SlideshowBox

ActionScript 3.0 Usage

Using SlideshowBox by Code

Here are the steps you need to follow in order to use the SlideshowBox component with ActionScript 3.0 code:

1. Create a new Flash ActionScript 3.0 file and set its frame rate to 30 fps. Open up the Components panel (Ctrl + F7 for Windows or Command + F7 for Mac), select the SlideshowBox component and drag it over the Library panel. Next, bring up the Actions panel (F9 for Windows or Option + F9 for Mac) and paste the following lines of code inside it.

2. Import the necessary classes for the SlideshowBox component and the SlideshowBox events class (SlideshowBoxEvent). In case you instantiate the SlideshowBox template by code, you need to import that class too. The template class names are the same as the template names and their complete class path is **com.jumpeye.slideshowBox.templates**.

```
import com.jumpeye.events.SlideshowBoxEvent;
import SlideshowBox;
// For instantiating the SimpleScale template using the template's constructor.
import com.jumpeye.slideshowBox.templates.SimpleScale;
```

3. Set the stage alignment and scale mode. This is necessary when the Flash Player window resizes or switches from normal screen to full screen and back.

```
stage.scaleMode = "noScale";
stage.align = "TL";
```

4. Create and set up the SlideshowBox instance. By setting the source property of the component, the album will immediately start working since it already has embedded into it the SimpleGrid template and will use the template's default settings.

```
var myAlbum:SlideshowBox = new SlideshowBox();
myAlbum.x = 0;
myAlbum.y = 0;
myAlbum.width = 550;
myAlbum.height = 400;
addChild(myAlbum);

myAlbum.source = "source.xml";
```

5. Listen for events triggered by the component. We recommend adding the listeners before setting the template and the source XML data by code otherwise some events like SlideshowBoxEvent.TEMPLATE_CREATION_DONE will be dispatched before the listener is added and the event will never be captured.

```
myAlbum.addEventListener(SlideshowBoxEvent.TEMPLATE_CREATION_DONE,
setUpTemplate);
myAlbum.addEventListener(SlideshowBoxEvent.INIT, templateInitialized);
myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_SHOW, largeImageDisplayed);
myAlbum.addEventListener(SlideshowBoxEvent.SLIDESHOW_START, slideShowStarted);
myAlbum.addEventListener(SlideshowBoxEvent.SLIDESHOW_STOP, slideShowStopped);

// The album template can be set up as soon as it is instantiated, on the
// SlideshowBoxEvent.TEMPLATE_CREATION_DONE event.
function setUpTemplate(evt:SlideshowBoxEvent):void {
      trace(myAlbum.templateName+" has been instantiated, you can now set the
template properties.");
      // Set up the album template.
      myAlbum.template.backgroundColor = 0xFF0000;
      myAlbum.template.slideShowSpeed = 3;
}

// The number of images in the list can only be retrieved after the
// album template has been initialized.
function templateInitialized(evt:SlideshowBoxEvent):void {
      trace("TOTAL IMAGES :: "+myAlbum.imageList.length);
      trace("Last image infos:");
      var lastImageIndex:uint = myAlbum.imageList.length;
      trace("index = "+myAlbum.imageList[lastImageIndex - 1].index);
      trace("title = "+myAlbum.imageList[lastImageIndex - 1].title);
      trace("description = "+myAlbum.imageList[lastImageIndex - 1].description);
}

function largeImageDisplayed(evt:SlideshowBoxEvent):void {
      trace("Image "+evt.item.index+" is now visible.");
}

function slideShowStarted(evt:SlideshowBoxEvent):void {
      trace("The slide show has begun.");
}

function slideShowStopped(evt:SlideshowBoxEvent):void {
      trace("The slide show has ended.");
}
```

6. Specify the album template you wish to use. This should be done after the event listeners were added. There are two ways to do this: either specify the template name or instantiate the template yourself by code:

A. Specify the template name (in this case you don't need to import the template class):

```
myAlbum.templateName = "SimpleSlide";
```

or

B. Instantiate the template by code (you must import the template class as shown at step 1):

```
var myTemplate:SimpleScale = new SimpleScale();
```

```
myAlbum.template = myTemplate;
```

7. Set the source property of the component. Once you set the source property, the SlideshowBox component will instruct the album template to start loading the images:

```
myAlbum.source = "source.xml";
```

8. Compile the application by hitting Ctrl + Enter for Windows or Cmd + Enter for Mac.

9. Remove the watermark. SlideshowBox displays a watermark over the slideshow if the slideshow is hosted on an Internet domain for which a domain key license has not been purchased. In this case SlideshowBox is considered to run in free license mode. To remove the watermark, you need to purchase a domain key license, provide it to the SlideshowBox component and host the slideshow on the domain for which the key was purchased.

```
var domainArray:Array = new Array("178g2i74V9m3");
MyAlbum.domainKeys = domainArray;
```

Code Samples

How to get the total number of images in the list

The total number of images in the slide show can only be retrieved after the album template has been initialized, that is after the *SlideshowBoxEvent.INIT* event is triggered. For this, simply get the length of the *imageList* property, which is an Array of image items, each items containing information about an image. For more information on the properties of an image item please see The Image Item Object Properties section of this document.

```
trace("TOTAL IMAGES :: "+myAlbum.imageList.length);
```

