

AFC

AFCOMPONENTS

A division of Professional Content, LLC

# Slideshow

## Slideshow User Guide



powered by **PRO**  
**CONTENT**

Professional Content LLC

816-298-0495

Kansas City, Missouri

[www.procontent.net](http://www.procontent.net)

Advanced Flash Components

[www.afcomponents.com](http://www.afcomponents.com)

6/11/2010

This page left  
intentionally blank

# Table of Contents

1. Component Installation.....	1
2. Getting Started .....	2
3. Using the Component Inspector .....	5
4. Using Actionscript to set the properties.....	6
5. Instantiate the component using code .....	8
6. Loading content using Actionscript .....	10
7. Types of content.....	12
External content.....	12
Library assets.....	12
Display Objects (Movie Clip, Bitmap, Sprite) .....	13
8. Pan and Zoom effect ( aka Ken Burns).....	16
9. Properties.....	17
xmlPath.....	17
items .....	17
itemsArray.....	17
scaleMode .....	18
alignType .....	18
cropAlignType.....	18
autoLoad.....	19
randomize .....	19
preloadImages .....	19
handCursor .....	20
interactiveltems.....	20
cornerRoundness.....	20
overSkinReference .....	20
border .....	21
borderColor.....	21
borderAlpha .....	21
borderThickness .....	22
preloaderType.....	22
preloaderSize.....	22

# Table of Contents

preloaderFillAlpha .....	22
preloaderLineAlpha .....	23
preloaderFillColor .....	23
preloaderLineColor .....	23
useCacheAsBitmap .....	24
temporaryClipReference .....	24
temporaryClipAlpha .....	24
temporaryClipColor .....	25
autoSlideshow .....	25
autoSlideshowDelay .....	25
autoSlideshowDirection .....	25
selectOver .....	26
selectedIndex .....	26
panZoom .....	26
transitionEffect .....	27
videoPlayer .....	27
numItems .....	28
slide .....	28
inTransitionPhase .....	28
isLoading .....	28
10. Methods .....	29
addItem() .....	29
addItemAt() .....	29
addItems() .....	29
addItemsAt() .....	30
removeItemAt() .....	30
removeItemsAt() .....	30
removeAllItems() .....	31
replaceItemAt() .....	31
spliceItems() .....	31
getItemDataAt() .....	32

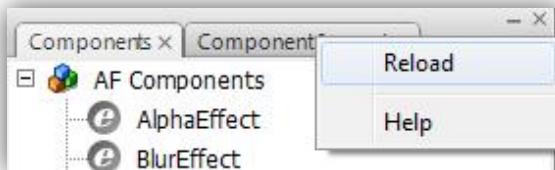
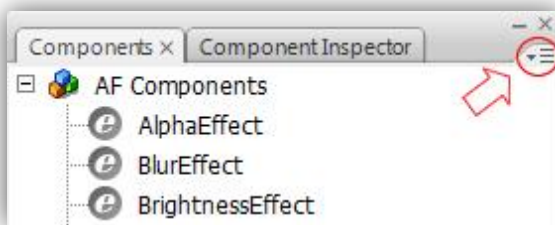
# Table of Contents

openSlide()	32
nextSlide()	33
previousSlide()	33
setSize()	33
move()	33
11. Events	34
XML_LOAD_START	34
XML_LOAD_PROGRESS	34
XML_LOAD_COMPLETE	34
XML_LOAD_ERROR	35
ITEM_ADD	35
ITEM_REMOVE	36
ITEM_CLICK	36
ITEM_DOUBLE_CLICK	37
ITEM_MOUSE_UP	37
ITEM_MOUSE_DOWN	38
ITEM_MOUSE_OVER	38
ITEM_MOUSE_OUT	39
ITEM_START	39
ITEM_PROGRESS	40
ITEM_COMPLETE	40
ITEM_ERROR	41
SLIDE_REQUEST	41
RESIZE	42
UPDATE	42
TRANSITION_START	42
TRANSITION_COMPLETE	43



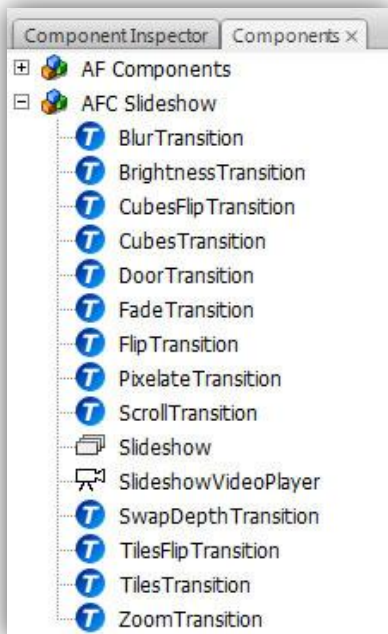
## 1. *Component Installation*

1. Double-Click on the Slideshow.mxp file.
2. Accept the license agreement.
3. If the Flash IDE was opened during installation process, reload the Components panel or restart the Flash IDE.

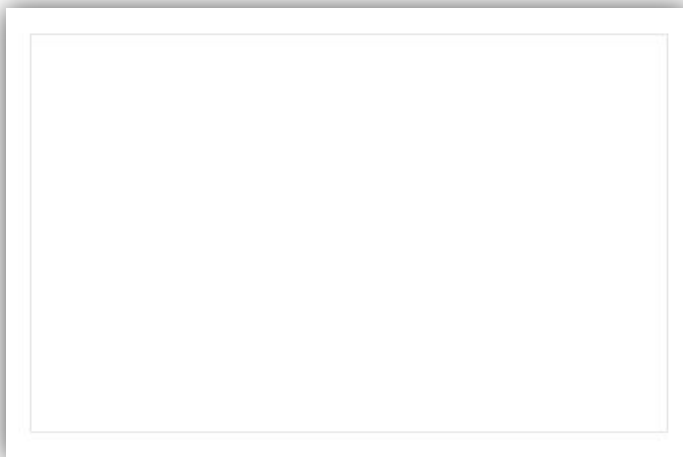


## 2. Getting Started

1. Create a new Actionscript 3 File and save the file as slideshow.fla.  
Test the movie in order to create the slideshow.swf file.
2. Open the Components panel (Window >Components) and drag an instance of the Slideshow component onto the stage. The Slideshow component can be found in the AFC Slideshow folder.

