



AFCOMPONENTS

A division of Professional Content, LLC

3D Wall

3D Wall User Guide



Professional Content LLC

816-298-0495

Kansas City, Missouri

www.procontent.net

Advanced Flash Components

www.afcomponents.com

6/11/2010

powered by **PRO
CONTENT**

This page left
intentionally blank

Table of Contents

1. Component Installation	1
2. Getting Started	2
3. Using the Component Inspector	6
4. Using Actionscript to set the properties.....	7
5. Instantiate the component using code	9
6. Loading content using Actionscript	11
7. Types of content.....	13
External content.....	13
Library assets	13
Display Objects (Movie Clip, Bitmap, Sprite)	14
8. Using the component in FLEX	17
9. Properties.....	18
xmlPath	18
items	18
itemsArray	18
itemWidth	19
itemHeight.....	19
currentPosition.....	19
currentPositionEasingStrength.....	19
scaleMode.....	20
direction.....	20
layers	20
cornerRoundness	21
overSkinReference.....	21
verticalAlign.....	21
horizontalAlign	22
horizontalDistance.....	22
verticalDistance	22
handCursor	23
smoothing.....	23
interactiveltems	23
border	23
borderColor	24

Table of Contents

borderAlpha	24
borderThickness	24
preloaderType.....	25
preloaderSize.....	25
preloaderFillAlpha	25
preloaderLineAlpha	25
preloaderFillColor	26
preloaderLineColor.....	26
zoomItemPreloaderType	26
zoomItemPreloaderSize	27
zoomItemPreloaderFillColor.....	27
zoomItemFadeDuration	27
reflection.....	27
reflectionDistance	28
reflectionAlpha	28
reflectionRatio.....	28
quality	28
cameraZoom.....	29
cameraFocus	29
cameraDepth	29
cameraAcceleration	30
temporaryClipReference.....	30
temporaryClipAlpha.....	30
temporaryClipColor	31
skipBrokenPath	31
selectedItemDepth	31
selectedItemMoveDuration.....	31
selectedItemMoveEasing.....	32
selectEasingStrength	32
dragScroll	32
dragScrollSpeed	33
dragScrollEasingStrength.....	33
dragScrollReversed.....	33
scrollAmount	33

Table of Contents

scrollDuration.....	34
scrollEasing.....	34
zoomDuration	34
zoomEasing	35
autoScroll	35
autoScrollDelay	36
autoScrollDirection	36
zoomAutoScroll	36
zoomAutoScrollDelay.....	36
zoomAutoScrollDirection	37
zoomItemScale.....	37
zoomItemScaleMode	37
useSmoothing.....	38
usePrecision	38
selectOver	38
zoomOver.....	39
effect.....	39
videoPlayer	39
livePreviewItems.....	40
10. Methods.....	41
addItem().....	41
addItemAt()	41
addItem()	41
addItemAt()	42
removeItem().....	42
removeItemAt().....	42
removeItemsAt()	43
removeAllItems()	43
replaceItem().....	43
replaceItemAt()	44
spliceItems().....	44
getItemAt()	45
getItemIndex().....	45

Table of Contents

select()	45
deselect()	46
selectNext()	46
selectPrevious()	46
scrollRight()	46
scrollLeft()	47
scrollUp()	47
scrollDown()	47
zoom()	47
zoomOut()	48
zoomNext()	48
zoomPrevious()	48
setSize()	48
move()	48
11. Events	50
ITEMS_LOAD_START	50
ITEMS_LOAD_PROGRESS	50
ITEMS_LOAD_COMPLETE	50
XML_LOAD_START	51
XML_LOAD_PROGRESS	51
XML_LOAD_COMPLETE	51
XML_LOAD_ERROR	52
ITEM_ADD	52
ITEM_REMOVE	53
ITEM_CLICK	53
ITEM_DOUBLE_CLICK	54
ITEM_MOUSE_UP	54
ITEM_MOUSE_DOWN	54
ITEM_MOUSE_OVER	55
ITEM_MOUSE_OUT	55
ITEM_START	56
ITEM_PROGRESS	56
ITEM_COMPLETE	57

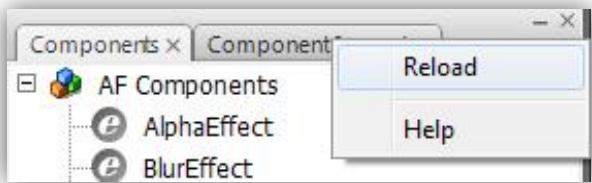
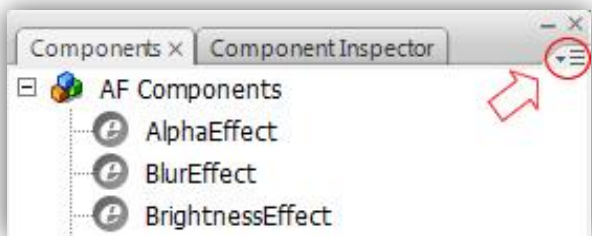
Table of Contents

ITEM_ERROR.....	57
ZOOM_ITEM_CLICK.....	58
ZOOM_ITEM_DOUBLE_CLICK.....	58
ZOOM_ITEM_MOUSE_UP.....	58
ZOOM_ITEM_MOUSE_DOWN.....	59
ZOOM_ITEM_MOUSE_OVER.....	59
ZOOM_ITEM_MOUSE_OUT.....	60
ZOOM_ITEM_START.....	60
ZOOM_ITEM_PROGRESS.....	61
ZOOM_ITEM_COMPLETE.....	61
ZOOM_ITEM_ERROR.....	62
ITEM_SELECTED.....	62
ITEM_DESELECTED.....	63
RESIZE.....	63
UPDATE.....	64
SCROLL_START.....	64
SCROLL_PROGRESS.....	64
SCROLL_COMPLETE.....	65
ZOOM_START.....	65
ZOOM_PROGRESS.....	65
ZOOM_COMPLETE.....	66
ZOOM_OUT_START.....	66
ZOOM_OUT_PROGRESS.....	66
ZOOM_OUT_COMPLETE.....	67
DISPLAY_ZOOM_ITEM.....	67
OUTSIDE_CLICK.....	67
OUTSIDE_DOUBLE_CLICK.....	68
OUTSIDE_MOUSE_UP.....	68
OUTSIDE_MOUSE_DOWN.....	68
REACHED_TOP.....	69
REACHED_BOTTOM.....	69
REACHED_LEFT.....	69
REACHED_RIGHT.....	70