How to retrieve information about an image

Information on any image from the list can be retrieved only after the album template has been initialized, that is when the *SlideshowBoxEvent.INIT* event is triggered. For this, simply get the item you need from the *imageList* Array, by specifying an index (from 0 to imageList.length – 1). The object retrieved has a series of properties containing information on the image (index, title, description...). For more information on the properties of an image item please see The Image Item Object Properties section of this document.

```
var lastImageIndex:uint = myAlbum.imageList.length;
trace("index = "+myAlbum.imageList[lastImageIndex - 1].index);
trace("title = "+myAlbum.imageList[lastImageIndex - 1].title);
trace("description = "+myAlbum.imageList[lastImageIndex - 1].description);
```

How to make the album resize if the Flash Player window resizes

When the *Event.RESIZE* event is triggered on the stage (the Flash Player window resizes) you need to set the new size on the album. But it is important to do so only when the Flash Player is in normal screen mode and **NOT** in full screen mode.

```
stage.addEventListener(Event.RESIZE, stageResize);

function stageResize(evt:Event):void {

	if (stage.displayState != "fullScreen") {
		// The album should be resized only if the stage is not in
		// full screen mode.
		myAlbum.width = stage.stageWidth;
		myAlbum.height = stage.stageHeight;
	}
}
```

How to stop the album to resize to full screen when entering in full screen mode

By default, when entering in full screen mode, the album will automatically resize so that it covers the entire stage. When the Flash application comprises of only the SlideshowBox album, this behavior is perfectly acceptable. However, when the SlideshowBox album is integrated into a larger Flash application that contains besides the album other display objects on the stage, when entering in full screen mode the album could cover the other display objects. If you do not want this to happen and would like the SlideshowBox component to keep its size regardless of the screen mode (full screen or normal screen) you need to override the default behavior (which is to resize to full screen).

For this, simply set the *overrideFullScreenChange* property of the SlideshowBox template to *true* and handle the *FullScreenEvent.FULL_SCREEN* event yourself. This can be done either when manually instantiating the template by code or, if you set the template by its name using the *templateName* property, you need to set the *overrideFullScreenChange* property after the album template has been created, so you'll need to listen for the SlideshowBoxEvent.TEMPLATE_CREATION_DONE event.

```
import SlideshowBox;
import com.jumpeye.slideshowBox.templates.SimpleScale;

stage.scaleMode = "noScale";
stage.align = "TL";

var myAlbum:SlideshowBox = new SlideshowBox();
addChild(myAlbum);

var myTemplate:SimpleScale = new SimpleScale();
myTemplate.backgroundColor = 0xFF0000;
myTemplate.slideShowSpeed = 3;
myTemplate.fullScreenButton = true;
myTemplate.overrideFullScreenChange = true;

myAlbum.template = myTemplate;
```

or

```
import SlideshowBox;
import com.jumpeye.events.SlideshowBoxEvent;

stage.scaleMode = "noScale";
stage.align = "TL";

var myAlbum:SlideshowBox = new SlideshowBox();
myAlbum.addEventListener(SlideshowBoxEvent.TEMPLATE_CREATION_DONE,
setUpTemplate);
addChild(myAlbum);

myAlbum.templateName = "SimpleScale";

function setUpTemplate(evt:SlideshowBoxEvent):void {
      myAlbum.template.backgroundColor = 0xFF0000;
      myAlbum.template.slideShowSpeed = 3;
      myAlbum.template.fullScreenButton = true;
      myAlbum.template.overrideFullScreenChange = true;
}
```

***Note**: The SlideshowBox component and the SimpleScale template must be placed in the projects's library for this to work.*

Properties

Name	Type/ Panel	Description
audioFile	String yes	The URL or path and file name to the mp3 file used as background sound. If there is an audio file specified, then the component will start to play it by default. In this case the audio player will be displayed in the top-right corner of the album and you can interact with it. If there is no audio file specified or there was an error reading the audio file, the player will not be displayed. **Example** `myAlbum.audioFile = "SomeBackgroundSong.mp3";`
audioPlayerColor	uint yes	Specifies the color used by the audio player icon. The default color is white (0xFFFFFF). **Example** `myAlbum.audioPlayerColor = 0x71B7E6;`
audioPlayerIcon	String yes	The icon used to represent the audio player. Possible values are "speaker" and "equalizer". The default value is "speaker". Depending on the selected icon, the audio player will be displayed either as a speaker or equalizer bars. Each type of icon has two states: audio on and audio off. The audio player states will change only when the user interacts with the audio player (by clicking on the icons).

