

AFC

AFCOMPONENTS

A division of Professional Content, LLC

Thumbnail Scroller User Guide



powered by **PRO**
CONTENT

Professional Content LLC

816-298-0495

Kansas City, Missouri

www.procontent.net

Advanced Flash Components

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. Using the component in FLEX	16
9. Properties.....	17
xmlPath	17
items	17
itemsArray	17
itemWidth	18
itemHeight.....	18
currentPosition.....	18
currentPositionEasingStrength.....	18
scaleMode.....	19
direction.....	19
verticalAlign.....	19
horizontalAlign	20
distance	20
cornerRoundness	20
overSkinReference.....	21
inactiveArea	21
handCursor	21
smoothing.....	21
interactiveltems	22
border	22
borderColor	22
borderAlpha	22

Table of Contents

borderThickness.....	23
preloaderType.....	23
preloaderSize.....	23
preloaderFillAlpha.....	24
preloaderLineAlpha.....	24
preloaderFillColor.....	24
preloaderLineColor.....	24
reflection.....	25
reflectionDistance.....	25
reflectionAlpha.....	25
reflectionRatio.....	26
animatedReflection.....	26
useCacheAsBitmap.....	26
temporaryClipReference.....	27
temporaryClipAlpha.....	27
temporaryClipColor.....	27
skipBrokenPath.....	28
mouseScroll.....	28
mouseScrollType.....	28
mouseScrollSpeed.....	29
mouseScrollEasing.....	29
motionBlur.....	29
motionBlurStrength.....	29
motionBlurQuality.....	30
scrollAmount.....	30
scrollDuration.....	30
scrollEasing.....	31
autoScroll.....	31
autoScrollDelay.....	31
autoScrollDirection.....	32
selectOver.....	32
selectedAlign.....	32
effect.....	33
livePreviewItems.....	33

Table of Contents

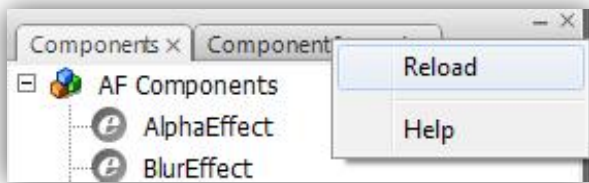
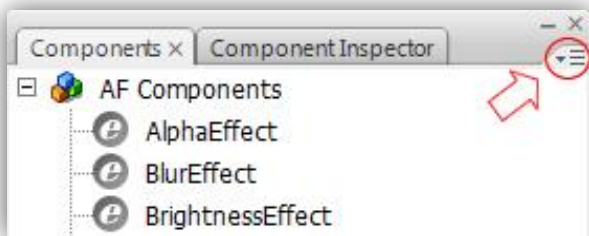
numItems	34
selectedIndex.....	34
isScrolling.....	34
10. Methods.....	35
addItem().....	35
addItemAt()	35
addItems().....	35
addItemsAt()	36
removeItem().....	36
removeItemAt().....	37
removeItemsAt()	37
removeAllItems()	37
replaceItem().....	38
replaceItemAt()	38
spliceItems().....	38
getItemAt()	39
getItemIndex().....	39
select().....	40
deselect()	40
selectNext().....	40
selectPrevious().....	40
scrollRight().....	41
scrollLeft()	41
scrollUp().....	41
scrollDown().....	41
setSize()	42
move()	42
11. Events.....	43
ITEMS_LOAD_START	43
ITEMS_LOAD_PROGRESS	43
ITEMS_LOAD_COMPLETE	43
XML_LOAD_START	44
XML_LOAD_PROGRESS	44

Table of Contents

XML_LOAD_COMPLETE.....	44
XML_LOAD_ERROR.....	45
ITEM_ADD.....	45
ITEM_REMOVE.....	46
ITEM_CLICK.....	46
ITEM_DOUBLE_CLICK.....	47
ITEM_MOUSE_UP.....	47
ITEM_MOUSE_DOWN.....	48
ITEM_MOUSE_OVER.....	48
ITEM_MOUSE_OUT.....	48
ITEM_START.....	49
ITEM_PROGRESS.....	49
ITEM_COMPLETE.....	50
ITEM_ERROR.....	50
ITEM_SELECTED.....	51
ITEM_DESELECTED.....	51
RESIZE.....	52
UPDATE.....	52
SCROLL_START.....	52
SCROLL_PROGRESS.....	53
SCROLL_COMPLETE.....	53
REACHED_TOP.....	53
REACHED_BOTTOM.....	54
REACHED_LEFT.....	54
REACHED_RIGHT.....	54