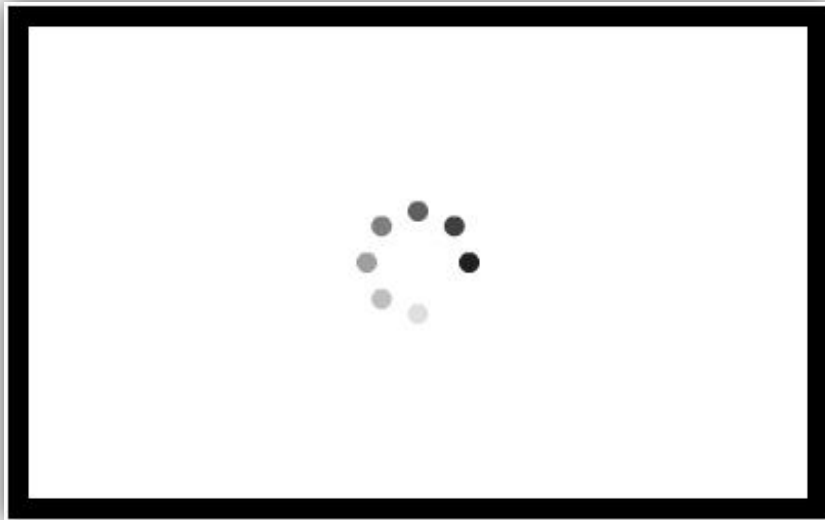


This is how the component looks on stage:



The component has a live preview which is very helpful because it allows you to see how the component will be rendered, without having to test the movie. If you chose a different preloader or set a border or other visual element, you will immediately be able to see those changes.

In the bellow image you can see how the component looks in the preview mode with a few properties changed.



To change the color and transparency of the preview items, use the “Temporary Clip Alpha” and “Temporary Clip Color” properties. You can also use the “Live Preview Items” property to set how many items you want to use in the live preview.

Now we are ready to load some content into the component.

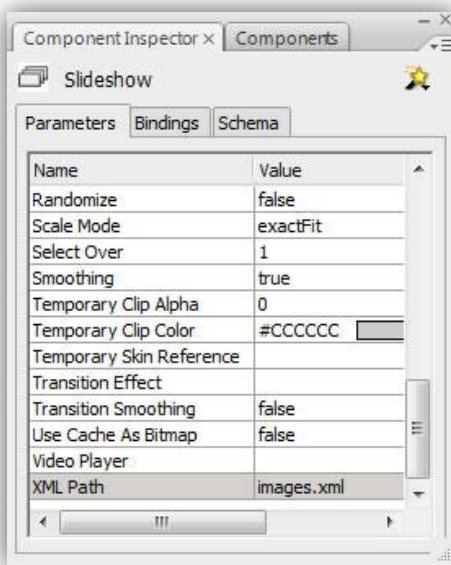
3. In the same folder as slideshow.swf create a new folder and name it “images”. Then copy the images you want to load into this folder.
4. For this example we’ll use an XML file to specify the location of the images. Create an XML file in the same directory as slideshow.swf and the images folder, and name it images.xml.
5. The XML file needs to have a structure similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<slideshow>
  <slide>
    <source>images/image1.jpg</source>
  </slide>
```

```
<slide>
  <source>images/image2.jpg</source>

</slide>
<slide>
  <source>images/image3.jpg</source>
</slide>
</ slideshow >
```

6. Open the Component Inspector panel (Window > Component Inspector) and click on the instance you've dragged onto the stage. You should now see all the customizable properties in the Component Inspector. Scroll down to XML Path and enter the path of the XML file we are using, "images.xml".

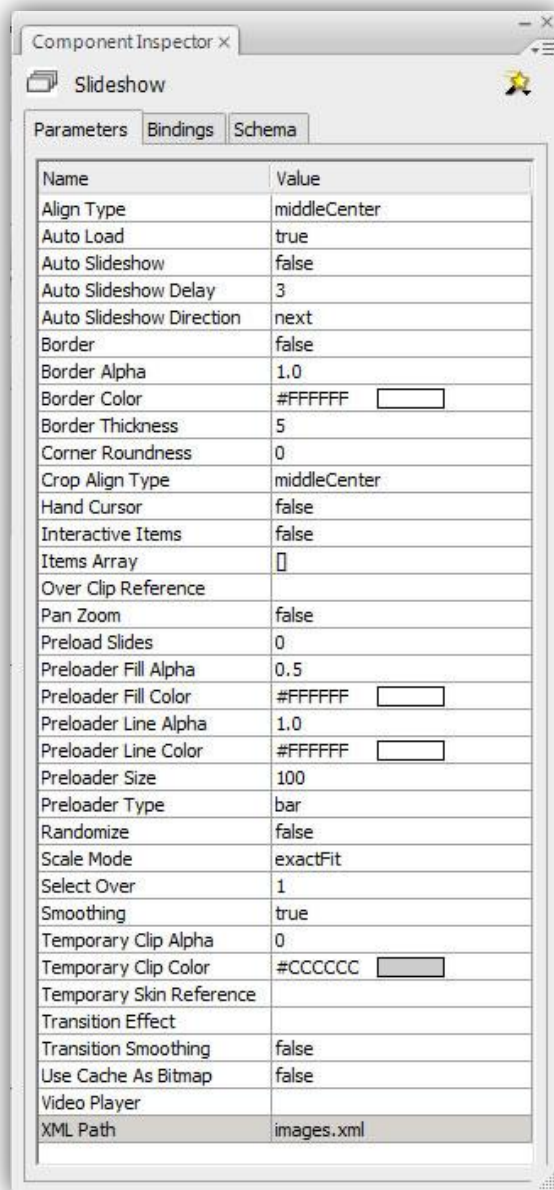


7. You are now ready to test the movie.

At this point, you have managed to load the images. In the following chapters you'll learn how you can use all the available properties and methods to customize the component to your needs.

### 3. Using the Component Inspector

The Component Inspector allows you to easily change the component's properties without using Actionscript code.



The Component Inspector has 2 columns: Name and Value.

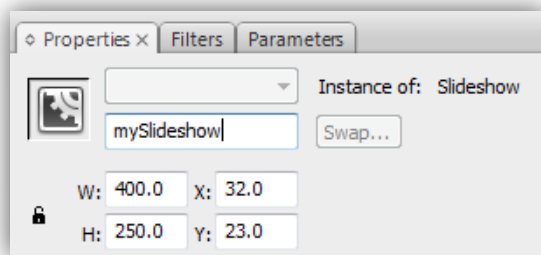
The Name column contains the name of the properties while the Value column contains the values of the properties. As you can see almost all properties have default values.

All these properties can also be changed using Actionscript code, as you will see in the next chapter.

## 4. Using Actionscript to set the properties

Setting the properties using Actionscript is very easy and can give you a greater flexibility than using the Component Inspector.

Before writing any code, you need to give the component an instance name. “mySlideshow”, for example.



It's also a good practice to create a separate layer for the Actionscript code.



Now, if you want, for example, to specify the path to the XML file using Actionscript, all the code you need to write is this:

```
mySlideshow.xmlPath = "images.xml"
```

Now, let's say you don't want to load the images from the beginning but only after you press a button. This is very easy to do using code.

First create a MovieClip symbol and give it an instance name of “myButton”.

Add this code in the actionscript layer:

```
import flash.events.MouseEvent;  
  
myButton.addEventListener(MouseEvent.CLICK, clickHandler);
```

```
function clickHandler(event:MouseEvent):void
{
    mySlideshow.xmlPath = "images.xml";
}
```

As you can see it's very easy and gives you the flexibility you couldn't have using just the Component Inspector.

In the next chapter you will see how you can instantiate the component itself using code, and not dragging it onto the stage, which will give an even greater flexibility.

## 5. *Instantiate the component using code*

Before instantiating the component, you need to make sure the component is in the library (Window > Library). If it's not there, first you need to drag an instance of the component into the library.

Now, we are ready to start coding.

1. Import the Slideshow class.

```
import com.afcomponents.slideshow.Slideshow;
```

2. Instantiate the Slideshow class.

```
var mySlideshow:Slideshow = new Slideshow();
```

3. Add the instance to the display list.

```
addChild(mySlideshow);
```

These 3 lines of code are the equivalent of dragging the component onto the stage and giving it the instance name of "mySlideshow". The advantage of using code over dragging the component onto the stage is that it allows you to choose when the component will appear on stage. Maybe you want to instantiate the component only after a button was pressed or some other piece of code executed.

The next example shows how to instantiate the component only after a button was pressed.

Like in the previous example, create a MovieClip symbol and give it the instance name of "myButton". Then, in the actionscript layer, add the following code:

```
import com.afcomponents.slideshow.Slideshow;
import flash.events.MouseEvent;

var mySlideshow:Slideshow;

myButton.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void
{
    mySlideshow = new Slideshow();
    addChild(mySlideshow);
    mySlideshow.xmlPath = "images.xml";
}
```

## 6. Loading content using Actionscript

You are not required to use an XML file to load the content. You can also do this with Actionscript, using the items property. All you need to do is create an array of Objects, each object containing the source property and then pass this array to the component's items property.

```
var myItems:Array = [{source:"images/image0.jpg"},
                    {source:"images/image1.jpg"},
                    {source:"images/image2.jpg"},
                    {source:"images/image3.jpg"},
                    {source:"images/image4.jpg"}];

mySlideshow.items = myItems;
```

Each item needs to have the "source" property to specify the location of the image. You can also set additional properties for each item to store data that you might want to use in your project. For example, you could specify a title, a description or an URL for each item.

*Example:*

```
{source:"images/image0.jpg", title:"Some title", description:"Some
description", link:"http://afcomponents.com"}
```

Note that you can add as many additional properties as you want and also the name of these properties is not important.

*Example:*

```
{source:"images/image0.jpg", abc:"Some data", xyz:"Some more data"}
```

You can also add additional data to each item if you use an XML file by creating additional attributes. The name and number of these additional attributes is not important.

*Example:*

```
<?xml version="1.0" encoding="utf-8"?>
<images>
  <slide>
    <source>images/image0.jpg</source>
    <title>Title</title>
  </slide>
  <slide>
    <source>images/image1.jpg</source>
    <abc>some data</abc>
  </slide>
  <slide>
    <source>images/image2.jpg</source>
    <xyz>more data</xyz>
  </slide>
  <slide>
    <source>images/image3.jpg</source>
  </slide>
  <slide>
    <source>images/image4.jpg</source>
  </slide>
</images>
```

If you don't want to set any additional data you could use another property to load the content, `itemsArray`. You can pass this property an array of strings that contains the location of the content you want to load.

*Example:*

```
var myItems:Array = ["images/image0.jpg", "images/image1.jpg",
"images/image2.jpg", "images/image3.jpg", "images/image4.jpg"].