		Example `myAlbum.audioPlayerIcon = "equalizer";`
audioPlayerOff	String yes	The class name of clip from the Library to be used as the "audio off" symbol for the audio player. If you only specify one of the ON or OFF states, the audio player will display the default icon (speaker or equalizer) for the other missing state. **Example** `myAlbum.audioPlayerOff = "AudioOffClip";`
audioPlayerOn	String yes	The class name of clip from the Library to be used as the "audio on" symbol for the audio player. If you only specify one of the ON or OFF states, the audio player will display the default icon (speaker or equalizer) for the other missing state. **Example** `myAlbum.audioPlayerOn = "AudioOnClip";`
audioPlayMode	String yes	The way the audio will play during the slideshow: play the audio automatically, the audio is stopped by default and synchronize with the slideshow. Possible values are - **audioOn** – starts the audio automatically - **audioOff** – loads the album but the audio file will not play and the audio player will display the OFF icon - **synchronizeWithSlideshow** – the audio file is played only when the slideshow is playing. In this case the audio player buttons work like mute/unmute buttons, so when the audio player is in the OFF state (mute) the audio file will not play, even if the slideshow is playing. The default value is *synchronizeWithSlideshow*. **Example** `myAlbum.audioPlayMode = "synchronizeWithSlideshow";`
config	XML or String no	The XML data or XML String containing the configuration parameters for the SlideshowBox component and the album template. It can also receive the URL of the XML file containing the configuration parameters. **Example** `myAlbum.config = "albumConfiguration.xml";` or `myAlbum.config = <SlideshowBox>` `<params>` `...` `</params>` `<template>` `...` `</template>` `</SlideshowBox>;`
domainKeys	Array yes	A list of hash keys generated for the Internet domains where the slideshow will be hosted on. The list can contain at least one key corresponding for an Internet domain.

		Note: If such a domain key is not specified, the slideshow will always display a watermark. When the slideshow is tested within the Adobe Flash Professional IDE or viewed on a local computer the watermark will always be displayed because the slideshow is not hosted on an Internet domain. Form more information on the domain keys please consult the User Manual. **Example** `myAlbum.domainKeys = ["yt2786g37JL896c3"];` or `myAlbum.domainKeys = ["yt2786g37JL896c3", "H78s84kdl153hdj", "dh872ja0lg1763jd"];`
imageList	Array no	**[read-only]** A list of objects containing information about each image. For more information about the image item object, please see **The Image Item Object Properties** section in this document. **Example** `trace("Number of images in the slide show : "+myAlbum.imageList.length);` `trace("Title of the third image: "+myAlbum.imageList[2].title);`
loopAudio	Boolean yes	Specifies whether the audio file will loop continuously or play only once. The default value is true. **Example** `myAlbum.loopAudio = false;`
selectedImage	Object no	**[read-only]** An object containing information about the currently selected image. For more information about the image item object, please see **The Image Item Object Properties** section in this document. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_SHOW, photoShow);` `function photoShow(evt:SlideshowBoxEvent):void {` `  trace("Image "+(myAlbum.selectedImage.index + 1)+" is now visible");` `}`
selectedIndex	int no	**[read-only]** The 0-based index of the currently selected image. The index of the first image is 0. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_SHOW, photoShow);`

		function photoShow(evt:SlideshowBoxEvent):void { trace("Image "+(myAlbum.selectedIndex + 1)+" is now visible"); }
source	XML or String yes	The XML data or XML String containing the list of images for the SlideshowBox component and the album template. It can also receive the URL of the XML file containing the list of images or the URL of the RSS object for Flickr, Picasa, PhotoBucket or Smugmug albums. ***Note:*** *For information on how to use the RSS for Flickr, Picasa, Photobucket or Smugmug albums, please consult the "Using SlideshowBox with RSS feeds" PDF file.* **Example** myAlbum.source = "source.xml"; or myAlbum.source = <album> <items> <item> . . . </item> <item> . . . </item> . . . </items> </album>; or myAlbum.source = "http://www.smugmug.com/hack/feed.mg?Type=search&Data=nature&format=rss200";
template	BaseAlbum no	A reference to the SlideshowBox template used by the component. By default, the component already has the SimpleGrid template embedded into it, so it can be used "out of the box" without having to write code to set up an album template. A reference to the album template can only be retrieved after the *SlideshowBoxEvent.TEMPLATE_CREATION_DONE* event has been triggered. Until then, the property will return the value *null*. When manually creating the template instance by code, the *SlideshowBoxEvent.TEMPLATE_CREATION_DONE* event will be instantly triggered when setting the *template* property with the reference to the template created by code. So this is why the listener for this event should be set before setting the template. **Example** import com.jumpeye.events.SlideshowBoxEvent; import com.jumpeye.slideshowBox.templates.SimpleScale; myAlbum.addEventListener(SlideshowBoxEvent.TEMP

		LATE_CREATION_DONE, setUpTemplate); var myTemplate:SimpleScale = new SimpleScale(); myAlbum.template = myTemplate;
templateName	String no	The name of the album template used by the component. The name of the album template does not include its entire class path, but only its name, as it is found in the documentation for each SlideshowBox template. The listener for the *SlideshowBoxEvent.TEMPLATE_CREATION_DONE* event should be added before setting the template. The default value is SimpleGrid. **Example** import com.jumpeye.events.SlideshowBoxEvent; myAlbum.addEventListener(SlideshowBoxEvent.TEMP LATE_CREATION_DONE, setUpTemplate); myAlbum.templateName = "SimpleScale";
volume	uint no	The volume of the audio file playing in the background. The default value is 70. **Example** myAlbum.volume = 100; // set the audio volume to maximum myAlbum.volume = 0; // set the audio to mute