Table of Contents

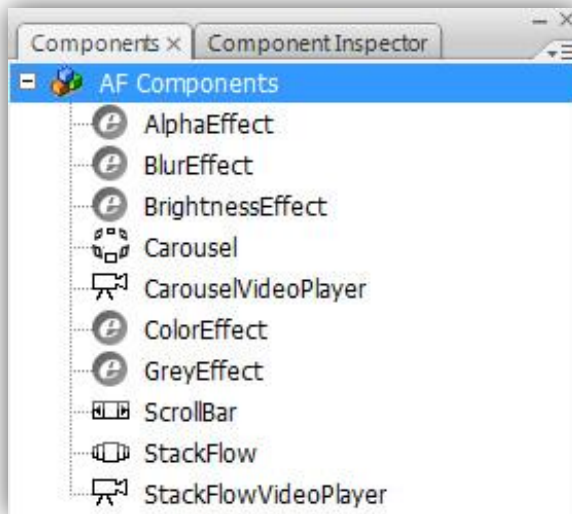
1. *Component Installation*

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

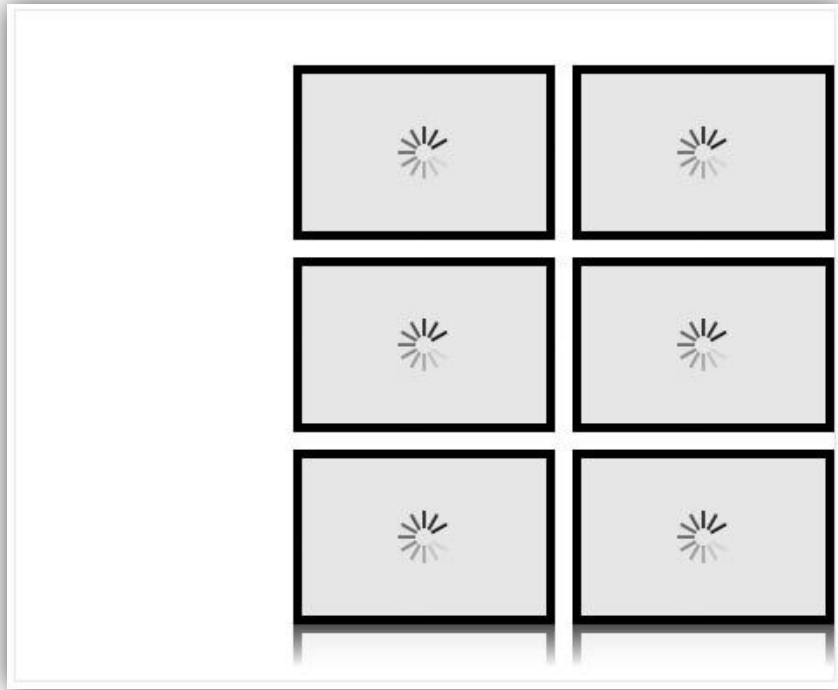


This is how the component looks on stage:



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 wall.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 wall.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"?>
<items>
  <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>
</items>

```

You can also use the “zoom” attribute to specify the location of an image that will be displayed when you zoom over an item. For example you can use a lower resolution of an image to be displayed in the wall and when the user zooms in on that item, display the higher resolution of the same image. The XML file for this example would look like this:

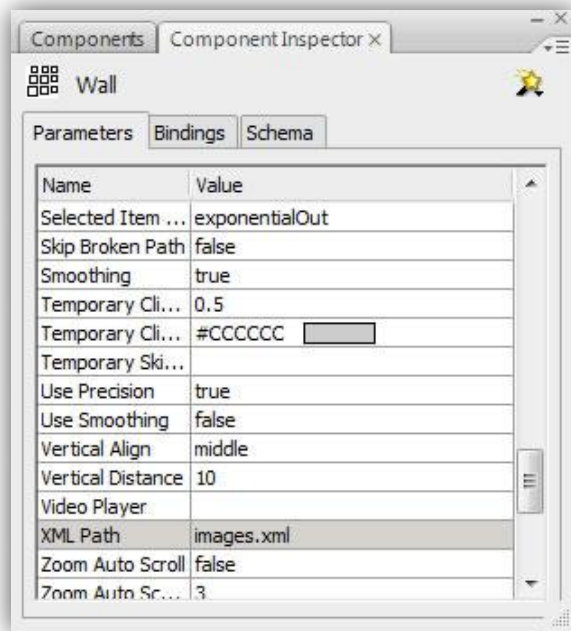
```

