage

Pour obtenir les fichiers d'application pour cet exemple, voir www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers de l'application Filter Workbench se trouvent dans le dossier Samples/FilterWorkbench. L'application se compose des fichiers suivants :

Fichier	Description
com/example/programmingas3/ filterWorkbench/ FilterWorkbenchController.as	Classe fournissant la fonctionnalité principale de l'application, ce qui inclut la permutation du contenu auquel les filtres sont appliqués et l'application de filtres au contenu.
com/example/programmingas3/ filterWorkbench/IFilterFactory.as	Interface définissant des méthodes communes qui sont implémentées par les différentes classes de factorisation du filtre. Cette interface définit la fonctionnalité commune que la classe FilterWorkbenchController applique pour interagir avec les différentes classes de factorisation du filtre.

Fichier	Description
<p>dans le dossier com/example/ programmingas3/filterWorkbench/ :</p> <p>BevelFactory.as BlurFactory.as ColorMatrixFactory.as ConvolutionFactory.as DropShadowFactory.as GlowFactory.as GradientBevelFactory.as GradientGlowFactory.as</p>	<p>Ensemble de classes qui implémentent toutes l'interface IFilterFactory. Chacune de ces classes permet de créer et définir des valeurs pour un type de filtre unique. Les panneaux de propriétés des filtres de l'application ont recours à ces classes de factorisation pour créer des occurrences de leurs filtres, que la classe FilterWorkbenchController extrait et applique au contenu de l'image.</p>
<p>com/example/programmingas3/ filterWorkbench/ColorStringFormatter.as</p>	<p>Classe d'utilitaire qui inclut une méthode de conversion d'une valeur de couleur numérique au format String hexadécimal</p>
<p>com/example/programmingas3/ filterWorkbench/GradientColor.as</p>	<p>Classe qui sert d'objet de valeur, permettant de combiner dans un objet unique les trois valeurs (couleur, alpha et rapport) qui sont associées aux différentes couleurs de GradientBevelFilter et GradientGlowFilter</p>
Interface utilisateur (Flash)	
<p>FilterWorkbench fla</p>	<p>Le fichier principal qui définit l'interface utilisateur de l'application.</p>
<p>flashapp/FilterWorkbench.as</p>	<p>Classe qui fournit la fonctionnalité relative à l'interface utilisateur de l'application principale. Cette classe est utilisée en tant que classe de document pour le fichier FLA de l'application.</p>
<p>Dans le dossier flashapp/filterPanels :</p> <p>BevelPanel.as BlurPanel.as ColorMatrixPanel.as ConvolutionPanel.as DropShadowPanel.as GlowPanel.as GradientBevelPanel.as GradientGlowPanel.as</p>	<p>Ensemble de classes qui permet de définir un filtre unique directement à partir de l'un des panneaux utilisés. Pour chaque classe, il existe également un symbole MovieClip associé dans la bibliothèque du fichier FLA de l'application principale, dont le nom correspond au nom de la classe (par exemple, le symbole « BlurPanel » est lié à la classe définie dans BlurPanel.as). Les composants de l'interface utilisateur sont placés et nommés avec ces symboles.</p>
<p>flashapp/ImageContainer.as</p>	<p>Un objet d'affichage qui sert de conteneur pour l'image chargée à l'écran</p>
<p>flashapp/ButtonCellRenderer.as</p>	<p>Le rendu de cellules personnalisées utilisé pour inclure un composant bouton dans l'une des cellules du composant DataGrid</p>

Fichier	Description
Contenu de type image filtrée	
com/example/programmingas3/ filterWorkbench/ImageType.as	Cette classe sert d'objet de valeur contenant le type et l'URL d'un fichier d'image unique dans lequel l'application peut charger et appliquer des filtres. Cette classe inclut également un ensemble de constantes représentant les fichiers d'image réels disponibles.
images/sampleAnimation.swf, images/sampleImage1.jpg, images/sampleImage2.jpg	Images et autres contenus visuels auxquels des filtres sont appliqués dans l'application.