Methods

Method	Description
next	Loads and displays the next large image in the list. This method is similar to using the *openLargeImage()* method with the index of the next image in the list (*selectedIndex* + 1). If the currently selected image is the last one (*selectedIndex* == *imageList.length* - 1) then the new index should point to the first image (index 0). ***Note**: The previous(), next() and openLargeImage() methods only take effect after the show transition of the currently selected large image has ended.* **Example** previousButton.addEventListener(MouseEvent.CLICK, showPreviousImage); nextButton.addEventListener(MouseEvent.CLICK, showNextImage); function showPreviousImage(evt:MouseEvent):void { if (myAlbum.selectedIndex == 0) myAlbum.openLargeImage(myAlbum.imageList.length - 1); myAlbum.previous(); trace(myAlbum.selectedIndex+" / "+ (myAlbum.imageList.length - 1)); }

	function showNextImage(evt:MouseEvent):void { if (myAlbum.selectedIndex == myAlbum.imageList.length - 1) myAlbum.openLargeImage(0); else myAlbum.next(); trace(myAlbum.selectedIndex+" / "+ (myAlbum.imageList.length - 1)); }
nextPage	Loads and displays the next page of thumbnails. This method works only if the album uses thumbnail pages, otherwise it will not have any effect. **Example** import com.jumpeye.events.SlideshowBoxEvent; myAlbum.addEventListener(SlideshowBoxEvent.PAGE_CHANGE, thumbsPageChanged); function thumbsPageChanged(evt:SlideshowBoxEvent):void { trace("Current thumbnail page :: "+evt.pageIndex); trace("First thumbnail index = "+evt.firstIndex); trace("Last thumbnail index = "+evt.lastIndex); } previousButton.addEventListener(MouseEvent.CLICK, showPreviousImage); nextButton.addEventListener(MouseEvent.CLICK, showNextImage); function showPreviousImage(evt:MouseEvent):void { myAlbum.previousPage(); } function showNextImage(evt:MouseEvent):void { myAlbum.nextPage(); }
openLargeImage	Loads and displays the large image identified by the index specified as parameter. The index is 0-based, meaning that the index of the first image 0, the index of the second image is 1 and so on. **Parametes** - *index*:uint – the index of the 0-based image from the album. **Example** previousButton.addEventListener(MouseEvent.CLICK, showPreviousImage); nextButton.addEventListener(MouseEvent.CLICK, showNextImage); function showPreviousImage(evt:MouseEvent):void { if (myAlbum.selectedIndex == 0) myAlbum.openLargeImage(myAlbum.imageList.length - 1); myAlbum.previous(); trace(myAlbum.selectedIndex+" / "+ (myAlbum.imageList.length - 1)); } function showNextImage(evt:MouseEvent):void { if (myAlbum.selectedIndex == myAlbum.imageList.length - 1) myAlbum.openLargeImage(0); else myAlbum.next(); trace(myAlbum.selectedIndex+" / "+ (myAlbum.imageList.length - 1)); }

previous	Loads and displays the previous large image in the list. This method is similar to using the *openLargeImage()* method with the index of the previous image in the list (*selectedIndex* - 1). If the currently selected image is the first one (*selectedIndex* == 0) then the new index should point to the last image (*imageList.length* - 1). **Example** `previousButton.addEventListener(MouseEvent.CLICK, showPreviousImage);` `nextButton.addEventListener(MouseEvent.CLICK, showNextImage);` `function showPreviousImage(evt:MouseEvent):void {` `if (myAlbum.selectedIndex == 0) myAlbum.openLargeImage(myAlbum.imageList.length - 1);` `myAlbum.previous();` `trace(myAlbum.selectedIndex+" / "+ (myAlbum.imageList.length - 1));` `}` `function showNextImage(evt:MouseEvent):void {` `if (myAlbum.selectedIndex == myAlbum.imageList.length - 1) myAlbum.openLargeImage(0);` `else myAlbum.next();` `trace(myAlbum.selectedIndex+" / "+ (myAlbum.imageList.length - 1));` `}`
pauseAudio	Pauses the audio playing in the background. If there is no audio file loaded, the function call will have no effect.
playAudio	Starts playing the audio in the background. If the audio was already paused, it will resume it. If there is no audio file loaded, the function will have no effect.
previousPage	Loads and displays the previous page of thumbnails. This method works only if the album uses thumbnail pages, otherwise it will not have any effect. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PAGE_CHANGE, thumbsPageChanged);` `function thumbsPageChanged(evt:SlideshowBoxEvent):void {` `trace("Current thumbnail page :: "+evt.pageIndex);` `trace("First thumbnail index = "+evt.firstIndex);` `trace("Last thumbnail index = "+evt.lastIndex);` `}` `previousButton.addEventListener(MouseEvent.CLICK, showPreviousImage);` `nextButton.addEventListener(MouseEvent.CLICK, showNextImage);` `function showPreviousImage(evt:MouseEvent):void {` `myAlbum.previousPage();` `}` `function showNextImage(evt:MouseEvent):void {` `myAlbum.nextPage();` `}`
startSlideShow	Starts the automatic slide show of the images. The delay between the images is set by the *slideShowSpeed* property.