<?xml version="1.0" encoding="utf-8"?>
<items>
  <item>
    <source>images/image1.jpg</source>
    <zoom>hq_images/image1.jpg</zoom>
  </item>
  <item>
    <source>images/image2.jpg</source>
    <zoom>hq_images/image2.jpg</zoom>
  </item>
  <item>
    <source>images/image3.jpg</source>
    <zoom>hq_images/image6.jpg</zoom>
  </item>
  <item>
    <source>images/image4.jpg</source>
    <zoom>hq_images/image4.jpg</zoom>
  </item>
  <item>
    <source>images/image5.jpg</source>
    <zoom>hq_images/image5.jpg</zoom>
  </item>
</items>

```

Another attribute that you can use is “video”. If you have the Video version of this component, the “video” attribute allows you to specify the path of a video file or the id of a YouTube video. More information about this feature you can find in the Wall Video Player’s user guide.

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”.

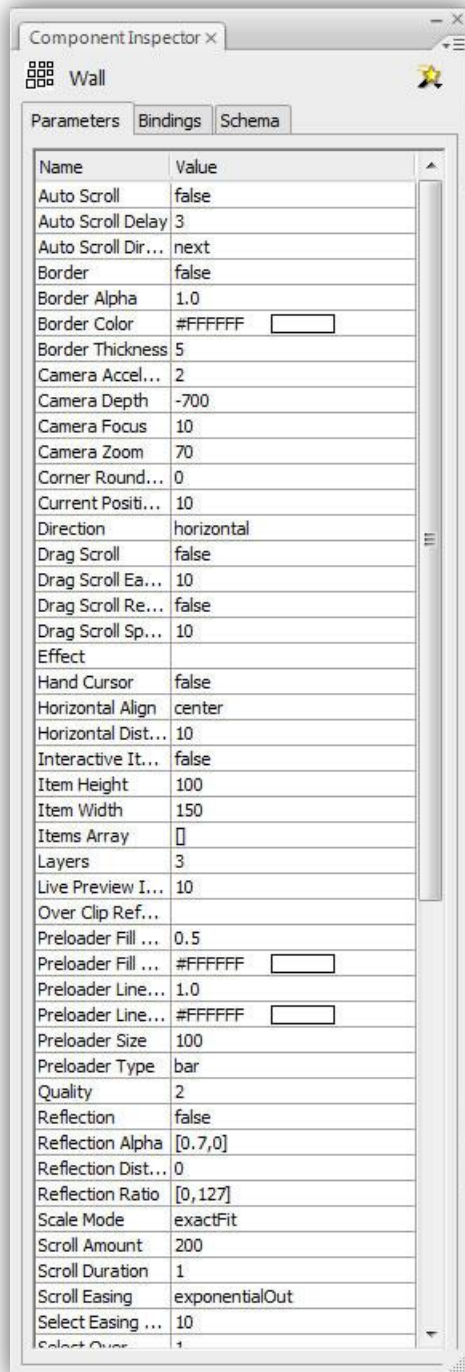


7. You are now ready to test the movie.

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

3. Using the Component Inspector

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



The Component Inspector has 2 columns: Name and Value.

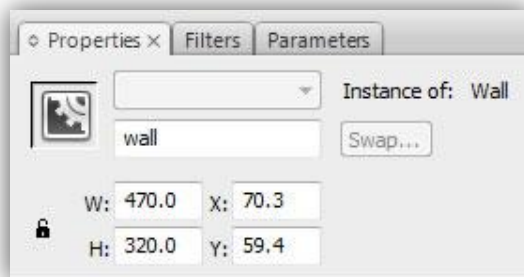
The Name column contains the name of the properties while the Value column contains the values of the properties.

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

4. *Using Actionscript to set the properties*

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

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



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



If you want to specify the path to the XML file using Actionscript, all the code you need to write is this:

```
myWall.xmlPath = "images.xml"
```

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);

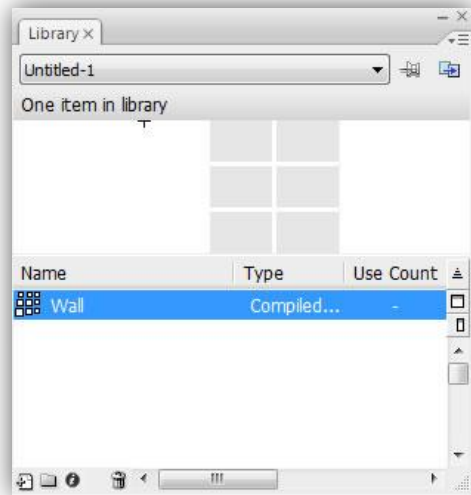
function clickHandler(event:MouseEvent):void
{
    myWall.xmlPath = "images.xml";
}
```

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, we are ready to start coding.

1. Import the Wall class.

```
import com.afcomponents.wall.Wall;
```

2. Instantiate the Wall class.

```
var myWall:Wall = new Wall();
```

3. Add the instance to the display list.

```
addChild(myWall);
```

These 3 lines of code are the equivalent of dragging the component onto the stage and giving it the instance name of “myWall”. 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.

Create a MovieClip symbol and give it the instance name of “myButton”. Then, in the actionscript layer, add the following code:

```
import com.afcomponents.wall.Wall;
import flash.events.MouseEvent;

var myWall:Wall;

myButton.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void
{
    myWall = new Wall();
    addChild(myWall);
    myWall.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"}];  
myWall.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:

```
<images>
  <item>
    <source>images/image1.jpg</source>
    <title>Title 1</title>
  </item>
  <item>
    <source>images/image2.jpg</source>
    <abc>some data</abc>
  </item>
  <item>
    <source>images/image3.jpg</source>
    <xyz>more data</xyz>
  </item>
  <item>
    <source>images/image4.jpg</source>
  </item>
  <item>
    <source>images/image5.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"].

myWall.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 isn't.