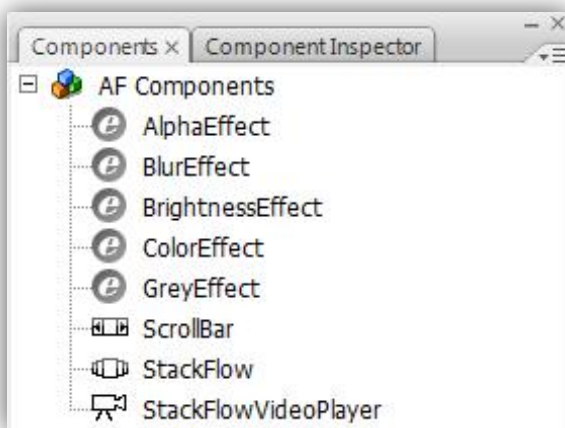
1. *Component Installation*

1. Double-Click on the ThumbnailScroller.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 scroller.fla.
Test the movie in order to create the scroller.swf file.
2. Open the Components panel (Window >Components) and drag an instance of the Thumbnail Scroller component onto the stage. The Thumbnail Scroller component can be found in the AFComponents folder.



This is how the component looks on stage:



As you can see, the component has a live preview which makes it easy for you to preview how the component will be rendered, without having to test the movie. If you chose a different preloader or change any other visual element, you will immediately be able to see those changes.

Bellow 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.

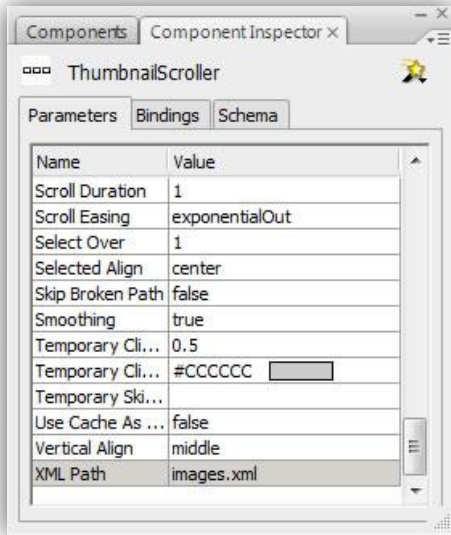
Now we are ready to load some content.

3. In the same folder as scroller.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 scroller.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"?>
<images>
  <item>
    <source>images/image1.jpg</source>
  </item>
  <item>
    <source>images/image2.jpg</source>
  </item>
  <item>
    <source>images/image3.jpg</source>
  </item>
  <item>
    <source>images/image4.jpg</source>
  </item>

  <item>
    <source>images/image5.jpg</source>
  </item>
</images>
```

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".

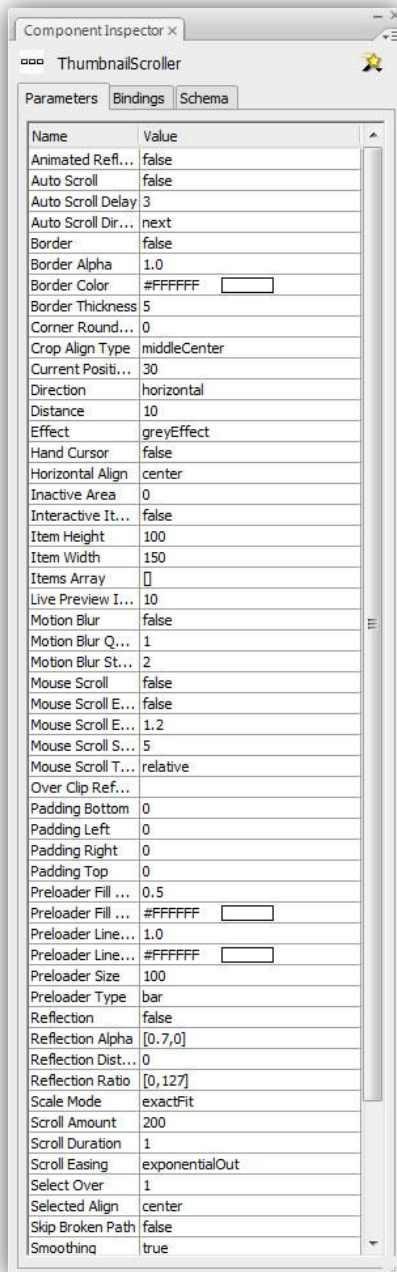


7. You are now ready to test the movie.

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 Actionsript code.



The Component Inspector has 2 columns: Name and Value.

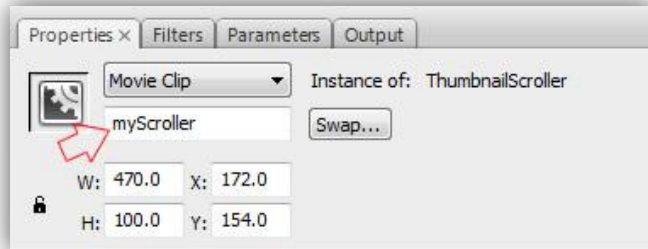
The Name column contains the name of the properties while the Value column contains the values of the properties. As you can see almost all properties have default values.

All these properties can also be changed using Actionsript 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. “myScroller”, 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:

```
myScroller.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);
```

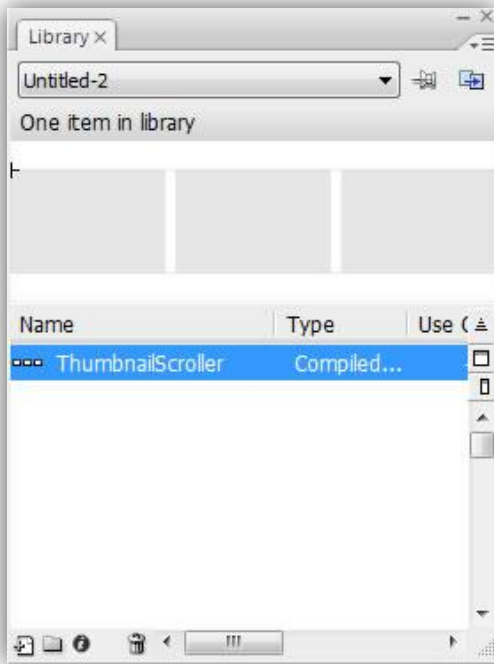
```
function clickHandler(event:MouseEvent):void
{
    myScroller.xmlPath = "images.xml";
}
```

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 learn how to instantiate the component using code as opposed to dragging it onto the stage, which will give you 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, let's add some code.

1. Import the Scroller class.

```
import com.afcomponents.scroller.ThumbnailScroller;
```

2. Instantiate the Thumbnail Scroller class.

```
var myScroller:ThumbnailScroller = new ThumbnailScroller();
```

3. Add the instance to the display list.

```
addChild(myScroller);
```

These 3 lines of code are the equivalent of dragging the component onto the stage and giving it the instance name of “myScroller”. 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 you press a button 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.scroller.ThumbnailScroller;
import flash.events.MouseEvent;

var myScroller:ThumbnailScroller;

myButton.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void
{
    myScroller = new ThumbnailScroller();
    addChild(myScroller);
    myScroller.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"}];
myScroller.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>
  <item>
    <source>images/image0.jpg</source>
    <title>Title</title>
  </item>
  <item>
    <source>images/image1.jpg</source>
    <abc>some data</abc>
  </item>
  <item>
    <source>images/image2.jpg</source>
    <xyz>more data</xyz>
  </item>
  <item>
    <source>images/image3.jpg</source>
  </item>
  <item>
    <source>images/image4.jpg</source>
  </item>
</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"];
myScroller.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 Thumbnail Scroller component allows you to load external files like JPGs, GIFs, PNGs and SWFs but also library assets and Display Objects.

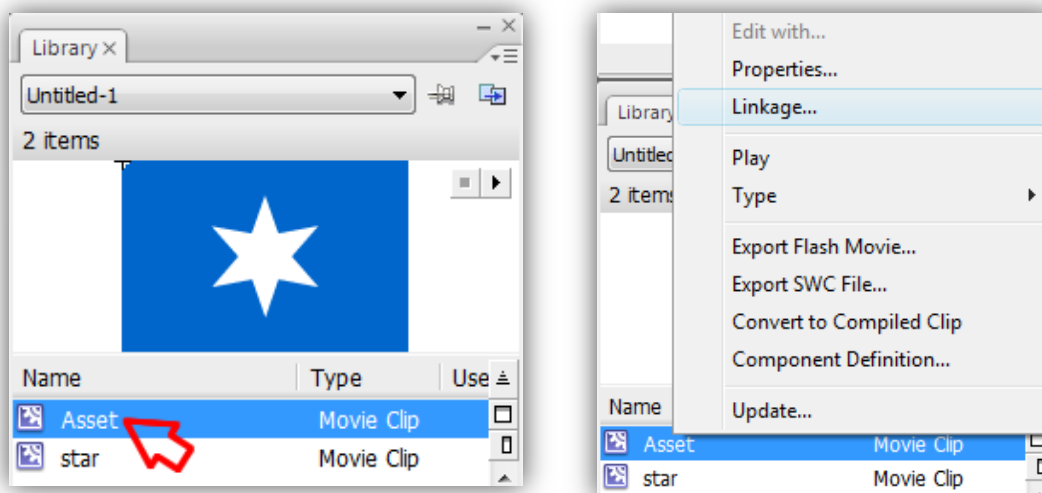
External content

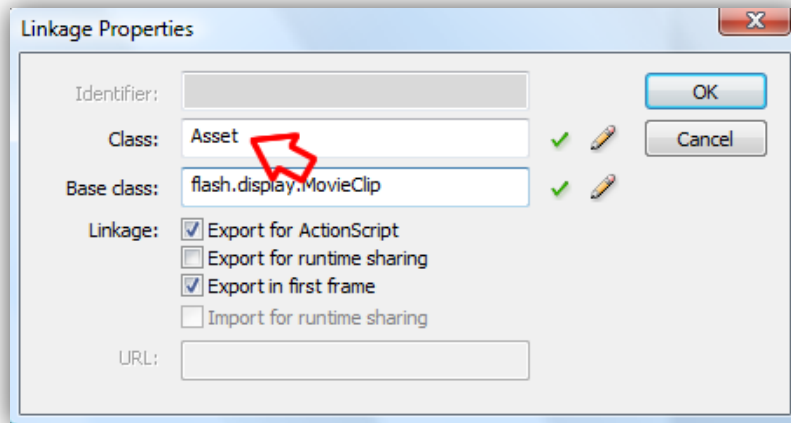
You've seen in the previous examples how to load external content. You simply need to specify the path to the content as the source.