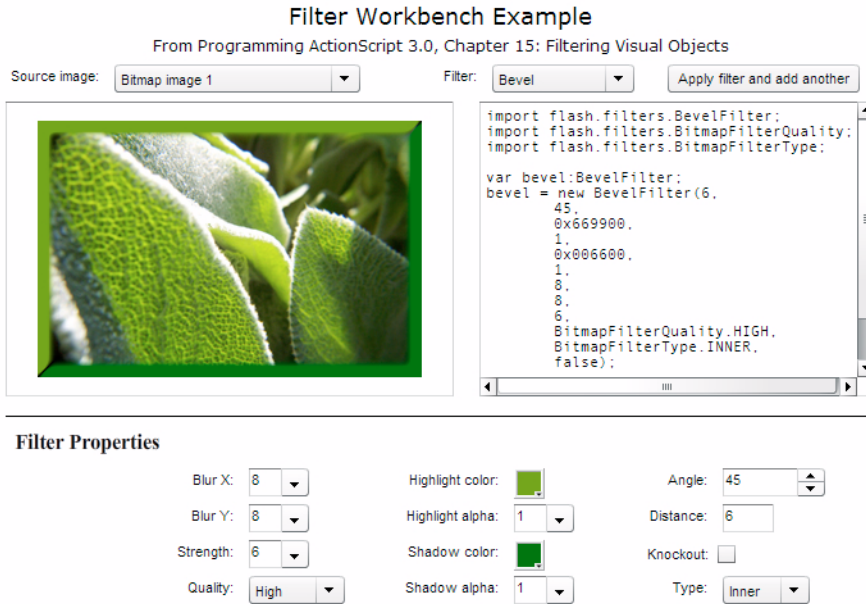
Expérimentation des filtres ActionScript

L'application Filter Workbench a été conçue pour vous permettre d'expérimenter les différents effets de filtre et générer le code ActionScript nécessaire. L'application permet d'effectuer des sélections à partir de différents types de fichier disposant de contenus visuels, ce qui inclut les images bitmap et une animation Flash, puis applique huit différents filtres ActionScript à l'image sélectionnée, de façon individuelle ou en combinaison avec d'autres filtres.

L'application inclut les filtres suivants :

- Biseau (`flash.filters.BevelFilter`)
- Flou (`flash.filters.BlurFilter`)
- Matrice de couleurs (`flash.filters.ColorMatrixFilter`)
- Convolution (`flash.filters.ConvolutionFilter`)
- Ombre portée (`flash.filters.DropShadowFilter`)
- Rayonnement (`flash.filters.GlowFilter`)
- Biseau dégradé (`flash.filters.GradientBevelFilter`)
- Rayonnement dégradé (`flash.filters.GradientGlowFilter`)

Lorsqu'un utilisateur a sélectionné une image et un filtre à appliquer à cette image, l'application affiche un panneau regroupant des contrôles permettant de définir les propriétés spécifiques du filtre sélectionné. Par exemple, l'image suivante affiche l'application avec le filtre Biseau sélectionné :



Lorsque l'utilisateur règle les propriétés du filtre, l'aperçu est mis à jour en temps réel. L'utilisateur peut également appliquer plusieurs filtres en personnalisant un autre, en cliquant sur le bouton Appliquer, ou personnaliser un autre filtre, en cliquant sur le bouton Appliquer, etc.

Les panneaux de filtre de l'application comportent un certain nombre de fonctionnalités et de limitations :

- Le filtre de matrice de couleurs inclut un ensemble de contrôles permettant d manipuler directement les propriétés d'image communes, ce qui inclut la luminosité, le contraste, la saturation et la teinte. En outre, il est possible de spécifier les valeurs de matrice de couleurs.