7. Types of content

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

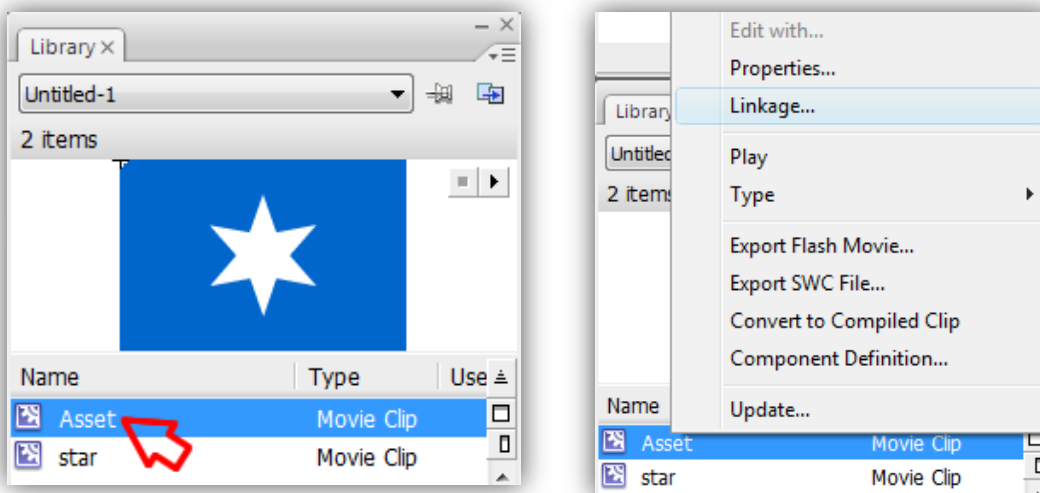
External content

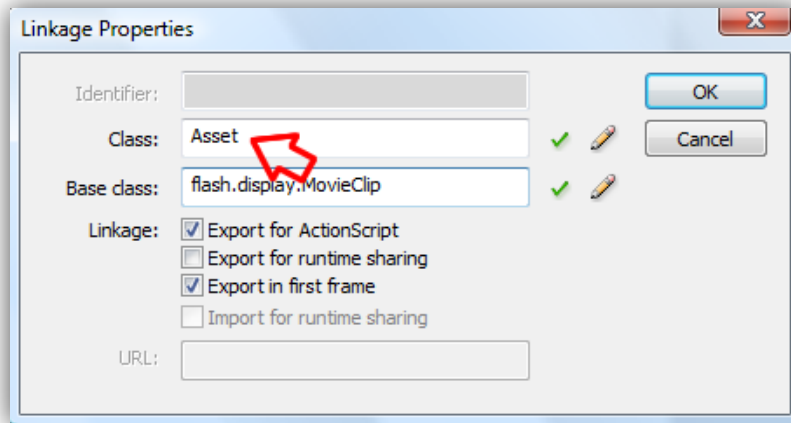
You've seen in the previous examples how to load external content. You simply need to specify the path to the content as the source.

```
myWall.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.





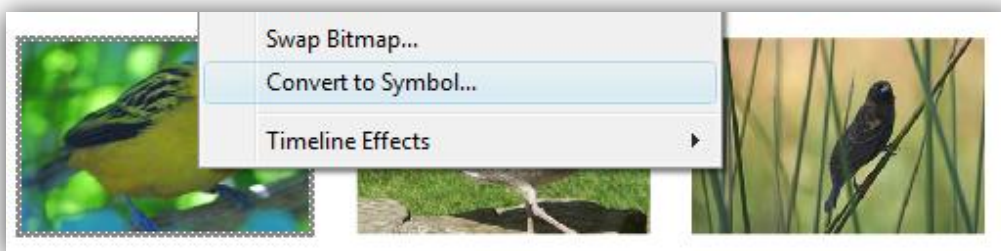
```
var myItems:Array = [{source:"Asset"},           {source:" images/image0.jpg"}, {source:" images/image1.jpg"}]
myWall.items = myItems;
```

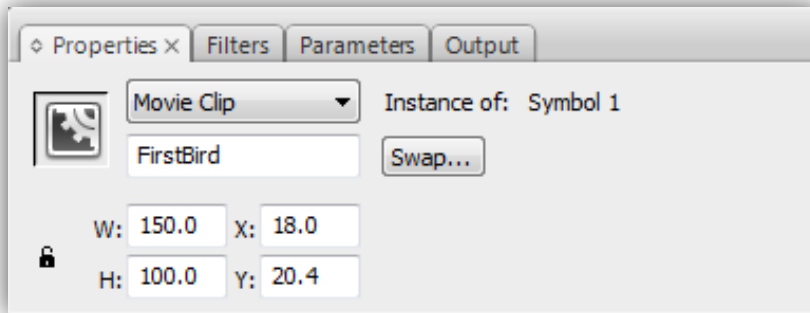
The name of the Class doesn't necessary have to be the same as the name of the symbol but when you specify the source you always use the name of the Class. I could have named the class MyAssetSample, for example, and then the source would be {source:"MyAssetSample"}.

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"}];  
  
myWall.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();  
  
myWall.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 myWall:Wall = new Wall();
var container:ComponentWrapper = new ComponentWrapper(myWall);

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 `Wall` instance.

```
myWall.setSize(700, 200);
myWall.border = true;
myWall.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: myGrid3D.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: myGrid3D.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: myGrid3D.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: myGrid3D.itemWidth = 200;

itemHeight

Component Inspector Name: Item Height.

Type: Number.

Default Value: 100.

Description: The maximum height of each item.

Example: myGrid3D.itemWidth = 150;

currentPosition

Component Inspector Name: Not Available.

Type: Number.

Default Value: 0.

Description: A number between 0 and 1 representing the current position of the grid.

Example: myGrid3D.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 grid is set using the `currentPosition` property.

Example: `myGrid3D.currentPositionEasingStrength = 50;`

scaleMode

Component Inspector Name: Scale Mode.

Type: String.

Default Value: "exactFit".

Available Values: "exactFit" and "maintainAspectRatio";

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.

Example: `myGrid3D.scaleMode = "maintainAspectRatio";`

direction

Component Inspector Name: Direction.