```
myScroller.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"}]

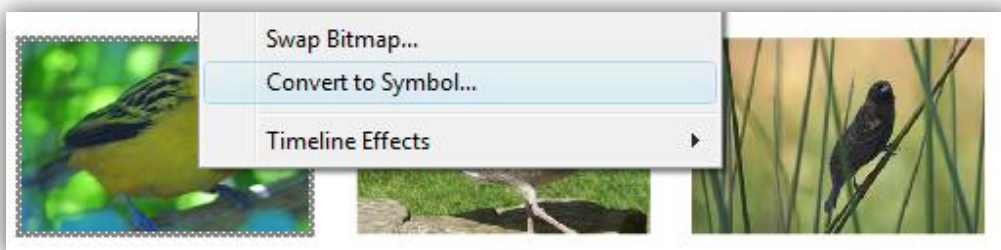
myScroller.items = myItems;
```

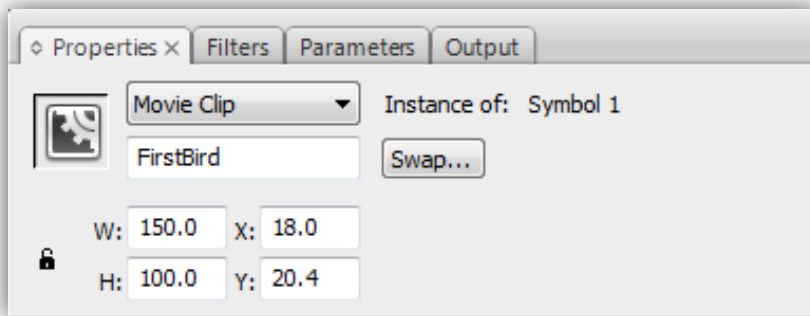
The name of the Class doesn't necessary have to be the same as the name of the symbol but when you specify the source you always use the name of the Class. I could have named the class MyAssetSample, for example, and then the source would be {source:"MyAssetSample"}.

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"}];  
  
myScroller.items = myItems;
```

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();  
  
myScroller.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.

```
<item> <source>rectangle</source></item>
```

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:

```
<item>  
    <source>myRectangle</source>  
</item>
```

8. *Using the component in FLEX*

This component was built for the Flash IDE but it is also compatible with the Flex framework. We have created a wrapper class to make it integrate seamlessly with your Flex application. The wrapper class, named `ComponentWrapper.as`, can be found in the `examples_flex` folder.

You will have to use this wrapper class in order to add the component to a Flex component like the `Panel` or `HBox`.

Example:

```
var myScroller:Scroller = new Scroller();
var container:ComponentWrapper = new ComponentWrapper(myScroller);

var panel:Panel = new Panel();
addChild(panel);

panel.addChild(container);
```

After instantiating the component and adding it to the stage you can simply ignore the wrapper class. If you want to set a property, call a method, or listen for an event, you will only need to reference the `Thumbnail Scroller` instance.

```
myScroller.setSize(700, 200);
myScroller.border = true;
myScroller.borderColor = 0xFF0000;
```

For a better understating about this workflow please see the example from the “`examples_flex`” folder.

