

예제: 필터 워크벤치

필터 워크벤치에서 제공하는 사용자 인터페이스를 통해 이미지 및 기타 시각적 내용에 다양한 필터를 적용해 보고 `ActionScript`에서 동일한 효과를 내는 결과 코드를 확인할 수 있습니다. 이 응용 프로그램은 필터를 시험해 볼 수 있는 도구를 제공할 뿐만 아니라 다음과 같은 작업을 수행하는 방법도 보여 줍니다.

- 다양한 필터 인스턴스 만들기
- 표시 객체에 여러 개의 필터 적용

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. 필터 워크벤치 응용 프로그램 파일은 `Samples/FilterWorkbench` 폴더에 있으며 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
<code>com/example/programmingas3/filterWorkbench/FilterWorkbenchController.as</code>	필터가 적용되는 내용을 전환하고 내용에 필터를 적용하는 등 응용 프로그램의 주요 기능을 제공하는 클래스입니다.
<code>com/example/programmingas3/filterWorkbench/IFilterFactory.as</code>	각 필터 팩토리 클래스를 통해 구현되는 공통 메서드를 정의하는 인터페이스입니다. 이 인터페이스는 <code>FilterWorkbenchController</code> 클래스가 개별 필터 팩토리 클래스와 상호 작용하기 위해 사용하는 공통 기능을 정의합니다.

파일	설명
com/example/programmingas3/ filterWorkbench/ 폴더에 있는 파일: BevelFactory.as BlurFactory.as ColorMatrixFactory.as ConvolutionFactory.as DropShadowFactory.as GlowFactory.as GradientBevelFactory.as GradientGlowFactory.as	IFilterFactory 인터페이스를 구현하는 각 클래스의 집합입니다. 각 클래스는 단일 필터 유형의 값을 만들고 설정하는 기능을 제공합니다. 응용 프로그램의 필터 속성 패널이 이러한 팩토리 클래스를 사용하여 특정 필터의 인스턴스를 만들면 FilterWorkbenchController 클래스는 이러한 인스턴스를 검색하여 이미지 내용에 적용합니다.
com/example/programmingas3/ filterWorkbench/ ColorStringFormatter.as	숫자 색상 값을 16진수 문자열 형식으로 변환하는 메서드를 포함하는 유틸리티 클래스입니다.
com/example/programmingas3/ filterWorkbench/GradientColor.as	GradientBevelFilter 및 GradientGlowFilter의 각 색상과 연결된 세 가지 값(색상, 알파 및 비율)을 단일 객체로 결합하는 값 객체 역할을 하는 클래스입니다.
사용자 인터페이스(Flash)	
FilterWorkbench fla	응용 프로그램의 사용자 인터페이스를 정의하는 기본 파일입니다.
flashapp/FilterWorkbench.as	기본 응용 프로그램의 사용자 인터페이스에 대한 기능을 제공하는 클래스입니다. 이 클래스는 응용 프로그램 FLA 파일의 문서 클래스로 사용됩니다.
flashapp/filterPanels 폴더에 있는 파일: BevelPanel.as BlurPanel.as ColorMatrixPanel.as ConvolutionPanel.as DropShadowPanel.as GlowPanel.as GradientBevelPanel.as GradientGlowPanel.as	단일 필터에 대한 옵션을 설정하는 데 사용되는 각 패널의 기능을 제공하는 클래스 집합입니다. 기본 응용 프로그램 FLA 파일의 라이브러리에는 각 클래스의 이름과 일치하는 연결된 MovieClip 심볼이 있습니다. 예를 들어 “BlurPanel” 심볼은 BlurPanel.as에 정의된 클래스와 링크되어 있습니다. 사용자 인터페이스를 구성하는 구성 요소는 이러한 심볼 내부에 위치하고 이름이 지정됩니다.
flashapp/ImageContainer.as	스크린에 로드된 이미지의 컨테이너 역할을 하는 표시 객체입니다.
flashapp/ButtonCellRenderer.as	DataGrid 구성 요소의 셀에 Button 구성 요소를 포함하기 위해 사용되는 사용자 정의 셀 렌더러입니다.