Type: String.

Default Value: "horizontal".

Available Values: "horizontal" and "vertical".

Description: The direction of the grid.

Example: `myGrid3D.direction = "vertical";`

layers

Component Inspector Name: Layers.

Type: uint.

Default Value: 3.

Description: The number of layers represents the number of rows when direction is set to “horizontal” or the number of columns when direction is set to “vertical”.

Example: myGrid3D.layers = 4;

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: myGrid3D.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: myGrid3D.overSkinReference = “Gloss”;

verticalAlign

Component Inspector Name: Vertical Align.

Type: String.

Default Value: “middle”.

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

Description: The vertical alignment of each item when the direction is set to “horizontal”.

Example: `myGrid3D.verticalAlign = "bottom";`

horizontalAlign

Component Inspector Name: Horizontal Align.

Type: String.

Default Value: "center".

Available Values: "center", "left" and "right".

Description: The horizontal alignment of each item when the direction is set to "vertical".

Example: `myGrid3D.horizontalAlign = "left";`

horizontalDistance

Component Inspector Name: Horizontal Distance.

Type: Number.

Default Value: 10.

Description: The horizontal distance between items.

Example: `myGrid3D.horizontalDistance = 25;`

verticalDistance

Component Inspector Name: Vertical Distance.

Type: Number.

Default Value: 10.

Description: The vertical distance between items.

Example: `myGrid3D.verticalDistance = 25;`

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: myGrid3D.handCursor = true;

smoothing

Component Inspector Name: Smoothing.

Type: Boolean.

Default Value: true.

Description: Indicates whether the images will be smoothed.

Example: myGrid3D.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: myGrid3D.interactiveltems = true;

border

Component Inspector Name: Border.

Type: Boolean.

Default Value: false.

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

Example: myGrid3D.border = true;

borderColor

Component Inspector Name: Border Color.

Type: uint.

Default Value: 0xFFFFFFFF.

Description: The color of the border.

Example: myGrid3D.borderColor = 0xFF00FF;

borderAlpha

Component Inspector Name: Border Alpha.

Type: Number.

Default Value: 1.0.

Description: The alpha of the border.

Example: myGrid3D.borderAlpha = 0.5;

borderThickness

Component Inspector Name: Border Thickness.

Type: Number.

Default Value: 5.

Description: The thickness of the border.

Example: myGrid3D.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: myGrid3D.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: myGrid3D.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: myGrid3D.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: myGrid3D.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: myGrid3D.preloaderFillColor = 0x0000FF;

preloaderLineColor

Component Inspector Name: Preloader Line Color.

Type: uint.

Default Value: 0xFFFFFFFF.

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

Example: myGrid3D.preloaderLineColor = 0xFF0000;

zoomItemPreloaderType

Component Inspector Name: Zoom Item Preloader Type.

Type: String.

Default Value: "circular1".

Available Values: "circular1", "circular2", "circular3" and "none"

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

Example: myGrid3D.zoomItemPreloaderType = "circular2";

zoomItemPreloaderSize

Component Inspector Name: Zoom Item Preloader Size.

Type: Number.

Default Value: 20.

Description: The radius of the preloader.

Example: myGrid3D.zoomItemPreloaderSize =15;

zoomItemPreloaderFillColor

Component Inspector Name: Zoom Item Preloader Fill Color.

Type: uint.

Default Value: 0xFFFFFFFF.

Description: The color of the preloader.

Example: myGrid3D. zoomItemPreloaderFillColor = 0x0000FF;

zoomItemFadeDuration

Component Inspector Name: Zoom Item Fade Duration.

Type: Number.

Default Value: 0.5.

Description: The duration, in seconds, for the zoomed item to fade in.

Example: myGrid3D.zoomItemFadeDuration = 1;

reflection

Component Inspector Name: Reflection.

Type: Boolean.

Default Value: false.

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

Example: `myGrid3D.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 item.

Example: `myGrid3D.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: `myGrid3D.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 containing the ratio values of the reflection.

Example: `myGrid3D.reflectionRatio = [0, 255];`

quality

Component Inspector Name: Quality.

Type: uint.

Default Value: 2.

Description: A number representing the quality of the perspective view for the items. Setting this property to higher values will decrease the performance.

Example: myGrid3D.quality = 1;

cameraZoom

Component Inspector Name: Camera Zoom.

Type: Number.

Default Value: 70.

Description: A number representing the zoom value of the 3D Camera.

Example: myGrid3D.cameraZoom = 40;

cameraFocus

Component Inspector Name: Camera Focus.

Type: Number.

Default Value: 10.

Description: A number representing the focus value of the 3D Camera.

Example: myGrid3D.cameraFocus = 15;

cameraDepth

Component Inspector Name: Camera Depth.

Type: Number.

Default Value: -700.

Description: A number representing the position, on the Z axis, of the 3D Camera.

Example: myGrid3D.cameraDepth = -800;

cameraAcceleration

Component Inspector Name: Camera Acceleration.

Type: Number.

Default Value: 2.

Description: A number representing the acceleration of the 3D Camera when the grid is moving.

Example: `myGrid3D.cameraAcceleration = 3;`

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: `myGrid3D.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: `myGrid3D.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: myGrid3D.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: myGrid3D.skipBrokenPath = true;

selectedItemDepth

Component Inspector Name: Selected Item Depth.

Type: Number.

Default Value: 100.

Description: Represents the position, on the Z axis, to which the item will move when it is selected.

Example: myGrid3D.selectedItemDepth = 200;

selectedItemMoveDuration

Component Inspector Name: Selected Item Move Duration.

Type: Number.

Default Value: 1.

Description: Represents the duration of the movement when an item is selected.

