

## 範例：Filter Workbench

「Filter Workbench」所提供的使用者介面，可將不同的濾鏡套用至影像與視覺內容，並檢視產生的程式碼，以便在 ActionScript 中產生相同的效果。除了提供試驗濾鏡的工具之外，此應用程式也提供下列技術：

- 建立各種濾鏡的實體
- 將多個濾鏡套用至顯示物件

若要取得此樣本的應用程式檔案，請參閱 [www.adobe.com/go/learn\\_programmingAS3samples\\_flash\\_tw](http://www.adobe.com/go/learn_programmingAS3samples_flash_tw)。您可以在 Samples/FilterWorkbench 資料夾中找到「Filter Workbench」應用程式檔案。此應用程式是由下列檔案組成：

檔案	說明
com/example/programmingas3/filterWorkbench/FilterWorkbenchController.as	提供應用程式主功能的類別，包括切換到套用濾鏡的內容，以及套用濾鏡到內容。
com/example/programmingas3/filterWorkbench/IFilterFactory.as	此介面定義由每個 filter factory 類別所實作的常用方法，並定義 FilterWorkbenchController 類別用來與個別的 filter factory 類別互動的常見功能。
在 com/example/programmingas3/filterWorkbench/ 資料夾中： BevelFactory.as BlurFactory.as ColorMatrixFactory.as ConvolutionFactory.as DropShadowFactory.as GlowFactory.as GradientBevelFactory.as GradientGlowFactory.as	一組類別，用於實作 IFilterFactory 介面。這些類別都提供建立及設定濾鏡的單一類型值之功能。應用程式內的濾鏡屬性面板會使用這些 factory 類別來建立其特定的濾鏡實體，這些濾鏡實體會由 FilterWorkbenchController 類別進行擷取並套用至影像內容。
com/example/programmingas3/filterWorkbench/ColorStringFormatter.as	Utility 類別，內含將數字顏色值轉換為十六進制字串格式的方法。

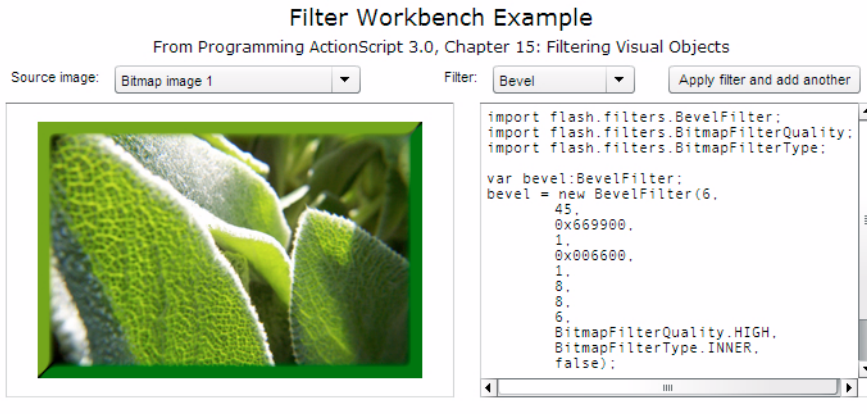
檔案	說明
com/example/programmingas3/ filterWorkbench/GradientColor.as	擔任 value 物件的類別，可將與 GradientBevelFilter 及 GradientGlowFilter 內的每個顏色關聯的三個值 (color、alpha 和 ratio) 結合成單一物件
<b>使用者介面 (Flash)</b>	
FilterWorkbench.fla	主要檔案，定義應用程式的使用者介面。
flashapp/FilterWorkbench.as	用來提供主應用程式的使用者介面功能之類別；此類別會在應用程式的 FLA 檔案當做文件類別來使用。
在 flashapp/filterPanels 資料夾中： BevelPanel.as BlurPanel.as ColorMatrixPanel.as ConvolutionPanel.as DropShadowPanel.as GlowPanel.as GradientBevelPanel.as GradientGlowPanel.as	一組類別，提供每個面板的功能，以設定單一濾鏡的選項。 每個類別中也內含一個關聯的 MovieClip 元件，該元件位於主應用程式 FLA 檔案的元件庫中，其名稱與類別的名稱相符 (例如，元件 "BlurPanel" 會連結至定義於 BlurPanel.as 中的類別)。構成使用者介面的組件將會定位並命名於元件內。
flashapp/ImageContainer.as	在螢幕上擔任已載入影像的容器之顯示物件。
flashapp/ButtonCellRenderer.as	自訂的儲存格輸出器，用來在 DataGrid 組件內的某個儲存格中嵌入 button 組件。
<b>已套用濾鏡的影像內容</b>	
com/example/programmingas3/ filterWorkbench/ImageType.as	此類別可作為 value 物件使用，內含單一影像檔的類型和 URL，應用程式可以在該影像檔上載入並套用濾鏡。此類別也包含一組常數，用於表示實際可用的影像檔。
images/sampleAnimation.swf、 images/sampleImage1.jpg、 images/sampleImage2.jpg	影像和其他的視覺內容，在應用程式中已套用濾鏡。