	Example `myAlbum.startSlideShow();`
stopSlideShow	Stops the automatic slide show of the images. **Example** `myAlbum.startSlideShow();`

Events

Event	Description
AUDIO_LOAD_COMPLETE	Dispatched after the audio (mp3) file has finished loading. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.AUDIO_LOAD_COMPLETE, audioLoadComplete);` `function audioLoadComplete(evt:SlideshowBoxEvent):void {` `  trace("The audio file has loaded and will start playing...");` `}`
AUDIO_LOAD_ERROR	Dispatched after an IO error occurred during the loading process of the audio file. **Properties** - *text*:String – the error message received from the IOErrorEvent object.. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.AUDIO_LOAD_ERROR, audioLoadError);` `function audioLoadError(evt:SlideshowBoxEvent):void {` `  trace("Error loading the audio file :: "+evt.text);` `}`
AUDIO_LOAD_OPEN	Dispatched when the loading of the audio file starts. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.AUDIO_LOAD_OPEN, audioLoadOpen);` `function audioLoadOpen(evt:SlideshowBoxEvent):void {` `  trace("Starting to load the audio file...");` `}`

AUDIO_LOAD_PROGRESS	Dispatched during the loading process of the audio file. **Properties** - *bytesLoaded*:Number – the number of bytes loaded so far of the audio file; - *bytesTotal*:Number – the total size in bytes of the audio file. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.AUDIO_LOAD_PROGRESS, audioLoadProgress);` `function audioLoadProgress(evt:SlideshowBoxEvent):void {` `  trace("Loading the audio file :: "+evt.bytesLoaded+" / "+evt.bytesTotal);` `}`
AUDIO_PLAY	Dispatched when the audio player starts or resumes playing in the background. **Example** `myAlbum.addEventListener(SlideshowBoxEvent.AUDIO_PLAY, audioPlay);` `function audioPlay(evt:SlideshowBoxEvent):void {` `  trace("Playing the background audio...");` `}`
AUDIO_STOP	Dispatched when the audio player stops playing the audio in the background. **Example** `myAlbum.addEventListener(SlideshowBoxEvent.AUDIO_STOP, audioStop);` `function audioStop(evt:SlideshowBoxEvent):void {` `  trace("The background audio has stopped...");` `}`
AUDIO_SECURITY_ERROR	Dispatched after a SecurityErrorEvent occured during the loading process of the audio file. **Properties** - *text*:String – the error message received from the SecurityErrorEvent object. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.AUDIO_SECURITY_ERROR, audioSecurityError);` `function audioSecurityError(evt:SlideshowBoxEvent):void {` `  trace("Security error while loading the audio file :: "+evt.text);` `}`
CONFIG_ASYNC_ERROR	Dispatched when there was an exception from native asynchronous code during the loading process of the configuration XML.

	Properties - *text*:String – the error message received from the AsyncErrorEvent object. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.CONFIG_ASYNC_ERROR, configAsyncError);` `function configAsyncError(evt:SlideshowBoxEvent):void {` `  trace("Asynchronous error while loading the configuration XML :: "+evt.text);` `}`
CONFIG_LOAD_COMPLETE	Dispatched after the configuration XML data has completely loaded. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.CONFIG_LOAD_COMPLETE, configLoadComplete);` `function configLoadComplete(evt:SlideshowBoxEvent):void {` `  trace("The configuration XML has completely loaded...");` `}`
CONFIG_LOAD_ERROR	Dispatched after an IO error occurred during the loading process of the configuration XML data. **Properties** - *text*:String – the error message received from the IOErrorEvent object.. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.CONFIG_LOAD_ERROR, configLoadError);` `function configLoadError(evt:SlideshowBoxEvent):void {` `  trace("Error loading the configuration XML :: "+evt.text);` `}`
CONFIG_LOAD_OPEN	Dispatched when the loading of the configuration XML starts. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.CONFIG_LOAD_OPEN, configLoadOpen);` `function configLoadOpen(evt:SlideshowBoxEvent):void {` `  trace("Starting to load the configuration XML...");` `}`

CONFIG_LOAD_PROGRESS	Dispatched during the loading process of the configuration XML. **Properties** - *bytesLoaded*:Number – the number of bytes loaded so far of the XML data; - *bytesTotal*:Number – the total size in bytes of the XML data. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.CONFIG_LOAD_PROGRESS, configLoadProgress);` `function configLoadProgress(evt:SlideshowBoxEvent):void { trace("Loading the configuration XML :: "+evt.bytesLoaded+" / "+evt.bytesTotal); }`
CONFIG_SECURITY_ERROR	Dispatched after a SecurityErrorEvent occured during the loading process of the configuration XML data. **Properties** - *text*:String – the error message received from the SecurityErrorEvent object. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.CONFIG_SECURITY_ERROR, configSecurityError);` `function configSecurityError(evt:SlideshowBoxEvent):void { trace("Security error while loading the configuration XML :: "+evt.text); }`
FULLSCREEN_CHANGE	Dispatched whenever the stage exists from full screen mode to normal screen mode or enters from normal screen mode to full screen mode. **Properties** - *fullScreen*:Boolean – a Boolean value that specifies whether the stage has entered into full screen mode (*true*) or into normal screen mode (*false*). **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.FULLSCREEN_CHANGE, fullScreenChange);` `function fullScreenChange(evt:SlideshowBoxEvent):void { if (evt.fullScreen) trace("The album has entered in full screen mode"); else trace("The album has left the full screen mode"); }`
INIT	Dispatched after the album template has been created and fully initialized. At this moment the album template has all its properties set and will start running.