Example: `myGrid3D.selectedItemMoveDuration = 0.4;`

selectedItemMoveEasing

Component Inspector Name: Selected Item Move Easing.

Type: String.

Default Value: "exponentialOut".

Description: Represents the easing of the movement when an item is selected.

Example: `myGrid3D.selectedItemMoveEasing = "linear";`

selectEasingStrength

Component Inspector Name: Select Easing Strength.

Type: Number.

Default Value: 10.

Description: Represents the strength of the easing when an item is selected.

Example: `myGrid3D.selectEasingStrength = 20;`

dragScroll

Component Inspector Name: Drag Scroll.

Type: Boolean.

Default Value: false.

Description: Indicates whether scrolling the component's items by mouse dragging is activated.

Example: `myGrid3D.dragScroll = true;`

dragScrollSpeed

Component Inspector Name: Drag Scroll Speed.

Type: Number.

Default Value: 30.

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

Example: myGrid3D.dragScrollSpeed = 30;

dragScrollEasingStrength

Component Inspector Name: Drag Scroll Easing Strength.

Type: Number.

Default Value: 10.

Description: The strength of the easing when dragScroll is set to true. A lower value indicates a stronger easing.

Example: myGrid3D.dragScrollEasingStrength = 30;

dragScrollReversed

Component Inspector Name: Drag Scroll Reversed.

Type: Boolean.

Default Value: false.

Description: Indicates whether the items are scrolling in reversed direction relative to the mouse movement.

Example: myGrid3D.dragScrollReversed = true;

scrollAmount

Component Inspector Name: Scroll Amount.

Type: Number.

Default Value: 200.

Description: The amount to scroll when using `scrollLeft()`, `scrollRight()`, `scrollUp()` and `scrollDown()`.

Example: `myGrid3D.scrollAmount = 100;`

scrollDuration

Component Inspector Name: Scroll Duration.

Type: Number.

Default Value: 1.

Description: The scroll duration in seconds.

Example: `myGrid3D.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", "cubicInOut", "elasticIn", "elasticOut", "elasticInOut", "exponentialIn", "exponentialOut", "exponentialInOut", "quadraticIn", "quadraticOut", "quadraticInOut", "quarticIn", "quarticOut", "quarticInOut", "quinticIn", "quinticOut", "quinticInOut", "sineIn", "sineOut", "sineInOut"

Description: The scroll easing.

Example: `myGrid3D.scrollEasing = "backOut";`

zoomDuration

Component Inspector Name: Zoom Duration.

Type: Number.

Default Value: 1.

Description: The zoom duration in seconds.

Example: myGrid3D.zoomDuration = 3;

zoomEasing

Component Inspector Name: Zoom Easing.

Type: String.

Default Value: "exponentialOut".

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

Description: The zoom easing.

Example: myGrid3D.zoomEasing = "backOut";

autoScroll

Component Inspector Name: Auto Scroll.

Type: Boolean.

Default Value: false.

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

Example: myGrid3D.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: myGrid3D.autoScrollDelay = 4;

autoScrollDirection

Component Inspector Name: Auto Scroll Direction.

Type: String.

Default Value: "next".

Available Values: "next" and "previous".

Description: Indicates whether the next item or the previous item will be selected.

Example: myGrid3D.autoScrollDirection = "previous";

zoomAutoScroll

Component Inspector Name: Zoom Auto Scroll.

Type: Boolean.

Default Value: false.

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

Example: myGrid3D.zoomAutoScroll = true;

zoomAutoScrollDelay

Component Inspector Name: Zoom Auto Scroll Delay.

Type: Number.

Default Value: 3.

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

Example: myGrid3D.zoomAutoScrollDelay = 4;

zoomAutoScrollDirection

Component Inspector Name: Zoom Auto Scroll Direction.

Type: String.

Default Value: "next".

Available Values: "next" and "previous".

Description: Indicates whether the next item or the previous item will be zoomed.

Example: myGrid3D.zoomAutoScrollDirection = "previous";

zoomItemScale

Component Inspector Name: Zoom Item Scale.

Type: Number.

Default Value: 3.

Description: Indicates how much the items will be scaled when zooming is used.

Example: myGrid3D.zoomItemScale = 2;

zoomItemScaleMode

Component Inspector Name: Zoom Item Scale Mode.

Type: String.

Default Value: "exactFit".

Available Values: "exactFit" and "maintainAspectRatio";

Description: The scaling of the zoomed item. “exactFit” will resize the item to the maximum width and height. “maintainAspectRatio” will resize the item to the maximum width or height, maintaining the aspect ratio.

Example: `myGrid3D.zoomItemScaleMode = “maintainAspectRatio”;`

useSmoothing

Component Inspector Name: Use Smoothing.

Type: Boolean.

Default Value: false.

Description: Indicates whether the images will be smoothed during the scrolling process. Leave this property to false for a smoother scrolling. When scrolling is complete, the images will automatically be smoothed.

Example: `myGrid3D.useSmoothing = true;`

usePrecision

Component Inspector Name: Use Precision.

Type: Boolean.

Default Value: true.

Description: Indicates whether the 3D perspective of the images will be precise during the scrolling process. Set this property to false for a smoother scrolling. If set to false, the images will look a little distorted. When the scrolling is complete, the 3D perspective will automatically become precise.

Example: `myGrid3D.usePrecision = false;`

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: `myGrid3D.selectOver = 3;`

zoomOver

Component Inspector Name: Zoom Over.

Type: uint.

Default Value: 1.

Description: Indicates over how many items the zooming will be executed when `zoomNext()` and `zoomPrevious()` methods are used.

Example: `myGrid3D.zoomOver = 3;`

effect

Component Inspector Name: Effect.

Type: Object.

Default Value: null.

Description: An instance of an Effect component.