9. *Properties*

xmlPath

Component Inspector Name: XML Path.

Type: String.

Default Value: “ ” (empty string).

Description: The location of the XML file.

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

itemWidth

Component Inspector Name: Item Width.

Type: Number.

Default Value: 150.

Description: The maximum width of each item.

Example: `myScroller.itemWidth = 200;`

itemHeight

Component Inspector Name: Item Height.

Type: Number.

Default Value: 100.

Description: The maximum height of each item.

Example: `myScroller.itemWidth = 150;`

currentPosition

Component Inspector Name: Not Available.

Type: Number.

Default Value: 0.

Description: A number between 0 and 1 representing the current horizontal/vertical position of the scroller.

Example: `myScroller.currentPosition = 0.5;`

currentPositionEasingStrength

Component Inspector Name: Current Position Easing Strength.

Type: Number.

Default Value: 30.

Description: Indicates the strength of the easing when the position of the scroller is set using the `currentPosition` property.

Example: `myScroller.currentPositionEasingStrength = 50;`

scaleMode

Component Inspector Name: Scale Mode.

Type: String.

Default Value: "exactFit".

Available Values: "exactFit", "maintainAspectRatio" 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.

Example: `myScroller.scaleMode = "maintainAspectRatio";`

direction

Component Inspector Name: Direction.

Type: String.

Default Value: "horizontal".

Available Values: "horizontal" and "vertical".

Description: The direction of the scroller.

Example: `myScroller.direction = "vertical";`

verticalAlign

Component Inspector Name: Vertical Align.

Type: String.

Default Value: "middle".

Available Values: “middle”, “top” and “bottom”.

Description: The alignment of each item when the direction is set to “horizontal”. It is not important when direction is set to “vertical”.

Example: `myScroller.verticalAlign = “bottom”;`

horizontalAlign

Component Inspector Name: Horizontal Align.

Type: String.

Default Value: “center”.

Available Values: “center”, “left” and “right”.

Description: The alignment of each item when the direction is set to “vertical”. It has no importance when direction is set to “horizontal”.

Example: `myScroller.horizontalAlign = “left”;`

distance

Component Inspector Name: Distance.

Type: Number.

Default Value: 10.

Description: The distance between items.

Example: `myScroller.distance = 25;`

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: `myScroller.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: `myScroller.overSkinReference = "Gloss";`

inactiveArea

Component Inspector Name: Inactive Area.

Type: Number.

Default Value: 0.

Description: The width/height of the area in which the mouse scrolling will be inactive.

Example: `myScroller.inactiveArea = 150;`

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: `myScroller.handCursor = true;`

smoothing

Component Inspector Name: Smoothing.

Type: Boolean.

Default Value: true.

Description: Indicates whether the images will be smoothed.

Example: `myScroller.smoothing = false;`

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: `myScroller.interactiveltems = true;`

border

Component Inspector Name: Border.

Type: Boolean.

Default Value: false.

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

Example: `myScroller.border = true;`

borderColor

Component Inspector Name: Border Color.

Type: uint.

Default Value: 0xFFFFFFFF.

Description: The color of the border.

Example: `myScroller.borderColor = 0xFF00FF;`

borderAlpha

Component Inspector Name: Border Alpha.

Type: Number.

Default Value: 1.0.

Description: The alpha of the border.

Example: `myScroller.borderAlpha = 0.5;`

borderThickness

Component Inspector Name: Border Thickness.

Type: Number.

Default Value: 5.

Description: The thickness of the border.

Example: `myScroller.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: `myScroller.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: `myScroller.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: `myScroller.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: `myScroller.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: `myScroller.preloaderFillColor = 0x0000FF;`

preloaderLineColor

Component Inspector Name: Preloader Line Color.

Type: uint.

Default Value: 0xFFFFFFFF.

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

Example: `myScroller.preloaderLineColor = 0xFF0000;`

reflection

Component Inspector Name: Reflection.

Type: Boolean.

Default Value: false.

Description: Indicates whether a reflection will be added for the scroller.

Example: `myScroller.reflection = true;`

reflectionDistance

Component Inspector Name: Reflection Distance.

Type: Number.

Default Value: 0.

Description: The distance of the reflection from the bottom edge of the scroller.

Example: `myScroller.reflectionDistance = 5;`

reflectionAlpha

Component Inspector Name: Reflection Alpha.

Type: Array of numbers.

Default Value: [0.7, 0].

Description: An array of two numbers representing the alpha values at the beginning and at the end of the reflection.

Example: `myScroller.reflectionAlpha = [1.0, 0.0];`

reflectionRatio

Component Inspector Name: Reflection Ratio.

Type: Array of numbers.