## 以 ActionScript 濾鏡進行試驗

「Filter Workbench」應用程式可用來協助您試驗各種濾鏡效果，並為該效果產生相關 ActionScript 程式碼。此應用程式可讓您從三種內含視覺內容的不同檔案中進行選取，包括點陣圖影像和 Flash 動畫，以及對選取的影像套用八種不同的 ActionScript 濾鏡，您可以個別套用或與其他濾鏡組合套用。此應用程式包含下列濾鏡：

- 斜角 (flash.filters.BevelFilter)
- 模糊 (flash.filters.BlurFilter)
- 顏色矩陣 (flash.filters.ColorMatrixFilter)
- 迴旋 (flash.filters.ConvolutionFilter)
- 投影 (flash.filters.DropShadowFilter)
- 光暈 (flash.filters.GlowFilter)
- 漸層斜角 (flash.filters.GradientBevelFilter)
- 漸層光暈 (flash.filters.GradientGlowFilter)

只要使用者選取了某個影像和要套用到該影像的濾鏡，應用程式就會顯示一個面板，內含設定所選濾鏡的特定屬性之控制項。例如，下列影像顯示了已選取「斜角」濾鏡的應用程式：



### Filter Properties

Blur X:	<input type="text" value="8"/>	Highlight color:	<input type="color" value="#669900"/>	Angle:	<input type="text" value="45"/>
Blur Y:	<input type="text" value="8"/>	Highlight alpha:	<input type="text" value="1"/>	Distance:	<input type="text" value="6"/>
Strength:	<input type="text" value="6"/>	Shadow color:	<input type="color" value="#006600"/>	Knockout:	<input type="checkbox"/>
Quality:	<input type="text" value="High"/>	Shadow alpha:	<input type="text" value="1"/>	Type:	<input type="text" value="Inner"/>

隨著使用者調整濾鏡的屬性，預覽內容便會即時更新。使用者也可以藉由自訂某個濾鏡、按一下「套用」按鈕、自訂另一個濾鏡、按一下「套用」按鈕等步驟，套用多個濾鏡。

應用程式的濾鏡面板有幾項功能和限制：

- 顏色矩陣濾鏡內含一組控制項，可直接操作一般影像的屬性，包括亮度、對比、飽和度和色相。此外，也可以指定自訂的顏色矩陣值。
- 使用 ActionScript 時才提供的迴旋濾鏡包含了一組常用的迴旋矩陣值，或者可以指定自訂值。不過，雖然 ConvolutionFilter 類別可接受任何大小的矩陣，「Filter Workbench」應用程式使用的是固定的 3 x 3 矩陣，這也是最常用的濾鏡大小。
- 置換對應濾鏡只可以在 ActionScript 內使用，無法在「Filter Workbench」應用程式內使用。除了對應影像之外，置換對應濾鏡也需要使用已套用濾鏡影像的內容。由於對應影像為決定濾鏡結果的主要輸入，因此如果無法載入或建立對應影像，則以置換對應濾鏡進行試驗的能力就會受到極大的限制。

## 建立濾鏡實體

「Filter Workbench」應用程式內含一組類別，每個可用的濾鏡都有一個類別，可由個別的面板用來建立濾鏡。當使用者選取濾鏡時，與濾鏡面板關聯的 ActionScript 程式碼會建立適當 filter factory 類別的實體（這些類別稱為「factory 類別」，由於其目的是要建立其它物件的實體，因此就像是在現實世界中製造個別產品的工廠）。

每當使用者變更面板上的屬性值，面板的程式碼便會呼叫 factory 類別內適當的方法。每個 factory 類別都內含特定的方法，可讓面板用來建立適當的濾鏡實體。例如，如果使用者選取了「模糊」濾鏡，應用程式便會建立一個 BlurFactory 實體。BlurFactory 類別內含一個 modifyFilter() 方法，可接受三個參數：blurX、blurY 和 quality，同時使用可用來建立所需的 BlurFilter 實體：

```
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));
}
```

另一方面，如果使用者選取了「迴旋」濾鏡，由於該濾鏡提供了更多彈性，便可控制較多屬性集。在 ConvolutionFactory 類別中，下列程式碼會在使用者於濾鏡面板上選取不同值時呼叫：

```
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));
    }
}

```

請注意，在各種情況下，變更濾鏡值時，`factory` 物件便會傳送一個 `Event.CHANGE` 事件，以通知偵聽程式該濾鏡的值已變更。`FilterWorkbenchController` 類別會實際執行將濾鏡套用至已套用濾鏡之內容的工作，並偵聽該事件，以查明何時需要擷取新的濾鏡副本，並重新將濾鏡套用至已套用濾鏡的內容上。