Example:

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

```
myGrid3D.effect = greyEffect; or myGrid3D.effect = "greyEffect";
```

videoPlayer

Component Inspector Name: Video Player.

Type: Object.

Default Value: null.

Description: An instance of a Grid3DVideoPlayer component.

Example:

```
var vp:Grid3DVideoPlayer = new Grid3DVideoPlayer();  
myGrid3D.videoPlayer = vp;
```

livePreviewItems

Component Inspector Name: Live Preview Items.

Type: uint.

Default Value: 10.

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

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:

```
myGrid3D.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:

```
myGrid3D.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"}];  
  
myGrid3D.addItems(myItems);
```

addItemsAt()

Implementation: addItems(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"}];  
  
myGrid3D.addItems(myItems, 5);
```

removeItem()

Implementation: removeItem(item:IDisplayItem):void

Description: Removes the specified item.

Parameters: item - The item to be removed.

Example:

```
myGrid3D.removeItem(myGrid3D.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:

```
myGrid3D.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:

```
myGrid3D.removeItemsAt(0, 3);
```

removeAllItems()

Implementation: removeAllItems():void

Description: Removes all items.

Example:

```
myGrid3D.removeAllItems();
```

replaceItem()

Implementation: replaceItem(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"};
myGrid3D.replaceItem(data, myGrid3D.getItemAt(5));
```

replaceItemAt()

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

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

Parameters: data - The data of the new item.

index -The index of the item to be replaced.

Example:

```
var data:Object = {source:"images/image1.jpg", desc:"Bird"};
myGrid3D.replaceItem(data, 2);
```

spliceItems()

Implementation:

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

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

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

removeCount - The number of elements to be removed.

items - The items to be added.

Example:

```
var newItems:Array = [{source:"images/image0.jpg"},
                      {source:"images/image1.jpg",title:"Bird"},
                      {source:"images/image2.jpg",title:"Sky"}];
```

```
{source:"images/image3.jpg"}];
```

```
myGrid3D.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(myGrid3D.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 = myGrid3D.getItemIndex(myGrid3D.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:

```
myGrid3D.select(5);
```

deselect()

Implementation: deselect(index:uint):void

Description: Deselects the item at the specified index.

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

Example:

```
myGrid3D.deselect(myGrid3D.selectedIndex);
```

selectNext()

Implementation: selectNext():void

Description: Selects the next item.

Example:

```
myGrid3D.selectNext();
```

selectPrevious()

Implementation: selectPrevious():void

Description: Selects the previous item.

Example:

```
myGrid3D.selectPrevious();
```

scrollRight()

Implementation: scrollRight():void

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

Example:

```
myGrid3D.scrollRight();
```

scrollLeft()

Implementation: scrollLeft():void

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

Example:

```
myGrid3D.scrollLeft();
```

scrollUp()

Implementation: scrollUp():void

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

Example:

```
myGrid3D.scrollUp();
```

scrollDown()

Implementation: scrollDown():void

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

Example:

```
myGrid3D.scrollDown();
```

zoom()

Implementation: zoom(index:uint):void

Description: Zooms the item at the specified index.

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

Example:

```
myGrid3D.zoom(5);
```

zoomOut()

Implementation: zoomOut():void

Description: Zooms out the grid.

Example:

```
myGrid3D.zoomOut();
```

zoomNext()

Implementation: zoomNext():void

Description: Zooms the next item.

Example:

```
myGrid3D.zoomNext();
```

zoomPrevious()

Implementation: zoomPrevious():void

Description: Zooms the previous item.

Example:

```
myGrid3D.zoomPrevious();
```

setSize()

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

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

Example:

```
myGrid3D.setSize(700, 500);
```

move()

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

Description: Moves the component at the specified position.

Example:

```
myGrid3D.move(100, 100);
```

11. Events

ITEMS_LOAD_START

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

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEMS_LOAD_START, eventHandler)
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEMS_LOAD_PROGRESS, eventHandler)
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
```

```
myGrid3D.addEventListener(Grid3DEvent.ITEMS_LOAD_COMPLETE, eventHandler
)
function eventHandler(event:Grid3DEvent):void
{
    trace("loading complete");
}
```

XML_LOAD_START

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

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.XML_LOAD_START, eventHandler);
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.XML_LOAD_PROGRESS,
eventHandler);
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.XML_LOAD_COMPLETE, eventHandler)
function eventHandler(event:Grid3DEvent):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.WallEvent;
myWall.addEventListener(WallEvent.XML_LOAD_ERROR, eventHandler)
function eventHandler(event:WallEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEM_ADD, eventHandler);
function eventHandler(event:Grid3DEvent):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"
```

```
}  
myGrid3D.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.flashotaku.events.Grid3DEvent;  
myGrid3D.addEventListener(Grid3DEvent.ITEM_REMOVE, eventHandler);  
function eventHandler(event:Grid3DEvent):void  
{  
    trace(event.index); // output 1  
}  
myGrid3D.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.flashotaku.events.Grid3DEvent;  
myGrid3D.addEventListener(Grid3DEvent.ITEM_CLICK, eventHandler);  
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;

myGrid3D.addEventListener(Grid3DEvent.ITEM_DOUBLE_CLICK,
    eventHandler);
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEM_MOUSE_UP, eventHandler);
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEM_MOUSE_DOWN, eventHandler);

function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEM_MOUSE_OVER, eventHandler);
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEM_MOUSE_OUT, eventHandler);
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEM_START, eventHandler);
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
```

```
myGrid3D.addEventListener(Grid3DEvent.ITEM_PROGRESS, eventHandler);
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEM_COMPLETE, eventHandler);
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEM_ERROR, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("Item at index: " + event.index + " could not be loaded");
}
```