Default Value: [0, 127].

Description: An array of two numbers representing the ratio values of the reflection.

Example: `myScroller.reflectionRatio = [0, 255];`

animatedReflection

Component Inspector Name: Animated Reflection.

Type: Boolean.

Default Value: false.

Description: Indicates whether the reflection will be animated. If the scroller has some animated content and you want the reflection to constantly refresh so that the animation will be visible in the reflection too, you need to set this property to true.

Example: `myScroller.animatedReflection = true;`

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: `myScroller.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: myScroller.temporaryClipReference = "TempClip";

temporaryClipAlpha

Component Inspector Name: Temporary Clip Alpha.

Type: Number.

Default Value: 0.5.

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

Example: myScroller.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: myScroller.temporaryClipColor = 0x000000;

skipBrokenPath

Component Inspector Name: Skip Broken Path.

Type: Boolean.

Default Value: false.

Description: Indicates whether the component will ignore invalid sources when external content is loaded.

Example: `myScroller.skipBrokenPath = true;`

mouseScroll

Component Inspector Name: Mouse Scroll.

Type: Boolean.

Default Value: false.

Description: Indicates whether the component's items will be scrolled when the mouse rolls over the component.

Example: `myScroller.mouseScroll = true;`

mouseScrollType

Component Inspector Name: Mouse Scroll Type.

Type: String.

Default Value: "relative".

Available values: "relative" and "absolute".

Description: "relative" will scroll the items based on the pointer's position and the total number of items while "absolute" will scroll the items based only on the pointer's position.

Example: `myScroller.mouseScrollType = "absolute";`

mouseScrollSpeed

Component Inspector Name: Mouse Scroll Speed.

Type: Number.

Default Value: 5.

Description: The speed of the scrolling when mouseScroll is set to true. A higher value indicates a higher speed.

Example: `myScroller.mouseScrollSpeed = 30;`

mouseScrollEasing

Component Inspector Name: Mouse Scroll Easing.

Type: Boolean.

Default Value: false.

Description: Indicates whether easing will be used when the scroller stops after the user moves the mouse away from the component.

Example: `myScroller.mouseScrollEasing = true;`

motionBlur

Component Inspector Name: Motion Blur.

Type: Boolean.

Default Value: false.

Description: Indicates whether the component will be blurred during the scrolling process.

Example: `myScroller.motionBlur = true;`

motionBlurStrength

Component Inspector Name: Motion Blur Strength.

Type: Number

Default Value: 2.

Description: Indicates the strength of the blur if motionBlur is set to true.

Example: myScroller.motionBlurStrength = 4;

motionBlurQuality

Component Inspector Name: Motion Blur Quality.

Type: Number

Default Value: 1.

Description: Indicates the quality of the blur if motionBlur is set to true.

Example: myScroller.motionBlurQuality = 4;

scrollAmount

Component Inspector Name: Scroll Amount.

Type: Number.

Default Value: 200.

Description: The amount of scrolling when scrollRight(), scrollLeft(), scrollUp(), scrollDown() methods are used.

Example: myScroller.scrollAmount = 100;

scrollDuration

Component Inspector Name: Scroll Duration.

Type: Number.

Default Value: 1.

Description: The scroll duration in seconds.

Example: myScroller.scrollDuration = 3;

scrollEasing

Component Inspector Name: Scroll Easing.

Type: String.

Default Value: "exponentialOut".

Available values: "linear", "backIn", "backOut", "backInOut", "bounceIn", "bounceOut", "bounceInOut", "circularIn", "circularOut", "circularInOut", "cubicIn", "cubicOut", "cubicOut", "elasticIn", "elasticOut", "elasticInOut", "exponentialIn", "exponentialOut", "exponentialInOut", "quadraticIn", "quadraticOut", "quadraticInOut", "quarticIn", "quarticOut", "quarticInOut", "quinticIn", "quinticOut", "quinticInOut", "sineIn", "sineOut", "sineInOut"

Description: The scroll easing.

Example: myScroller.scrollEasing = "backOut";

autoScroll

Component Inspector Name: Auto Scroll.

Type: Boolean.

Default Value: false.

Description: Indicates whether the component's items will be scrolled automatically.

Example: myScroller.autoScroll = true;

autoScrollDelay

Component Inspector Name: Auto Scroll Delay.

Type: Number.

Default Value: 3.

Description: Indicates the delay of the auto scroll, in seconds.

Example: myScroller.autoScrollDelay = 4;

autoScrollDirection

Component Inspector Name: Auto Scroll Direction.

Type: String.

Default Value: "next".

Available Values: "next" and "previous".

Description: Indicates the direction in which the items will be scrolled.