`FilterWorkbenchController` 類別不需要知道每個 `filter factory` 類別的特殊細節 — 它只需要知道濾鏡已變更，並且能夠存取濾鏡的副本即可。為了支援此動作，應用程式包含了一個介面 (即 `IFilterFactory`)，該介面會定義 `filter factory` 類別需要提供的行爲，因此應用程式的 `FilterWorkbenchController` 實體便可以執行其工作。`IFilterFactory` 定義了 `getFilter()` 方法，可在 `FilterWorkbenchController` 類別內使用：

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

請注意，`getFilter()` 介面方法定義會指定其傳回 `BitmapFilter` 實體，而非特定的濾鏡類型。`BitmapFilter` 類別不會定義特定的濾鏡類型。因此，`BitmapFilter` 是建立所有濾鏡類別的基底類別。每個 `filter factory` 類別都會定義一個特定的 `getFilter()` 方法實作，將所建立的濾鏡物件參考傳回。例如，以下是 `ConvolutionFactory` 類別的縮寫版原始碼：

```

public class ConvolutionFactory extends EventDispatcher implements
    IFilterFactory
{
    // ----- 私有變數 -----
    private var _filter:ConvolutionFilter;
    ...
    // ----- IFilterFactory 實作 -----
    public function getFilter():BitmapFilter
    {
        return _filter;
    }
    ...
}

```

`ConvolutionFactory` 類別實作 `getFilter()` 方法時，會傳回一個 `ConvolutionFilter` 實體，即使任何呼叫 `getFilter()` 的物件都不需要知道這件事 — 依照 `ConvolutionFactory` 所遵守的 `getFilter()` 方法之定義來看，它必須傳回任何 `BitmapFilter` 實體，該實體可以是 `ActionScript` 的任何濾鏡類別。

## 將濾鏡套用至顯示物件

如先前章節所說明，「Filter Workbench」應用程式會使用到 `FilterWorkbenchController` 類別的實體（之後稱為「控制器實體」），該實體實際執行將濾鏡套用至所選視覺物件的工作。在控制器實體可以套用濾鏡之前，必須先得知應該將濾鏡套用到哪個影像或視覺內容。使用者選取某個影像時，應用程式會呼叫 `FilterWorkbenchController` 類別內的 `setFilterTarget()` 方法，並傳遞定義於 `ImageType` 類別內的其中一個常數至該方法：

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

控制器實體會使用該資訊來載入指定的檔案，並在載入時將其儲存在名為 `_currentTarget` 的實體變數中：

```
private var _currentTarget:DisplayObject;

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

當使用者選取了某個濾鏡時，應用程式會呼叫控制實體的 `setFilter()` 方法，為控制器提供相關 `filter factory` 物件的參考，並將之儲存在一個名為 `_filterFactory` 的實體變數中。

```
private var _filterFactory:IFilterFactory;

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

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

請注意，如先前的內容所提及，控制器實體不知道所提供 `filter factory` 實體的特定資料類型；它只知道該物件實作了 `IFilterFactory` 實體，表示其擁有一個 `getFilter()` 方法，並且會在濾鏡變更時傳送 `change (Event.CHANGE)` 事件。

當使用者在濾鏡的面板內變更濾鏡的屬性時，控制器實體會發現該濾鏡已透過 `filter factory` 的 `change` 事件改變，而該事件會呼叫控制器實體的 `filterChange()` 方法。此時，控制實體的方法便會呼叫 `applyTemporaryFilter()` 方法：

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

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

    // 將目前的濾鏡暫時加入組合
    _currentFilters.push(currentFilter);

    // 更新濾鏡目標的濾鏡組合
    _currentTarget.filters = _currentFilters;

    // 從組合中移除目前的濾鏡
    // (如此做不會將其從濾鏡目標移除，因為
    // 目標使用的是濾鏡陣列內部的副本)。
    _currentFilters.pop();
}
```

將濾鏡套用至顯示物件的工作會在 `applyTemporaryFilter()` 方法內發生。首先，控制器會藉由呼叫 `filter factory` 的 `getFilter()` 方法來擷取對濾鏡物件的參考。

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

控制器實體擁有一個名為 `_currentFilters` 的 `Array` 實體變數，當中會儲存所有已套用至顯示物件的濾鏡。下一個步驟是要將最近更新的濾鏡加入該陣列：

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

接下來，程式碼會將濾鏡的陣列指定給顯示物件的 `filters` 屬性，以將濾鏡實際套用到影像：

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

最後，由於這個最近加入的濾鏡仍是一個「作用中」的濾鏡，不該將其永久套用至顯示物件，因此要將該濾鏡從 `_currentFilters` 陣列中移除：

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

將這個濾鏡從陣列中移除並不會影響已套用濾鏡的顯示物件，因為顯示物件已在將濾鏡陣列指定給 `filters` 屬性時對其製作副本，並且會使用該內部陣列，而不是原始的陣列。基於這個原因，任何對濾鏡陣列的變更都不會影響顯示物件，除非您再次將陣列指定給顯示物件的 `filters` 屬性。