- Le filtre convolution, qui est disponible uniquement avec ActionScript, inclut un ensemble de valeurs de matrices de convolution utilisées de façon standard. Sinon, vous pouvez spécifier des valeurs personnalisées. Cependant, alors que la classe ConvolutionFilter accepte les matrices quelle que soit leur taille, l'application Filter Workbench utilise une matrice 3 x 3, la taille de filtre le plus utilisé.
- Le filtre de mappage du déplacement, qui est disponible uniquement dans ActionScript, n'est pas disponible dans l'application Filter Workbench. Un filtre de mappage du déplacement nécessite une carte image en supplément du contenu d'image filtré. La carte image est la principale entrée qui détermine le résultat du filtre, par conséquent, sans la possibilité de charger ou créer une carte image, la possibilité d'expérimenter le filtre de mappage du déplacement serait très limitée.

Création d'occurrences de filtre

L'application Filter Workbench inclut un ensemble de classes, un pour chacun des filtres disponibles, qui sont utilisés par les différents panneaux pour créer les filtres. Lorsqu'un utilisateur sélectionne un filtre, le code ActionScript associé au panneau de filtre crée une occurrence de la classe factorielle de filtres requise. (Ces classes sont appelées *classes factorielles* dans la mesure où elles ont pour objet de créer des occurrences d'autres objets.)

Lorsque l'utilisateur modifie la valeur de la propriété sur le panneau, le code du panneau appelle la méthode requise dans la classe factorielle. Toute classe factorielle inclut des méthodes spécifiques que le panneau utilise pour créer l'occurrence de filtre requise. Par exemple, si l'utilisateur sélectionne le filtre Flou, l'application crée une occurrence BlurFactory. La classe BlurFactory inclut la méthode `modifyFilter()` qui accepte trois paramètres : `blurX`, `blurY` et `quality`, qui sont utilisés conjointement pour créer l'occurrence BlurFilter voulue :

```
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));
}
```

D'autre part, si l'utilisateur sélectionne le filtre Convolution, ce filtre autorise une plus grande souplesse et par conséquent doit contrôler davantage de propriétés. Dans la classe ConvolutionFactory, le code suivant est appelé lorsque l'utilisateur sélectionne une autre valeur dans le panneau de filtres :

```
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));
}
```

Vous pouvez constater que dans chaque cas, lorsque les valeurs de filtre sont modifiées, l'objet factoriel distribue un événement `Event.CHANGE` pour prévenir les écouteurs que les valeurs de filtre ont changé. La classe `FilterWorkbenchController`, qui procède à l'application des filtres au contenu, écoute cet événement pour déterminer s'il est nécessaire d'extraire une nouvelle copie du filtre et l'appliquer de nouveau au contenu filtré.

La classe `FilterWorkbenchController` ne nécessite pas de détails spécifiques sur chaque classe factorielle de filtre. Elle doit seulement savoir si le filtre a changé et avoir accès à une copie du filtre. Pour prendre ceci en charge, l'application inclut une interface, `IFilterFactory`, qui définit le comportement qu'une classe factorielle de filtres doit présenter de façon à ce que l'occurrence `FilterWorkbenchController` de l'application puisse exécuter ses tâches. L'interface `IFilterFactory` définit la méthode `getFilter()` qui est utilisée dans la classe `FilterWorkbenchController` :

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

Vous pouvez constater que la définition de méthode d'interface `getFilter()` spécifie qu'elle renvoie une occurrence `BitmapFilter` et non pas un type spécifique de filtre. La classe `BitmapFilter` ne définit pas de type spécifique de filtre. Par contre, `BitmapFilter` constitue la classe de base sur laquelle toutes les classes de filtre sont créées. Toute classe factorielle de filtre définit une implémentation spécifique de la méthode `getFilter()` dans laquelle elle renvoie une référence à l'objet de filtre qu'elle a créé. Par exemple, voici une version abrégée du code source de la classe `ConvolutionFactory` :

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

Dans l'implémentation de la classe `ConvolutionFactory` de la méthode `getFilter()`, elle renvoie une occurrence `ConvolutionFilter`, bien que tout objet qui appelle `getFilter()` ne soit pas forcément informé. Conformément à la définition de la méthode `getFilter()` appliquée par `ConvolutionFactory`, elle doit renvoyer l'occurrence `BitmapFilter`, qui pourrait être une occurrence de l'une des classes de filtre `ActionScript`.