	***Note**: To make the album template use non-default settings before it starts running, we recommend setting the properties on the SlideshowBoxEvent.TEMPLATE_CREATION_DONE event.* **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.INIT, templateInitialized);` `function templateInitialized(evt:SlideshowBoxEvent):void { trace("The album template has been initialized. The total number of images is "+myAlbum.imageList.length); }`
PAGE_CHANGE	Dispatched whenever the thumbnail pages change either from user interaction or during the slide show. **Properties** - *pageIndex*:int – the 0-based index of the currently selected thumbnail page; - *firstIndex*:int – the 0-based index of the first thumbnail in the page; - *lastIndex*:int – the 0-based index of the last thumbnail in the page. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PAGE_CHANGE, thumbsPageChanged);` `function thumbsPageChanged(evt:SlideshowBoxEvent):void { trace("Current thumbnail page :: "+evt.pageIndex); trace("First thumbnail index = "+evt.firstIndex); trace("Last thumbnail index = "+evt.lastIndex); }`
PHOTO_ASYNC_ERROR	Dispatched when there was an exception from native asynchronous code during the loading process of a large image (normal or full screen mode). **Properties** - *text*:String – the error message received from the AsyncErrorEvent object. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_ASYNC_ERROR, photoAsyncError);` `function photoAsyncError(evt:SlideshowBoxEvent):void { trace("Asynchronous error while loading image "+(evt.item.index + 1)+" :: "+evt.text); }`

PHOTO_CLOSE	Dispatched after an opened large image has been close, after the closing tween has finished. This event is dispatched only by the album templates that support opening and closing large images. For other album templates the large images are always opened and visible. **Parameters** - *item*:Object – an object containing information about the image item. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_CLOSE, photoClose);` `function photoClose(evt:SlideshowBoxEvent):void {` `trace("Image "+(evt.item.index + 1)+" has been closed");` `}`
PHOTO_HIDE	Dispatched before the hide transition of a large image (normal or full screen mode) starts. This transition starts because a new image has to been selected to load and display. The hide transition continues with the show transition (*SlideshowBox.PHOTO_SHOW*) that displays the newly selected large image. **Parameters** - *item*:Object – an object containing information about the image item. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_HIDE, photoHide);` `function photoHide(evt:SlideshowBoxEvent):void {` `trace("Hiding image "+(evt.item.index + 1)+"");` `}`
PHOTO_LOAD_COMPLETE	Dispatched after the large image (normal or fulls screen mode) has been completely loaded. **Parameters** - *item*:Object – an object containing information about the image item. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_LOAD_COMPLETE, photoLoadComplete);` `function photoLoadComplete(evt:SlideshowBoxEvent):void {` `trace("Finished loading image "+ (evt.item.index + 1));` `}`
PHOTO_LOAD_ERROR	Dispatched after an IO error occurred during the loading process of a large image (normal or fulls screen mode). **Properties**

	- *text*:String – the error message received from the IOErrorEvent object; - *item*:Object – an object containing information about the image item. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_LOAD_ERROR, photoLoadError);` `function photoLoadError(evt:SlideshowBoxEvent):void {` `  trace("Error loading "+(evt.item.index + 1)+" :: "+evt.text);` `}`
PHOTO_LOAD_OPEN	Dispatched when a new large image (normal or fulls screen mode) starts loading. **Parameters** - *item*:Object – an object containing information about the image item. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_LOAD_OPEN, photoLoadOpen);` `function photoLoadOpen(evt:SlideshowBoxEvent):void {` `  trace("Start loading image "+(evt.item.index + 1));` `}`
PHOTO_LOAD_PROGRESS	Dispatched during the loading process of a large image (normal or full screen mode). **Parameters** - *item*:Object – an object containing information about the image item; - *bytesLoaded*:Number – the number of bytes loaded so far from the image; - *bytesTotal*:Number – the total size in bytes of the image item. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.THUMB_LOAD_PROGRESS, thumbLoadProgress);` `function photoLoadProgress(evt:SlideshowBoxEvent):void {` `  trace("Loading image "+(evt.item.index + 1)+" :: "+evt.bytesLoaded+" / "+evt.bytesTotal);` `}`
PHOTO_OPEN	Dispatched after an opened large image has been close, after the closing tween has finished. This event is dispatched only by the album templates that support opening and closing large images. For other album templates the large images are always opened and visible. **Parameters** - *item*:Object – an object containing information about the image item. **Example**