Example: `myScroller.autoScrollDirection = "previous";`

selectOver

Component Inspector Name: Select Over.

Type: uint.

Default Value: 1.

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

Example: `myScroller.selectOver = 3;`

selectedAlign

Component Inspector Name: Selected Align.

Type: String.

Default Value: "center".

Available Values: "center", "left", "right", "middle", "top", "bottom" and "none".

Description: Indicates the alignment of the selected item. "center", "left", "right" need to be used when direction is set to "horizontal" and "middle", "top", "bottom" when direction is set to "vertical".

Example: `myScroller.selectedAlign = "left";`

effect

Component Inspector Name: Effect.

Type: Object.

Default Value: null.

Description: An instance of an Effect component.

Example:

```
var greyEffect:GreyEffect = new GreyEffect();
```

```
myScroller.effect = greyEffect; or myScroller.effect = "greyEffect";
```

livePreviewItems

Component Inspector Name: Live Preview Items.

Type: uint.

Default Value: 10.

Description: The number of items used for the live preview.

*Read-only properties***numItems**

Type: uint.

Description: Returns the total number of items.

Example: `trace(myScroller.numItems);`

selectedIndex

Type: int.

Description: Returns the index of the selected item.

Example: `trace(myScroller.selectedIndex);`

isScrolling

Type: Boolean.

Description: Indicates whether the component is currently scrolling.

Example: `trace(myScroller.isScrolling);`

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:

```
myScroller.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:

```
myScroller.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"}];

myScroller.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"}];

myScroller.addItem(myItems, 5);
```

removeItem()

Implementation: removeItem(item:IDisplayItem):void

Description: Removes the specified item.

Parameters: item - The item to be removed.

Example:

```
myScroller.removeItem(myScroller.getItemAt(4));
```

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:

```
myScroller.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:

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

removeAllItems()

Implementation: removeAllItems():void

Description: Removes all items.

Example:

```
myScroller.removeAllItems();
```

replaceltem()

Implementation: replaceltem(data:Object, item:IDisplayItem)

Description: Replace an existing item with a new item.

Parameters: data - The data of the new item.

item - The item to be replaced.

Example:

```
var data:Object = {source:"images/image1.jpg", desc:"Bird"};
myScroller.replaceltem(data, myScroller.getItemAt(5));
```

replaceltemAt()

Implementation: replaceltemAt(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"};
myScroller.replaceltem(data, 2);
```

spliceltems()

Implementation:

spliceltems(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"}];  
  
myScroller.spliceItems(0, 3, newItems);
```

getItemAt()

Implementation: getItemAt(index:uint):IDisplayItem

Description: Returns the item at the specified index.

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

Example:

```
trace(myScroller.getItemAt(2).index);// output 2
```

getItemIndex()

Implementation: getItemIndex(item:IDisplayItem):int

Description: Returns the index of the specified item.

Parameters: item - The item to be located.

Example:

```
var nr:int = myScroller.getItemIndex(myScroller.getItemAt(7));  
  
trace(nr); // output 7
```

select()

Implementation: select(index:uint):void

Description: Selects the item at the specified index.

Parameter: index – The index of the item to be selected.

Example:

```
myScroller.select(5);
```

deselect()

Implementation: select(index:uint):void

Description: Deselects the item at the specified index.

Parameter: index – The index of the item to be deselected.

Example:

```
myScroller.deselect(myScroller.selectedIndex);
```

selectNext()

Implementation: selectNext():void

Description: Selects the next item.

Example:

```
myScroller.selectNext();
```

selectPrevious()

Implementation: selectPrevious():void

Description: Selects the previous item.

Example:

```
myScroller.selectPrevious();
```

scrollRight()

Implementation: scrollRight():void

Description: Scrolls the component to the right by a certain amount, given by the scrollAmount property.

Example:

```
myScroller.scrollRight();
```

scrollLeft()

Implementation: scrollLeft():void

Description: Scrolls the component to the left by a certain amount, given by the scrollAmount property.

Example:

```
myScroller.scrollLeft();
```

scrollUp()

Implementation: scrollUp():void

Description: Scrolls the component to the top by a certain amount, given by the scrollAmount property.

Example:

```
myScroller.scrollUp();
```

scrollDown()

Implementation: scrollDown():void

Description: Scrolls the component to the bottom by a certain amount, given by the scrollAmount property.

Example:

```
myScroller.scrollDown();
```

setSize()

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

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

Example:

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

move()

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

Description: Moves the component at the specified position.

Example:

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

11. Events

ITEMS_LOAD_START

Description: Dispatched when the loading of a group of items begins.