mySlideshow.itemsArray = myItems;
```

As you can see, using this property requires less code. Also, this property is available in the Component Inspector while the `items` property is not available in the Component Inspector.

## 7. Types of content

The Slideshow component allows you to load external files like JPGs, GIFs, PNGs and SWFs but also library assets and Display Objects.

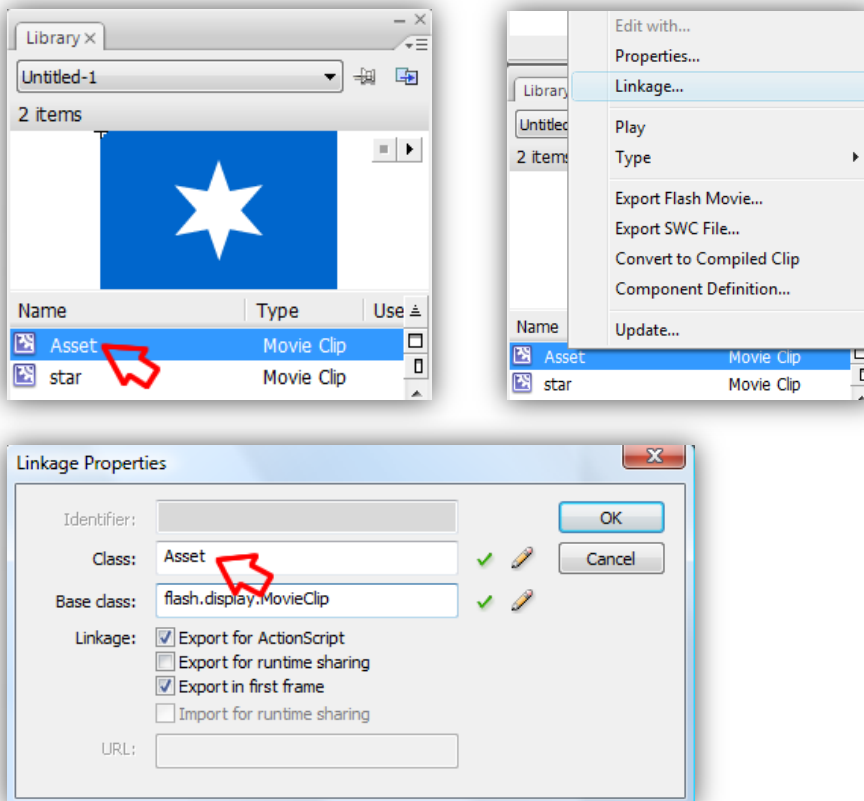
### External content

You've seen in the previous examples how to load external content. You simply need to specify the path to the content as the source.

```
mySlideshow.addItem({source: "images/MyImage.jpg"});
```

### Library assets

Before loading a symbol from the Library, you need to associate a Class to the symbol. For this, go to the Library, right-click on the symbol, chose Linkage, select Export for Actionscript, and enter a name in the Class field. Then, all you need to do is specify that Class name as the source.



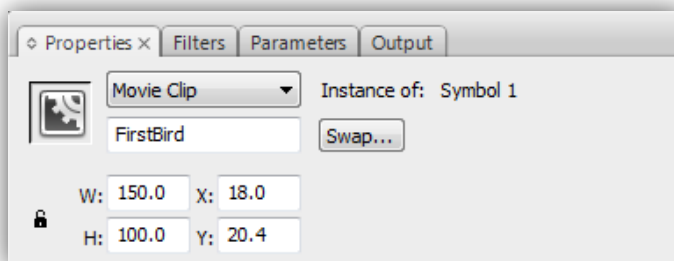
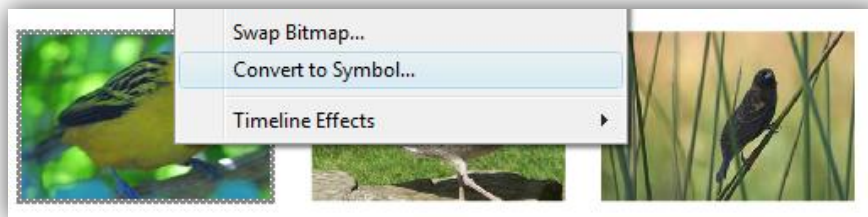
```
var myItems:Array = [{source:"Asset"}, {source:" images/image0.jpg"},  
{source:" images/image1.jpg"}]  
  
mySlideshow.items = myItems;
```

The name of the Class doesn't necessary have to be the same as the name of the symbol but when you specify the source you always use the name of the Class. I could have named the class MyAssetSample, for example, and then the source would be {source:"MyAssetSample"}.

You can also pass the source without quotation marks: {source:MyAssetSample}.

### Display Objects (Movie Clip, Bitmap, Sprite)

For this example I've imported 3 images (Bitmaps) to the Stage. Now, all I need to do is convert the Bitmaps to Movie Clips and give them instance names. To convert a Bitmap to Movie Clip right-click on the image and chose "Convert to Symbol". You can specify a name for the Symbol if you want (it's not necessary for this example) then click OK. Then click on the image, which is now a Movie Clip, and give it an instance name in the Properties panel.



You need to follow these steps for each image you want to load. After this you simply need to specify the instance names of the movie clips as the source.

```
var myItems:Array = [{source:"FirstBird"}, {source:"SecondBird"},  
{source:"ThirdBird"}];  
  
mySlideshow.items = myItems;
```

You should also move the movie clips outside the stage, so that they won't be visible outside the component.

When you test the movie, the component will take those movie clips from the stage and load them as items.

Note that any movie clip can be loaded in a single item. If you want to load the same movie clip into multiple items, you should load it as a library asset.

It's very similar when you work with Sprite objects. You actually have two options here. One would be to specify the instance of the Sprite object as the source, without quotations.

```
var rectangle:Sprite = new Sprite();  
rectangle.graphics.beginFill(0xFF0000);  
rectangle.graphics.drawRect(0, 0, 100, 100);  
rectangle.graphics.endFill();  
  
mySlideshow.addItem({source:rectangle});
```

The other option is used when you use an XML file to load the items and you need to somehow specify the Sprite object in the XML.

```
<slide>  
    <source>rectangle</source>  
</slide>
```

In this case you need to also give a name to the Sprite object and add the object to the display list.