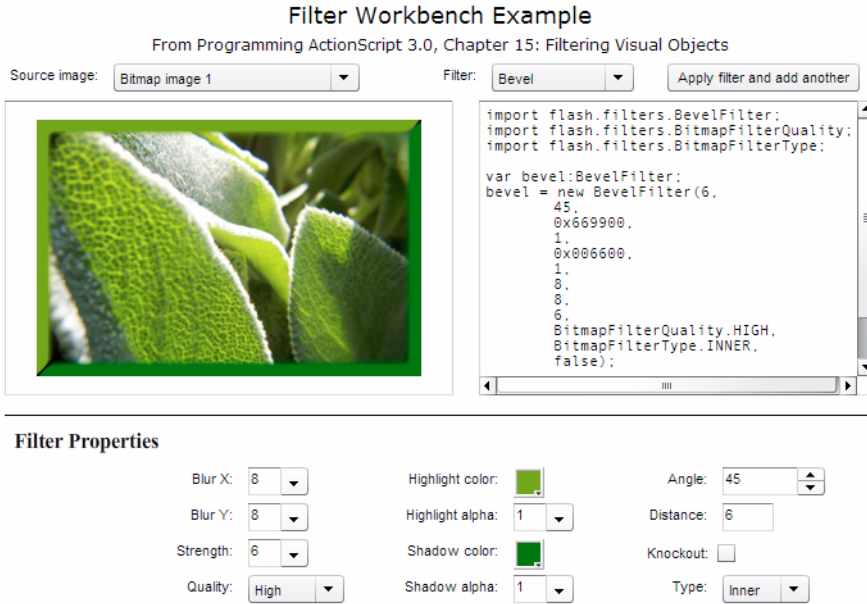
파일	설명
필터링된 이미지 내용 com/example/programmingas3/ filterWorkbench/ImageType.as	응용 프로그램에서 로드하여 필터를 적용할 수 있는 단일 이미지 파일의 유형 및 URL을 포함하는 값 객체 역할을 하는 클래스입니다. 이 클래스에는 사용할 수 있는 실제 이미지 파일을 나타내는 상수 집합도 포함되어 있습니다.
images/sampleAnimation.swf, images/sampleImage1.jpg, images/sampleImage2.jpg	응용 프로그램에서 필터가 적용되는 이미지 및 기타 시각적 내용입니다.

ActionScript 필터 시험하기

필터 워크벤치 응용 프로그램은 사용자가 다양한 필터 효과를 시험해 보고 이런 효과에 대해 관련 ActionScript 코드를 생성할 수 있도록 설계되었습니다. 이 응용 프로그램을 사용하면 비트맵 이미지 및 Flash 애니메이션 등의 시각적 내용을 포함하는 세 가지 다른 파일을 선택할 수 있으며 선택한 이미지에 8개의 다른 ActionScript 필터를 개별적으로 적용하거나 다른 필터와 조합하여 적용할 수 있습니다. 응용 프로그램에는 다음 필터가 포함되어 있습니다.

- 경사(flash.filters.BevelFilter)
- 흐림(flash.filters.BlurFilter)
- 색상 매트릭스(flash.filters.ColorMatrixFilter)
- 회선(flash.filters.ConvolutionFilter)
- 그림자(flash.filters.DropShadowFilter)
- 광선(flash.filters.GlowFilter)
- 그래디언트 경사(flash.filters.GradientBevelFilter)
- 그래디언트 광선(flash.filters.GradientGlowFilter)

사용자가 이미지와 이 이미지에 적용할 필터를 선택하면 응용 프로그램은 선택한 필터의 특정 속성을 설정하는 컨트롤이 있는 패널을 표시합니다. 예를 들어, 다음 이미지는 [경사] 필터가 선택된 응용 프로그램을 보여 줍니다.



사용자가 필터 속성을 조정하면 미리 보기가 실시간으로 업데이트됩니다. 사용자는 한 필터를 사용자 정의하고 [적용] 버튼을 클릭한 다음 다른 필터를 사용자 정의하고 [적용] 버튼을 클릭하는 방식으로 여러 개의 필터를 적용할 수도 있습니다.

응용 프로그램의 필터 패널의 몇 가지 기능과 제한 사항은 다음과 같습니다.