	`import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_OPEN, photoOpen);` `function photoOpen(evt:SlideshowBoxEvent):void {` `  trace("Opening image "+(evt.item.index + 1));` `}`
PHOTO_SECURITY_ERROR	Dispatched after a SecurityErrorEvent occurred during the loading process of a large image (normal or full screen mode). **Properties** - *text*:String – the error message received from the SecurityErrorEvent object; - *item*:Object – an object containing information about the image item. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_SECURITY_ERROR, photoSecurityError);` `function photoSecurityError(evt:SlideshowBoxEvent):void {` `  trace("Security error while loading image "+(evt.item.index + 1)+" :: "+evt.text);` `}`
PHOTO_SHOW	Dispatched after the show transition of a large image (normal or full screen mode) has ended. The show transition displays a newly selected image and takes place just after the hide transition of the previous image or right after a new image has been opened (*SlideshowBox.PHOTO_OPEN*). Practically, it is dispatched each time a new large image is displayed (either from user selection or from automatic slide show). **Parameters** - *item*:Object – an object containing information about the image item. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.PHOTO_SHOW, photoShow);` `function photoShow(evt:SlideshowBoxEvent):void {` `  trace("Image "+(evt.item.index + 1)+" is now visible");` `  trace(myAlbum.selectedIndex+" :: "+"+myAlbum.selectedImage.title);` `}`
SLIDESHOW_START	Dispatched when the image slide show starts, either from user interaction (pressing the play slide show control) or by code: calling the *startSlideShow()* method or setting the *autoSlideShow* property to *true*. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.SLIDESHOW_START, slideShowStarted);`

	function slideShowStarted(evt:SlideshowBoxEvent):void { trace("The slide show has begun."); }
SLIDESHOW_STOP	Dispatched when the image slide show stops, either from user interaction (pressing the stop slide show control) or by code: calling the *stopSlideShow()* method or setting the *autoSlideShow* property to *false*. **Example** import com.jumpeye.events.SlideshowBoxEvent; myAlbum.addEventListener(SlideshowBoxEvent.SLIDESHOW_STOP, slideShowStopped); function slideShowStopped(evt:SlideshowBoxEvent):void { trace("The slide show has endded."); }
SOURCE_ASYNC_ERROR	Dispatched when there was an exception from native asynchronous code during the loading process of the source XML. **Properties** - *text*:String – the error message received from the AsyncErrorEvent object. **Example** import com.jumpeye.events.SlideshowBoxEvent; myAlbum.addEventListener(SlideshowBoxEvent.SOURCE_ASYNC_ERROR, sourceAsyncError); function sourceAsyncError(evt:SlideshowBoxEvent):void { trace("Asynchronous error while loading the list of images :: "+evt.text); }
SOURCE_LOAD_COMPLETE	Dispatched after the source XML data has completely loaded. **Example** import com.jumpeye.events.SlideshowBoxEvent; myAlbum.addEventListener(SlideshowBoxEvent.SOURCE_LOAD_COMPLETE, sourceLoadComplete); function sourceLoadComplete(evt:SlideshowBoxEvent):void { trace("List of images has completely loaded..."); }
SOURCE_LOAD_ERROR	Dispatched after an IO error occurred during the loading process of the source XML data. **Properties** - *text*:String – the error message received from the IOErrorEvent object.. **Example** import com.jumpeye.events.SlideshowBoxEvent;

	`` myAlbum.addEventListener(SlideshowBoxEvent.SOURCE_LOAD_ERROR, sourceLoadError); function sourceLoadError(evt:SlideshowBoxEvent):void { trace("Error loading the list of images :: "+evt.text); } ```
SOURCE_LOAD_OPEN	Dispatched when the loading of the source XML starts. **Example** ``` import com.jumpeye.events.SlideshowBoxEvent; myAlbum.addEventListener(SlideshowBoxEvent.SOURCE_LOAD_OPEN, sourceLoadOpen); function sourceLoadOpen(evt:SlideshowBoxEvent):void { trace("Starting to load the list of images..."); } ```
SOURCE_LOAD_PROGRESS	Dispatched during the loading process of the source XML. **Properties** - *bytesLoaded*:Number – the number of bytes loaded so far of the XML data; - *bytesTotal*:Number – the total size in bytes of the XML data. **Example** ``` import com.jumpeye.events.SlideshowBoxEvent; myAlbum.addEventListener(SlideshowBoxEvent.SOURCE_LOAD_PROGRESS, sourceLoadProgress); function sourceLoadProgress(evt:SlideshowBoxEvent):void { trace("Loading the list of images :: "+evt.bytesLoaded+" / "+evt.bytesTotal); } ```
SOURCE_SECURITY_ERROR	Dispatched if a SecurityErrorEvent occurred during the loading process of the source XML data. **Properties** - *text*:String – the error message received from the SecurityErrorEvent object. **Example** ``` import com.jumpeye.events.SlideshowBoxEvent; myAlbum.addEventListener(SlideshowBoxEvent.SOURCE_SECURITY_ERROR, sourceSecurityError); function sourceSecurityError(evt:SlideshowBoxEvent):void { trace("Security error while loading the list of images :: "+evt.text); } ```