```
var rectangle:Sprite = new Sprite();
rectangle.graphics.beginFill(0xFF0000);
rectangle.graphics.drawRect(0, 0, 100, 100);
rectangle.graphics.endFill();
rectangle.name = "rectangle";
addChild(rectangle);
```

The name of the instance needs to be the same as the name used in the XML file. If you name the Sprite object "myRectangle" (rectangle.name = "myRectangle"), in the XML file you need to have:

```
<slide>
  <source>myRectangle</source>
</slide>
```

## 8. *Pan and Zoom effect ( aka Ken Burns)*

The Slideshow component supports the popular pan and zoom effect, also known as Ken Burns. For each slide you can assign a set of properties that will determine the exact behavior of the effect. These properties are:

1. startZoom - the initial zoom of the slide
2. endZoom - the final zoom of the slide
3. startFocalX - the horizontal coordinate of the initial focal point. It takes a value between 0 to 1, 0 being the left edge and 1 the right edge. If you want the focal point to be in the middle of the image, you would set this value to 0.5.
4. startFocalY - the vertical coordinate of the initial focal point. It takes a value between 0 to 1, 0 being the top edge and 1 the bottom edge.
5. endFocalX - the vertical coordinate of the final focal point. It takes a value between 0 to 1, 0 being the left edge and 1 the bottom right.
6. endFocalY - the vertical coordinate of the final focal point. It takes a value between 0 to 1, 0 being the top edge and 1 the bottom edge.
7. effectDuration - the duration of the pan and zoom effect. Setting this property to a higher value will result in a slower motion.

These properties need to be set in the XML file for each slide, like in the example below:

```
<slide>
  <source>images /image1.jpg</source>
  <startZoom>1.3</startZoom>
  <endZoom>1</endZoom>
  <startFocalX>0.5</startFocalX>
  <startFocalY>0.5</startFocalY>
  <endFocalX>0.7</endFocalX>
  <endFocalY>0.7</endFocalY>
  <effectDuration>20</effectDuration>
</slide>
```

Also, make sure the “Pan Zoom” property is set to true when you want to use this effect.

## 9. *Properties*

### xmlPath

*Component Inspector Name:* XML Path.

*Type:* String.

*Default Value:* “ ” (empty string).

*Description:* The location of the XML file.

*Example:* mySlideshow.xmlPath = “files/xml/images.xml”;

### items

*Component Inspector Name:* Not Available.

*Type:* Array of objects.

*Default Value:* [] (empty array).

*Description:* An array of objects, each object containing the source of the content and other additional data.

*Example:* mySlideshow.items = [{source:“files/images/image0.jpg”}, {source:“files/images/image1.jpg”}, {source:“files/images/image2.jpg”}];

### itemsArray

*Component Inspector Name:* Items Array.

*Type:* Array of strings.

*Default Value:* [] (empty array).

*Description:* An array of strings, each string containing the source of the content.

*Example:* mySlideshow.itemsArray = [“files/images/image0.jpg”, “files/images/image1.jpg”, “files/images/image2.jpg”];

### scaleMode

*Component Inspector Name:* Scale Mode.

*Type:* String.

*Default Value:* "exactFit".

*Available Values:* "exactFit", "maintainAspectRatio", "crop", "cropToFit" and "noScale";

*Description:* The scaling of each item. "exactFit" will resize all items to the maximum width and height. "maintainAspectRatio" will resize the items to the maximum width or height, maintaining the aspect ratio. "noScale" will ignore the maximum width and height, maintaining the items at the original dimensions. "crop" will maintain the original size of the image, while "cropToFit" will resize the image so that the width or height of the image will fit in the view area.

*Example:* mySlideshow.scaleMode = "maintainAspectRatio";

### alignType

*Component Inspector Name:* Align Type.

*Type:* String.

*Default Value:* "middleCenter".

*Available Values:* "topLeft", "topCenter", "topRight", "middleLeft", "middleCenter", "middleRight", "bottomLeft", "bottomCenter", "bottomRight" .

*Description:* The alignment of the items.

*Example:* mySlideshow.alignType = "topLeft";

### cropAlignType

*Component Inspector Name:* Crop Align Type.

*Type:* String.

*Default Value:* "middleCenter".

*Available Values:* "topLeft", "topCenter", "topRight", "middleLeft", "middleCenter", "middleRight", "bottomLeft", "bottomCenter", "bottomRight" .

*Description:* The alignment of the cropped image.

*Example:* `mySlideshow.cropAlignType = "topLeft";`

### **autoLoad**

*Component Inspector Name:* Auto Load.

*Type:* Boolean.

*Default Value:* true.

*Description:* Indicates whether the first item will load automatically.

*Example:* `mySlideshow.autoLoad = false;`

### **randomize**

*Component Inspector Name:* Randomize.

*Type:* Boolean.

*Default Value:* false.

*Description:* Indicates whether the slides will be loaded in a random order.

*Example:* `mySlideshow.randomize = true;`

### **preloadImages**

*Component Inspector Name:* Preload Images.

*Type:* Boolean.

*Default Value:* false.

*Description:* Indicates whether images will automatically be preloaded as soon as the slideshow is instantiated so that when a new slide is opened, it will be opened instantaneously without having to wait for it to load.

*Example:* `mySlideshow.preloadImages = true;`

### handCursor

*Component Inspector Name:* Hand Cursor.

*Type:* Boolean.

*Default Value:* false.

*Description:* Indicates whether a hand cursor will be displayed when the mouse rolls over an item.

*Example:* mySlideshow.handCursor = true;

### interactiveltems

*Component Inspector Name:* Interactive Items.

*Type:* Boolean.

*Default Value:* false.

*Description:* Indicates whether the items will be interactive. Set this property to true if the items contain objects that need to respond to mouse interaction.

*Example:* mySlideshow.interactiveltems = true;

### cornerRoundness

*Component Inspector Name:* Corner Roundness.

*Type:* Number.

*Default Value:* 0.

*Description:* The amount by which the corners of the items will be rounded.

*Example:* mySlideshow.cornerRoundness = 50;

### overSkinReference

*Component Inspector Name:* Over Skin Reference.

*Type:* String.

*Default Value:* "".

*Description:* The name of a Class associated to a library symbol that will be displayed on top of each item. This property can be used to create a gloss effect or to add a watermark.

*Example:* `mySlideshow.overSkinReference = "Gloss";`

### **border**

*Component Inspector Name:* Border.

*Type:* Boolean.

*Default Value:* false.

*Description:* Indicates whether a border will be added for the item.

*Example:* `mySlideshow.border = true;`

### **borderColor**

*Component Inspector Name:* Border Color.

*Type:* uint.

*Default Value:* 0xFFFFFFFF.

*Description:* The color of the border.

*Example:* `mySlideshow.borderColor = 0xFF00FF;`

### **borderAlpha**

*Component Inspector Name:* Border Alpha.

*Type:* Number.

*Default Value:* 1.0.

*Description:* The alpha of the border.

*Example:* `mySlideshow.borderAlpha = 0.5;`

### borderThickness

*Component Inspector Name:* Border Thickness.

*Type:* Number.

*Default Value:* 5.

*Description:* The thickness of the border.

*Example:* `mySlideshow.borderThickness = 0xFF00FF;`

### preloaderType

*Component Inspector Name:* Preloader Type.

*Type:* String.

*Default Value:* "bar".

*Available Values:* "bar", "pie", "circular1", "circular2", "circular3" and "none"

*Description:* The type of preloader. "none" indicates that no preloader will be used.

*Example:* `mySlideshow.preloaderType = "circular2";`

### preloaderSize

*Component Inspector Name:* Preloader Size.

*Type:* Number.

*Default Value:* 100.

*Description:* The width of the preloader if preloaderType is set to "bar" and the radius of the preloader if preloaderType is set to "pie", "circular1", "circular2" or "circular3".

*Example:* `mySlideshow.preloaderSize = 15;`

### preloaderFillAlpha

*Component Inspector Name:* Preloader Fill Alpha.

*Type:* Number.

*Default Value:* 0.5.

*Description:* The alpha of the preloader's fill portion.

*Example:* mySlideshow.preloaderFillAlpha = 0.7;

### **preloaderLineAlpha**

*Component Inspector Name:* Preloader Line Alpha.

*Type:* Number.

*Default Value:* 1.0.

*Description:* The alpha of the preloader's line portion.

*Example:* mySlideshow.preloaderLineAlpha = 0.2;

### **preloaderFillColor**

*Component Inspector Name:* Preloader Fill Color.

*Type:* uint.

*Default Value:* 0xFFFFFFFF.

*Description:* The color of the preloader's fill portion.

*Example:* mySlideshow.preloaderFillColor = 0x0000FF;

### **preloaderLineColor**

*Component Inspector Name:* Preloader Line Color.

*Type:* uint.

*Default Value:* 0xFFFFFFFF.

*Description:* The color of the preloader's line portion.

*Example:* mySlideshow.preloaderLineColor = 0xFF0000;

### useCacheAsBitmap

*Component Inspector Name:* Use Cache As Bitmap.

*Type:* Boolean.

*Default Value:* false.

*Description:* Indicates the value of the cacheAsBitmap property for each item. Setting this property to true might result in better performance but it could also be a problem if the component has too many items. More info at

<http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/flash/display/DisplayObject.html#cacheAsBitmap>.

*Example:* `mySlideshow.useCacheAsBitmap = true;`

### temporaryClipReference

*Component Inspector Name:* Temporary Clip Reference.

*Type:* Object.

*Default Value:* null.

*Description:* The name of a Class associated to a library symbol that will be displayed for each item until the content of the item is loaded. If no value is specified, automatically a rectangle will be used with a specified alpha and color. If you don't want a temporary clip to be displayed at all, simply set the temporaryClipAlpha property to 0.

*Example:* `mySlideshow.temporaryClipReference = "TempClip";`

### temporaryClipAlpha

*Component Inspector Name:* Temporary Clip Alpha.

*Type:* Number.

*Default Value:* 0.0.

*Description:* The alpha of the temporary clip. This property will be ignored if a reference to a temporary clip is set.

*Example:* `mySlideshow.temporaryClipAlpha = 1.0;`

### temporaryClipColor

*Component Inspector Name:* Temporary Clip Color.

*Type:* uint.

*Default Value:* 0xCCCCCC.

*Description:* The color of the temporary clip. This property will be ignored if a reference to a temporary clip is set.

*Example:* mySlideshow temporaryClipColor = 0x000000;

### autoSlideshow

*Component Inspector Name:* Auto Slideshow.

*Type:* Boolean.

*Default Value:* false.

*Description:* Indicates whether the component will be in auto slideshow mode.

*Example:* mySlideshow.autoSlideshow = true;

### autoSlideshowDelay

*Component Inspector Name:* Auto Slideshow Delay.

*Type:* Number.

*Default Value:* 3.

*Description:* Indicates the delay of the auto scrollslideshow, in seconds.

*Example:* mySlideshow.autoSlideshowDelay = 4;

### autoSlideshowDirection

*Component Inspector Name:* Auto Slideshow Direction.

*Type:* String.

*Default Value:* "next".

*Available Values:* “next” and “previous”.

*Description:* Indicates the direction in which the items will be selected.

*Example:* `mySlideshow.autoSlideshowDirection = “previous”;`

### **selectOver**

*Component Inspector Name:* Select Over.

*Type:* uint.

*Default Value:* 1.

*Description:* Indicates over how many items the selection will be executed when `selectNext()` and `selectPrevious()` methods are used.

*Example:* `mySlideshow.selectOver = 3;`

### **selectedIndex**

*Component Inspector Name:* Selected Index.

*Type:* uint.

*Default Value:* 0.

*Description:* Sets the index of the item that will be opened first.

*Example:* `mySlideshow.selectedIndex= 3;`

### **panZoom**

*Component Inspector Name:* Pan Zoom.

*Type:* Boolean.

*Default Value:* false.

*Description:* Enables or disables the pan and zoom effect, also known as Ken Burns effect.

*Example:* `mySlideshow.panZoom = true;`

**transitionEffect**

*Component Inspector Name:* Transition Effect.

*Type:* Object.

*Default Value:* null.

*Description:* An instance of a TransitionEffect component.

*Example:*

```
var fadeTransition:FadeTransition = new FadeTransition();
```

```
mySlideshow.transitionEffect = fadeTransition;
```

```
or mySlideshow.transitionEffect = "fadeTransition";
```

**videoPlayer**

*Component Inspector Name:* Video Player.

*Type:* Object.

*Default Value:* null.

*Description:* An instance of a SlideshowVideoPlayer component.

*Example:*