- 색상 매트릭스 필터에는 밝기, 대비, 채도 및 색조를 포함하는 공통 이미지 속성을 직접 조작할 수 있는 컨트롤 집합이 포함되어 있습니다. 또한 사용자 정의 매트릭스 값을 지정할 수 있습니다.
- ActionScript를 통해서만 사용할 수 있는 회전 필터에는 주로 사용되는 회전 매트릭스 값 집합이 포함되어 있지만 사용자 정의 값을 지정할 수도 있습니다. ConvolutionFilter 클래스에는 모든 크기의 매트릭스를 사용할 수 있지만 필터 워크벤치 응용 프로그램은 가장 일반적으로 사용되는 고정된 3 x 3 매트릭스를 사용합니다.
- ActionScript에서만 사용할 수 있는 위치 변경 맵 필터는 필터 워크벤치 응용 프로그램에서 사용할 수 없습니다. 위치 변경 맵 필터에는 필터링된 이미지 내용 외에 맵 이미지가 있어야 합니다. 맵 이미지는 필터의 결과를 정의하는 기본 입력이므로 맵 이미지를 로드하거나 만드는 기능이 없으면 위치 변경 맵 필터를 시험하는 기능은 극히 제한됩니다.

필터 인스턴스 만들기

필터 워크벤치 응용 프로그램에는 사용할 수 있는 필터별로 하나씩 지정된 클래스 집합이 포함되어 있는데 개별 패널은 이러한 클래스를 사용하여 필터를 만듭니다. 사용자가 필터를 선택하면 필터 패널과 연결된 **ActionScript** 코드에서는 해당하는 필터 팩토리 클래스의 인스턴스를 만듭니다. (이러한 클래스는 실제 팩토리에서 개별 제품을 만드는 것처럼 다른 객체의 인스턴스를 만들므로 팩토리 클래스라고 불립니다.)

사용자가 패널에서 속성 값을 변경할 때마다 패널의 코드는 팩토리 클래스의 해당하는 메시지를 호출합니다. 각 팩토리 클래스에는 패널이 적절한 필터 인스턴스를 만들 때 사용하는 특정 메시지가 포함되어 있습니다. 예를 들어 사용자가 [흐림] 필터를 선택하는 경우 응용 프로그램은 **BlurFactory** 인스턴스를 만듭니다. **BlurFactory** 클래스에는 `blurX`, `blurY` 및 `quality` 라는 세 개의 매개 변수를 사용하는 `modifyFilter()` 메시지가 있는데 이 매개 변수는 원하는 **BlurFilter** 인스턴스를 만드는 데 함께 사용됩니다.

```
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));
}
```

반면 사용자가 유연성이 훨씬 더 높은 [회선] 필터를 선택하는 경우 제어할 수 있는 속성 집합은 더 커지게 됩니다. 사용자가 필터 패널에서 다른 값을 선택하면 **ConvolutionFactory** 클래스에서 다음 코드가 호출됩니다.

```
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));
}
```

각각의 경우에 필터 값이 변경되면 팩토리 객체는 `Event.CHANGE` 이벤트를 전달하여 리스너에게 필터 값이 변경되었음을 알립니다. 필터링된 내용에 필터를 실제로 적용하는 작업을 수행하는 `FilterWorkbenchController` 클래스는 필터의 새 복사본을 검색하여 필터링된 내용에 다시 적용해야 할 시기를 확인하기 위해 이벤트를 수신 대기합니다.

`FilterWorkbenchController` 클래스는 각 필터 팩토리 클래스의 자세한 내용을 알 필요가 없으며 단지 필터가 변경되었음을 알고 필터의 복사본에 액세스할 수만 있으면 됩니다. 이를 위해 응용 프로그램에는 응용 프로그램의 `FilterWorkbenchController` 인스턴스가 작업을 수행할 수 있도록 필터 팩토리 클래스에서 제공해야 하는 비헤이비어를 정의하는 `IFilterFactory` 인터페이스가 포함되어 있습니다. `IFilterFactory`는 `FilterWorkbenchController` 클래스에서 사용되는 `getFilter()` 메서드를 정의합니다.

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

`getFilter()` 인터페이스 메서드 정의에는 특정 유형의 필터가 아니라 `BitmapFilter` 인스턴스가 반환되도록 지정되어 있습니다. `BitmapFilter` 클래스는 특정 유형의 필터를 정의하지 않습니다. `BitmapFilter`는 모든 필터 클래스의 기본 클래스입니다. 각 필터 팩토리 클래스는 자신이 작성한 필터 객체에 대한 참조를 반환하는 `getFilter()` 메서드의 특정 구현을 정의합니다. 예를 들어 다음은 `ConvolutionFactory` 클래스 소스 코드의 단축된 버전입니다.

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

`ConvolutionFactory` 클래스가 구현하는 `getFilter()` 메서드는 `ConvolutionFilter` 인스턴스를 반환하는데 이런 사실을 `getFilter()`를 호출하는 객체는 알 필요가 없습니다.