Application de filtres à des objets d'affichage

Comme expliqué dans la section précédente, l'application Filter Workbench a recours à une occurrence de la classe `FilterWorkbenchController` (appelée ici « occurrence de contrôleur »), qui procède à l'application des filtres à l'objet visuel sélectionné. Avant que l'occurrence de contrôleur puisse appliquer un filtre, elle doit connaître l'image ou le contenu visuel auquel ce filtre doit s'appliquer. Lorsque l'utilisateur sélectionne une image, l'application appelle la méthode `setFilterTarget()` dans la classe `FilterWorkbenchController`, en transmettant l'une des constantes définies dans la classe `ImageType` :

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

Ces informations permettent à l'occurrence de contrôleur de charger le fichier voulu et de le stocker dans une variable d'occurrence appelée `_currentTarget` lors de son chargement :

```
private var _currentTarget:DisplayObject;

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

Lorsqu'un utilisateur sélectionne un filtre, l'application appelle la méthode `setFilter()` de l'occurrence de contrôleur, ce qui donne une référence au contrôleur pour l'objet factoriel de filtre voulu, qu'il stocke dans une variable d'occurrence appelée `_filterFactory`.

```
private var _filterFactory:IFilterFactory;

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

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

Comme indiqué précédemment, l'occurrence de contrôleur ne connaît pas le type de données de l'occurrence factorielle de filtre qu'elle a reçu. Elle sait uniquement que l'objet implémente l'occurrence `IFilterFactory`, ce qui signifie qu'elle a une méthode `getFilter()` et qu'elle distribue un événement `change` (`Event.CHANGE`) lorsque le filtre change.

Lorsque l'utilisateur modifie les propriétés d'un filtre dans le panneau de ce filtre, l'occurrence de contrôleur est informée de cette modification par l'intermédiaire de l'événement `change` du filtre factoriel, qui appelle la méthode `filterChange()` de l'occurrence de contrôleur.

Cette méthode, en retour, appelle la méthode `applyTemporaryFilter()` :

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

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

    // Ajoute de façon temporaire le filtre actuel à l'ensemble
    _currentFilters.push(currentFilter);

    // Actualise l'ensemble de filtres de la cible du filtre
    _currentTarget.filters = _currentFilters;

    // Supprime le filtre actuel de l'ensemble
    // (Ceci ne supprime pas le filtre de la cible, dans la mesure
    // où la cible utilise une copie du tableau de filtres en interne.)
    _currentFilters.pop();
}
```

Le filtre est appliqué à l'objet d'affichage dans la méthode `applyTemporaryFilter()`. Tout d'abord, le contrôleur extrait une référence de l'objet filtre en appelant la méthode `getFilter()` du filtre factoriel.

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

L'occurrence de contrôleur comporte la variable d'occurrence `Array`, appelée `_currentFilters`, qui stocke tous les filtres qui ont été appliqués à l'objet d'affichage. L'étape suivante consiste à ajouter le filtre qui vient d'être mis à jour à ce tableau :

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

Ensuite, le code affecte le tableau de filtres à la propriété `filters` de l'objet d'affichage, qui applique les filtres à l'image :

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

Enfin, dans la mesure où ce filtre, qui vient d'être ajouté, reste fonctionnel, il ne doit pas être appliqué de façon permanente à l'objet d'affichage. Il faut donc le supprimer du tableau

```
_currentFilters :
```

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

La suppression de ce filtre du tableau n'affecte pas l'objet d'affichage filtré, dans la mesure où un objet d'affichage crée une copie du tableau de filtres lorsqu'il est affecté à la propriété `filters`. Il utilise ce tableau interne de préférence au tableau d'origine. Par conséquent, toute modification apportée au tableau de filtres n'affecte pas l'objet d'affichage tant que le tableau n'a pas été affecté de nouveau à la propriété `filters` de l'objet d'affichage.