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEMS_LOAD_START, eventHandler)
function eventHandler(event:ScrollerEvent):void
{
    trace("loading begins");
}
```

ITEMS_LOAD_PROGRESS

Description: Dispatched every time an items was loaded, during the loading of a group of items.

Special Properties: itemsLoaded – The number of loaded items.

itemsTotal – The total number of items to be loaded.

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEMS_LOAD_PROGRESS,eventHandler)
function eventHandler(event:ScrollerEvent):void
{
    trace("Loaded " + event.itemsLoaded + " out of " + event.itemsTotal);
}
```

ITEMS_LOAD_COMPLETE

Description: Dispatched when the loading of a group of items is complete.

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEMS_LOAD_COMPLETE,eventHandler)
```

```
function eventHandler(event:ScrollerEvent):void
{
    trace("loading complete");
}
```

XML_LOAD_START

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

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.XML_LOAD_START, eventHandler);
function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.XML_LOAD_PROGRESS, eventHandler);
function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
```

```
myScroller.addEventListener(ScrollerEvent.XML_LOAD_COMPLETE, eventHandler)
function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.XML_LOAD_ERROR, eventHandler)
function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_ADD, eventHandler);

function eventHandler(event:ScrollerEvent):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"
```

```
}  
myScroller.addItemAt({source:"images/image1.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.ScrollerEvent;  
myScroller.addEventListener(ScrollerEvent.ITEM_REMOVE, eventHandler);  
  
function eventHandler(event:ScrollerEvent):void  
{  
    trace(event.index); // output 1  
}  
myScroller.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.ScrollerEvent;  
myScroller.addEventListener(ScrollerEvent.ITEM_CLICK, eventHandler);
```

```
function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_DOUBLE_CLICK, eventHandler);
```

```
function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_MOUSE_UP, eventHandler);
```

```
function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_MOUSE_DOWN, eventHandler);

function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_MOUSE_OVER, eventHandler);

function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_MOUSE_OUT, eventHandler);

function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_START, eventHandler);

function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_PROGRESS, eventHandler);
```

```
function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_COMPLETE, eventHandler);
```

```
function eventHandler(event:ScrollerEvent):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.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_ERROR, eventHandler);

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

ITEM_SELECTED

Description: Dispatched when an item was selected.

Special Properties: item – The item which was selected.

index – The index of the item.

data – The data of the item.

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_SELECTED, eventHandler);

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

ITEM_DESELECTED

Description: Dispatched when an item was deselected.

Special Properties: item – The item on which was deselected.

index – The index of the item.

data – The data of the item.

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.ITEM_DESELECTED, eventHandler);
```

```
function eventHandler(event:ScrollerEvent):void
{
    trace("Index of the deselected item: " + event.index);
}
```

RESIZE

Description: Dispatched when the size of the component changes.

Example:

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

SCROLL_START

Description: Dispatched when the scrolling starts.

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.SCROLL_START, eventHandler);
```

```
function eventHandler(event:ScrollerEvent):void
{
    trace("scroll started");
}
```

SCROLL_PROGRESS

Description: Dispatched during the scrolling process.

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.SCROLL_PROGRESS, eventHandler);

function eventHandler(event:ScrollerEvent):void
{
    trace("is scrolling");
}
```

SCROLL_COMPLETE

Description: Dispatched when the scrolling is finished.

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.SCROLL_COMPLETE, eventHandler);

function eventHandler(event:ScrollerEvent):void
{
    trace("scroll complete");
}
```

REACHED_TOP

Description: Dispatched when the component has reached the top edge after scrollUp() method was called.

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.REACHED_TOP, eventHandler);
```

```
function eventHandler(event:ScrollerEvent):void
{
    trace("reached top");
}
```

REACHED_BOTTOM

Description: Dispatched when the component has reached the top edge after scrollDown() method was called.

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.REACHED_BOTTOM, eventHandler);
function eventHandler(event:ScrollerEvent):void
{
    trace("reached bottom");
}
```

REACHED_LEFT

Description: Dispatched when the component has reached the left edge after scrollLeft() method was called.

Example:

```
import com.afcomponents.events.ScrollerEvent;
myScroller.addEventListener(ScrollerEvent.REACHED_LEFT, eventHandler);
function eventHandler(event:ScrollerEvent):void
{
    trace("reached left");
}
```

REACHED_RIGHT

Description: Dispatched when the component has reached the right edge after scrollRight() method was called.

Example:

```
import com.afcomponents.events.ScrollerEvent;
```

```
myScroller.addEventListener(ScrollerEvent.REACHED_RIGHT, eventHandler);  
function eventHandler(event:ScrollerEvent):void  
{  
    trace("reached right");  
}
```