`ConvolutionFactory`가 따르는 `getFilter()` 메서드의 정의에 따르면 이 메서드는 모든 `ActionScript` 필터 클래스의 인스턴스가 될 수 있는 `BitmapFilter` 인스턴스를 반환해야 합니다.

표시 객체에 필터 적용하기

이전 단원에서 설명된 대로 필터 워크벤치 응용 프로그램은 선택한 시각적 객체에 필터를 실제로 적용하는 작업을 수행하는 `FilterWorkbenchController` 클래스의 인스턴스(여기에서는 “컨트롤러 인스턴스”라고 함)를 사용합니다. 컨트롤러 인스턴스는 필터를 적용하기 전에 먼저 필터를 적용할 이미지 또는 시각적 내용에 대해 알고 있어야 합니다. 사용자가 이미지를 선택하면 응용 프로그램은 `ImageType` 클래스에 정의된 상수 중 하나를 전달하여 `FilterWorkbenchController` 클래스의 `setFilterTarget()` 메서드를 호출합니다.

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

해당 정보를 사용하여 컨트롤러 인스턴스는 지정된 파일을 로드하고 이를 `_currentTarget`이라는 인스턴스 변수에 저장합니다.

```
private var _currentTarget:DisplayObject;

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

사용자가 필터를 선택하면 응용 프로그램은 관련 필터 팩토리 객체에 대한 참조를 지정하여 컨트롤러 인스턴스의 `setFilter()` 메서드를 호출하고, 이 컨트롤러 인스턴스는 해당 참조를 `_filterFactory`라는 인스턴스 변수에 저장합니다.

```
private var _filterFactory:IFilterFactory;

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

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

앞에서 설명한 것처럼 컨트롤러 인스턴스는 지정된 필터 팩토리 인스턴스의 특정 데이터 유형에 대해 알지 못하며 단지 이 객체가 `IFilterFactory` 인스턴스를 구현하므로 `getFilter()` 메서드를 가지고 있고, 이 메서드는 필터가 변경되면 `change(Event.CHANGE)` 이벤트를 전달한다는 것만 알고 있습니다.

사용자가 필터 패널에서 필터 속성을 변경하면 컨트롤러 인스턴스는 컨트롤러 인스턴스의 `filterChange()` 메서드를 호출하는 필터 팩토리의 `change` 이벤트를 통해 필터가 변경되었음을 확인합니다. 이 메서드는 다시 `applyTemporaryFilter()` 메서드를 호출합니다.

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

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

    // 임시로 집합에 현재 필터 추가
    _currentFilters.push(currentFilter);

    // 필터 대상의 필터 집합을 새로 고침
    _currentTarget.filters = _currentFilters;

    // 집합에서 현재 필터 제거
    // 대상에서는 내부적으로 필터 배열의 복사본을 사용하기 때문에
    // 집합에서 필터를 제거해도 필터 대상에서는 제거되지 않습니다.
    _currentFilters.pop();
}
```

표시 객체에 필터를 적용하는 작업은 `applyTemporaryFilter()` 메서드에서 발생합니다. 먼저 컨트롤러는 필터 팩토리의 `getFilter()` 메서드를 호출하여 필터 객체에 대한 참조를 검색합니다.

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

컨트롤러 인스턴스에는 `_currentFilters`라고 하는 `Array` 인스턴스 변수가 있는데 이 변수에는 표시 객체에 적용된 모든 필터가 저장되어 있습니다. 다음 단계는 새로 업데이트된 필터를 해당 배열에 추가하는 것입니다.

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

그런 다음, 이미지에 필터를 실제로 적용하는 표시 객체의 `filters` 속성에 필터의 배열을 지정합니다.

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

마지막으로, 가장 최근에 추가된 이 필터는 표시 객체에 영구적으로 적용되지 않아야 하는 “working” 필터이므로 이 필터를 `_currentFilters` 배열에서 제거합니다.

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

표시 객체는 `filters` 속성에 지정될 때 필터 배열의 복사본을 만들어서 원래 배열 대신 이 내부 배열을 사용합니다. 따라서 배열에서 이 필터를 제거해도 필터링된 표시 객체는 영향을 받지 않습니다. 따라서 필터의 배열을 변경해도 배열이 표시 객체의 `filters` 속성에 다시 지정될 때까지 표시 객체는 영향을 받지 않습니다.