TEMPLATE_CREATION_DONE	Dispatched right after the SlideshowBox template has been instantiated. This is the moment when you can get a reference to the album template and set the properties with new values, other than the defaults, before the template starts running. ***Note**: To make the album template use non-default settings before it starts running, we recommend setting the properties on the SlideshowBoxEvent.TEMPLATE_CREATION_DONE event.* **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.TEMPLATE_CREATION_DONE, setUpTemplate);` `function setUpTemplate(evt:SlideshowBoxEvent):void {` `trace(myAlbum.templateName+" has been instantiated, template properties can now be set");` `// Set up the album template.` `myAlbum.template.autoSlideShow = false;` `myAlbum.template.backgroundColor = 0xFF0000;` `myAlbum.template.slideShowSpeed = 3;` `myAlbum.template.fullScreenButton = true;` `// You can also retrieve the template's properties at this point.` `trace("Showing image information ? "+myAlbum.template.showImageInfos);` `}`
THUMB_ASYNC_ERROR	Dispatched when there was an exception from native asynchronous code during the loading process of a thumbnail image. **Properties** - *text*:String – the error message received from the AsyncErrorEvent object. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.THUMB_ASYNC_ERROR, thumbAsyncError);` `function thumbAsyncError(evt:SlideshowBoxEvent):void {` `trace("Asynchronous error while loading thumbnail "+(evt.item.index + 1)+" :: "+evt.text);` `}`
THUMB_LOAD_COMPLETE	Dispatched after the thumbnail image has been completely loaded. **Parameters** - *item*:Object – an object containing information about the image item. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.THUMB_LOAD_COMPLETE, thumbLoadComplete);` `function thumbLoadComplete(evt:SlideshowBoxEvent):void {` `trace("Finished loading thumbnail "+`

	(evt.item.index + 1)); }
THUMB_LOAD_ERROR	Dispatched after an IO error occurred during the loading process of a thumbnail image. **Properties** - *text*:String – the error message received from the IOErrorEvent object; - *item*:Object – an object containing information about the image item. **Example** import com.jumpeye.events.SlideshowBoxEvent; myAlbum.addEventListener(SlideshowBoxEvent.THUMB_LOAD_ERROR, thumbLoadError); function thumbLoadError(evt:SlideshowBoxEvent):void { trace("Error loading thumbnail "+ (evt.item.index + 1)+" :: "+evt.text); }
THUMB_LOAD_OPEN	Dispatched when a new thumbnail image starts loading. **Parameters** - *item*:Object – an object containing information about the image item. **Example** import com.jumpeye.events.SlideshowBoxEvent; myAlbum.addEventListener(SlideshowBoxEvent.THUMB_LOAD_OPEN, thumbLoadOpen); function thumbLoadOpen(evt:SlideshowBoxEvent):void { trace("Start loading thumbnail "+ (evt.item.index + 1)); }
THUMB_LOAD_PROGRESS	Dispatched during the loading process of a thumbnail image. **Parameters** - *item*:Object – an object containing information about the image item; - *bytesLoaded*:Number – the number of bytes loaded so far from the thumbnail image; - *bytesTotal*:Number – the total size in bytes of the thumbnail image. **Example** import com.jumpeye.events.SlideshowBoxEvent; myAlbum.addEventListener(SlideshowBoxEvent.THUMB_LOAD_PROGRESS, thumbLoadProgress); function thumbLoadProgress(evt:SlideshowBoxEvent):void { trace("Loading "+(evt.item.index + 1)+" :: "+evt.bytesLoaded+" / "+evt.bytesTotal); }
THUMB_SECURITY_ERROR	Dispatched after a SecurityErrorEvent occurred during the loading process of a thumbnail image.

	Properties - *text*:String – the error message received from the SecurityErrorEvent object; - *item*:Object – an object containing information about the image item. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.THUMB_SECURITY_ERROR, thumbSecurityError);` `function thumbSecurityError(evt:SlideshowBoxEvent):void {` `trace("Security error while loading thumbnail "+(evt.item.index + 1)+" :: "+evt.text);` `}`
THUMB_SELECTED	Dispatched after a new thumbnail has been selected by user interaction, during the slide show or by calling ActionScript code. **Properties** - *index*:int – the 0-based index of the newly selected thumbnail image. **Example** `import com.jumpeye.events.SlideshowBoxEvent;` `myAlbum.addEventListener(SlideshowBoxEvent.THUMB_SELECTED, thumbSelected);` `function thumbSelected(evt:SlideshowBoxEvent):void {` `trace("Thumbnail "+(evt.index + 1)+" has been selected.");` `}`

The Image Item Object Properties

Name	Type	Description
description	String	A description of the image.
fullScreenImagePath	String	The URL for the large image loaded when the album is in full screen mode.
height	Number	The height of the large image when it is displayed in the album. This value depends on the size of the album and it might be less than the real height of the image (*originalHeight*). **Note**: If the large image item for which the object has been requested is not currently displayed in the album, then this property has a value of *NaN*.
index	int	The index of the image associated to the requested image object. The index is 0-based, meaning that the first image has an index of 0.
largeImagePath	String	The URL to the large image loaded when the album is in normal screen

		mode.
originalHeight	Number	The original height of the large image used in the normal screen mode.
originalWidth	Number	The original width of the large image used in the normal screen mode.
thumbnailPath	String	The URL of the thumbnail image used by the album.
title	String	The title of the image.
width	Number	The width of the large image when it is displayed in the album. This value depends on the size of the album and it might be less than the real width of the image (*originalWidth*). **Note**: If the large image item for which the object has been requested is not currently displayed in the album, then this property has a value of *NaN*.
xmlNode	XML	The XML node associated with the image, as it is set in the source XML data.

Note: By adding new XML elements to the <item> element in the source XML file of the album, you can add new properties to the image item object that can be accessed using code, exactly like the existing properties. For details regarding the structure of the source XML file please read The Source XML File.pdf document found in the package.