ZOOM_ITEM_CLICK

Description: Dispatched when a zoomed 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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_ITEM_CLICK, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace(event.index);
}
```

ZOOM_ITEM_DOUBLE_CLICK

Description: Dispatched when a zoomed 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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_ITEM_DOUBLE_CLICK,
eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace(event.data);
}
```

ZOOM_ITEM_MOUSE_UP

Description: Dispatched when the mouse is released over a zoomed 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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_ITEM_MOUSE_UP,
eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace(event.index);
}
```

ZOOM_ITEM_MOUSE_DOWN

Description: Dispatched when the mouse is pressed over a zoomed 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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_ITEM_MOUSE_DOWN,
eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace(event.index);
}
```

ZOOM_ITEM_MOUSE_OVER

Description: Dispatched when the mouse pointer is moved over a zoomed 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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_ITEM_MOUSE_OVER,
eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace(event.index);
}
```

ZOOM_ITEM_MOUSE_OUT

Description: Dispatched when the mouse pointer is moved away from a zoomed 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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_ITEM_MOUSE_OUT,
eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace(event.index);
}
```

ZOOM_ITEM_START

Description: Dispatched when the content of a zoomed 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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_ITEM_START, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("Item at index: " + event.index + " has started loading");
}
```

ZOOM_ITEM_PROGRESS

Description: Dispatched during the loading of a zoomed 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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_ITEM_PROGRESS,
eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("Loaded " + event.bytesLoaded + " out of "
+event.bytesTotal);
}
```

ZOOM_ITEM_COMPLETE

Description: Dispatched when the content of a zoomed 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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_ITEM_COMPLETE,
eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("Item at index: " + event.index + " has finished loading");
}
```

ZOOM_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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_ITEM_ERROR, eventHandler);
function eventHandler(event:Grid3DEvent):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.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEM_SELECTED, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("Index of the selected item: " + event.index);
}
```

ITEM_DESELECTED

Description: Dispatched when an item was deselected.

Special Properties: item – The item which was deselected.

index – The index of the item.

data – The data of the item.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ITEM_DESELECTED, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("Index of the deselected item: " + event.index);
}
```

RESIZE

Description: Dispatched when the size of the component changes.

Example:

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

UPDATE

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

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.UPDATE, eventHandler);

function eventHandler(event:Grid3DEvent):void
{
    trace("Something changed!");
}
```

SCROLL_START

Description: Dispatched when the scrolling starts.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.SCROLL_START, eventHandler);

function eventHandler(event:Grid3DEvent):void
{
    trace("scroll started");
}
```

SCROLL_PROGRESS

Description: Dispatched during the scrolling process.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.SCROLL_PROGRESS, eventHandler);

function eventHandler(event:Grid3DEvent):void
{
    trace("is scrolling");
}
```

SCROLL_COMPLETE

Description: Dispatched when the scrolling is complete.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.SCROLL_COMPLETE, eventHandler);

function eventHandler(event:Grid3DEvent):void

{
    trace("scroll complete");
}
```

ZOOM_START

Description: Dispatched when the zooming starts.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_START, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("zoom started");
}
```

ZOOM_PROGRESS

Description: Dispatched during the zooming process.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_PROGRESS, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("is zooming");
}
```

ZOOM_COMPLETE

Description: Dispatched when the zooming is complete.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_COMPLETE, eventHandler);

function eventHandler(event:Grid3DEvent):void
{
    trace("zoom complete");
}
```

ZOOM_OUT_START

Description: Dispatched when the zoom out starts.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_OUT_START, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("zoom out started");
}
```

ZOOM_OUT_PROGRESS

Description: Dispatched during the zoom out process.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_OUT_PROGRESS,
eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("is zooming out");
}
```

ZOOM_OUT_COMPLETE

Description: Dispatched when the zoom out is complete.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.ZOOM_OUT_COMPLETE,
    eventHandler);

function eventHandler(event:Grid3DEvent):void
{
    trace("zoom out complete");
}
```

DISPLAY_ZOOM_ITEM

Description: Dispatched when the zoomed item is displayed.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.DISPLAY_ZOOM_ITEM,
    eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace(event.index);
}
```

OUTSIDE_CLICK

Description: Dispatched when the user clicks outside the component's items.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.OUTSIDE_CLICK, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("outside click");
}
```

OUTSIDE_DOUBLE_CLICK

Description: Dispatched when the user double-clicks outside the component's items.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.OUTSIDE_DOUBLE_CLICK,
    eventHandler);

function eventHandler(event:Grid3DEvent):void
{
    trace("outside double click");
}
```

OUTSIDE_MOUSE_UP

Description: Dispatched when the mouse is released outside the component's items.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.OUTSIDE_MOUSE_UP, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("outside mouse up");
}
```

OUTSIDE_MOUSE_DOWN

Description: Dispatched when the mouse is pressed outside the component's items.

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid3D.addEventListener(Grid3DEvent.OUTSIDE_MOUSE_DOWN,
    eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("outside mouse down");
}
```

REACHED_TOP

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

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid.addEventListener(Grid3DEvent.REACHED_TOP, eventHandler);

function eventHandler(event:Grid3DEvent):void
{
    trace("reached top");
}
```

REACHED_BOTTOM

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

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid.addEventListener(Grid3DEvent.REACHED_BOTTOM, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("reached bottom");
}
```

REACHED_LEFT

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

Example:

```
import com.flashotaku.events.Grid3DEvent;
myGrid.addEventListener(Grid3DEvent.REACHED_LEFT, eventHandler);
function eventHandler(event:Grid3DEvent):void{
    trace("reached left");
}
```

REACHED_RIGHT

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

Example:

```
import com.flashotaku.events.Grid3DEvent;

myGrid.addEventListener(Grid3DEvent.REACHED_RIGHT, eventHandler);
function eventHandler(event:Grid3DEvent):void
{
    trace("reached right");
}
```