```
var vp:SlideshowVideoPlayer = new SlideshowVideoPlayer();
```

```
mySlideshow.videoPlayer = vp;
```

*Read-only properties***numItems**

*Type:* uint.

*Description:* Returns the total number of slides.

*Example:* `trace(mySlideshow.numItems);`

**slide**

*Type:* DisplayObject.

*Description:* Returns the instance of the current slide.

*Example:* `trace(mySlideshow.slide);`

**inTransitionPhase**

*Type:* Boolean.

*Description:* Indicates whether the component is in the transition phase.

*Example:* `trace(mySlideshow.inTransitionPhase);`

**isLoading**

*Type:* Boolean.

*Description:* Indicates whether a slide is currently loading.

*Example:* `trace(mySlideshow.isLoading);`

## 10. Methods

### addItem()

*Implementation:* addItem(data:Object):void

*Description:* Adds a new item at the end of the component.

*Parameters:* data - An object containing the item's data.

*Example:*

```
mySlideshow.addItem({source:"images/image1.jpg", title:"Flower", description:"Image description"});
```

### addItemAt()

*Implementation:* addItemAt(data:Object, index:uint):void

*Description:* Adds a new item at the specified index.

*Parameters:* data - An object containing the item's data.

index - The index at which the item is to be added.

*Example:*

```
mySlideshow.addItemAt({source:"images/image1.jpg", title:"Flower", description:"Image description"}, 3);
```

### addItems()

*Implementation:* addItems(items:Array):void

*Description:* Adds an array of items to the end of the component.

*Parameters:* items - An array of items to be added.

*Example:*

```
var myItems:Array = [{source:"images/image0.jpg"},
```

```
{source:"images/image1.jpg",title:"Bird"},  
{source:"images/image2.jpg",title:"Sky"},  
{source:"images/image3.jpg"}];  
  
mySlideshow.addItem(myItems);
```

### addItemAt()

*Implementation:* addItem(items:Array, index:uint):void

*Description:* Adds an array of items at the specified index.

*Parameters:* items - An array of items to be added.

index - The index at which the items will be added.

*Example:*

```
var myItems:Array = [{source:"images/image0.jpg"},  
                    {source:"images/image1.jpg",title:"Bird"},  
                    {source:"images/image2.jpg",title:"Sky"},  
                    {source:"images/image3.jpg"}];  
  
mySlideshow.addItem(myItems, 5);
```

### removeItemAt()

*Implementation:* removeItemAt(index:uint):void

*Description:* Removes the item at the specified index.

*Parameters:* index - The index of the item to be removed.

*Example:*

```
mySlideshow.removeItemAt(1);
```

### removeItemsAt()

*Implementation:* removeItemsAt(startIndex:uint, removeCount:uint)

*Description:* Removes a specified number of items starting at a specified index.

*Parameters:* startIndex - The index of the first item which is to be removed.

removeCount - The number of items to be removed.

*Example:*

```
mySlideshow.removeItemsAt(0, 3);
```

### **removeAllItems()**

*Implementation:* removeAllItems():void

*Description:* Removes all items.

*Example:*

```
mySlideshow.removeAllItems();
```

### **replaceItemAt()**

*Implementation:* replaceItemAt(data:Object, index:uint)

*Description:* Replace an existing item at a specified index with a new item.

*Parameters:* data - The data of the new item.

index -The index of the item to be replaced.

*Example:*

```
var data:Object = {source:"images/image1.jpg", desc:"Bird"};
```

```
mySlideshow.replaceItem(data, 2);
```

### **spliceItems()**

*Implementation:*

```
spliceItems(startIndex:uint, removeCount:uint, items:Array = null):void
```

*Description:* Removes a specified number of items, starting at a specified index and adds an array of items at the specified index.

*Parameters:* startIndex - The index of the first element which is to be removed.

removeCount - The number of elements to be removed.

items - The items to be added.

*Example:*

```
var newItems:Array = [{source:"images/image0.jpg"},  
                    {source:"images/image1.jpg",title:"Bird"},  
                    {source:"images/image2.jpg",title:"Sky"},  
                    {source:"images/image3.jpg"}];  
  
mySlideshow.spliceItems(0, 3, newItems);
```

### getItemDataAt()

*Implementation:* getItemDataAt(index:uint):Object

*Description:* Returns the item data at the specified index.

*Parameters:* index - The index of the item to be returned.

*Example:*

```
trace(mySlideshow.getItemDataAt(2).index);// output 2
```

### openSlide()

*Implementation:* openSlide(index:uint):void

*Description:* Opens the item at the specified index.

*Parameter:* index – The index of the item to be opened.

*Example:*

```
mySlideshow.openSlide(5);
```

### nextSlide()

*Implementation:* nextSlide():void

*Description:* Opens the next slide.

*Example:*

```
mySlideshow.nextSlide();
```

### previousSlide()

*Implementation:* previousSlide():void

*Description:* Opens the previous slide.

*Example:*

```
mySlideshow.previousSlide();
```

### setSize()

*Implementation:* setSize(width:Number, height:Number):void

*Description:* Sets the component's size at the specified width and height.

*Example:*

```
mySlideshow.setSize(800, 150);
```

### move()

*Implementation:* move(x:Number, y:Number):void

*Description:* Moves the component at the specified position.

*Example:*

```
mySlideshow.move(100, 100);
```

## 11. Events

### XML\_LOAD\_START

*Description:* Dispatched when the loading of the XML file begins.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.XML_LOAD_START,
eventHandler);
function eventHandler(event:SlideshowEvent):void
{
    trace("XML loading begins");
}
```

### XML\_LOAD\_PROGRESS

*Description:* Dispatched during the loading of the XML file.

*Special Properties:* bytesLoaded – The number of bytes loaded thus far.

bytesTotal – The total number of bytes to be loaded.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.XML_LOAD_PROGRESS,
eventHandler);
function eventHandler(event:SlideshowEvent):void
{
    trace("Loaded " + event.bytesLoaded + " out of " +
event.bytesTotal);
}
```

### XML\_LOAD\_COMPLETE

*Description:* Dispatched when the loading of an XML file is complete.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.XML_LOAD_COMPLETE, eventHandler)
function eventHandler(event:SlideshowEvent):void
{
    trace("XML file loaded");
}
```

### XML\_LOAD\_ERROR

*Description:* Dispatched when an error occurs during the loading of the XML file.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.XML_LOAD_ERROR, eventHandler)
function eventHandler(event:SlideshowEvent):void
{
    trace("XML load error");
}
```

### ITEM\_ADD

*Description:* Dispatched when a new item was added.

*Special Properties:* item – The item that was added.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.ITEM_ADD, eventHandler);

function eventHandler(event:SlideshowEvent):void
{
    trace(event.index); // output 3
    trace(event.data); // output {source:"images/image1.jpg",
title:"Flower", desc:"Sun flower"}
```

```
        trace(event.data.title); // output "Flower"
    }

    mySlideshow.addItemAt({source:"images/imagel.jpg", title:"Flower",
    desc:"Sun flower"}, 3);
```

## ITEM\_REMOVE

*Description:* Dispatched when an item was removed.

*Special Properties:* item – The item that was removed.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.ITEM_REMOVE,
eventHandler);

function eventHandler(event:SlideshowEvent):void
{
    trace(event.index); // output 1
}
mySlideshow.removeItemAt(1);
```

## ITEM\_CLICK

*Description:* Dispatched when an item was clicked.

*Special Properties:* item – The item that was clicked.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
```

```
mySlideshow.addEventListener(SlideshowEvent.ITEM_CLICK, eventHandler);

function eventHandler(event:SlideshowEvent):void{
    trace(event.index);
}
```

### ITEM\_DOUBLE\_CLICK

*Description:* Dispatched when an item was double-clicked.

*Special Properties:* item – The item that was double-clicked.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.ITEM_DOUBLE_CLICK,
eventHandler);

function eventHandler(event:SlideshowEvent):void
{
    trace(event.data);
}
```

### ITEM\_MOUSE\_UP

*Description:* Dispatched when the mouse is released over an item.

*Special Properties:* item – The item under the mouse pointer.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.ITEM_MOUSE_UP,
eventHandler);
```

```
function eventHandler(event:SlideshowEvent):void
{
    trace(event.index);
}
```

### ITEM\_MOUSE\_DOWN

*Description:* Dispatched when the mouse is pressed over an item.

*Special Properties:* item – The item under the mouse pointer.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.ITEM_MOUSE_DOWN,
eventHandler);

function eventHandler(event:SlideshowEvent):void
{
    trace(event.index);
}
```

### ITEM\_MOUSE\_OVER

*Description:* Dispatched when the mouse pointer is moved over an item.

*Special Properties:* item – The item under the mouse pointer.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.ITEM_MOUSE_OVER,
eventHandler);
```

```
function eventHandler(event:SlideshowEvent):void
{
    trace(event.index);
}
```

### ITEM\_MOUSE\_OUT

*Description:* Dispatched when the mouse pointer is moved away from an item.

*Special Properties:* item – The item under the mouse pointer.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.ITEM_MOUSE_OUT,
eventHandler);

function eventHandler(event:SlideshowEvent):void
{
    trace(event.index);
}
```

### ITEM\_START

*Description:* Dispatched when the content of an item begins to load.

*Special Properties:* item – The item that begins to load its content.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.ITEM_START, eventHandler);

function eventHandler(event:SlideshowEvent):void{
    trace("Item at index: " + event.index + " has started loading");
}
```

```
}
```

## ITEM\_PROGRESS

*Description:* Dispatched during the loading of an item's content.

*Special Properties:* item – The item that is loading its content.

index – The index of the item.

data – The data of the item.

bytesLoaded – The number of bytes loaded thus far.

bytesLoaded – The number of bytes to be loaded.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.ITEM_PROGRESS,
eventHandler);
function eventHandler(event:SlideshowEvent):void
{
    trace("Loaded " + event.bytesLoaded + " out of " +
event.bytesTotal);
}
```

## ITEM\_COMPLETE

*Description:* Dispatched when the content of an item is completely loaded.

*Special Properties:* item – The item that has completely loaded its content.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.ITEM_COMPLETE,
eventHandler);
```

```
function eventHandler(event:SlideshowEvent):void
{
    trace("Item at index: " + event.index + " has finished loading");
}
```

### ITEM\_ERROR

*Description:* Dispatched when an error has occurred during the loading process.

*Special Properties:* item – The item on which the error occurred.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.ITEM_ERROR, eventHandler);

function eventHandler(event:SlideshowEvent):void
{
    trace("Item at index: " + event.index + " could not be loaded");
}
```

### SLIDE\_REQUEST

*Description:* Dispatched when a slide is requested.

*Special Properties:* item – The item on which the error occurred.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.SLIDE_REQUEST,
slideRequestHandler);
```

```
function slideRequestHandler(event:SlideshowEvent):void
{
    trace("The slide at index: " + event.index + " was requested");
}
```

## RESIZE

*Description:* Dispatched when the size of the component changes.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.RESIZE, eventHandler);
function eventHandler(event:SlideshowEvent):void
{
    trace(myStackFlow.width);
}
```

## UPDATE

*Description:* Dispatched after the component was updated because a property changed or an item was added or removed.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.UPDATE, eventHandler);
function eventHandler(event:SlideshowEvent):void
{
    trace("Something changed!");
}
```

## TRANSITION\_START

*Description:* Dispatched when a transition starts.

*Special Properties:* item – The item which has loaded.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.TRANSITION_START,
transitionStartHandler);

function transitionStartHandler(event:SlideshowEvent):void
{
    trace("Index of the selected item: " + event.index);
}
```

## TRANSITION\_COMPLETE

*Description:* Dispatched when a slide is requested.

*Special Properties:* item – The item which has loaded.

index – The index of the item.

data – The data of the item.

*Example:*

```
import com.afcomponents.events.SlideshowEvent;
mySlideshow.addEventListener(SlideshowEvent.TRANSITION_COMPLETE,
transitionCompleteHandler);

function transitionCompleteHandler(event:SlideshowEvent):void
{
    trace("Index of the selected item: " + event.index);
}
```