

AFC

AFCOMPONENTS

A division of Professional Content, LLC

3D Wall

Video Player 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

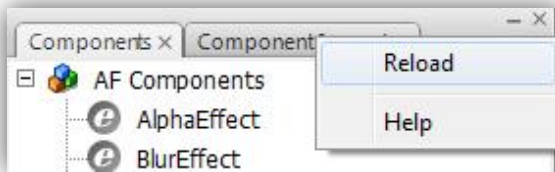
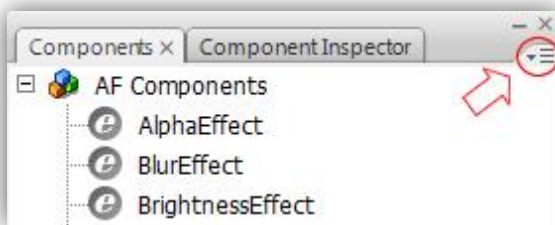
1. Component Installation	1
2. Getting Started	2
3. Setting the component's properties	4
4. Styling the component's assets	5
5. RTMP Streaming	6
6. Using the component in FLEX	7
7. Properties	8
scaleMode	8
autoPlay	8
autoReplay	8
bufferTime	9
skins	9
fadeControls	9
fadeInDuration	9
fadeOutDuration	10
fadeOutDelay	10
initialFadeOutDelay	10
playButtonAlpha	11
playButtonScale	11
smoothing	11
marginBottom	11
marginLeft	12
marginRight	12
fullScreenMarginBottom	12
fullScreenMarginLeft	12
fullScreenMarginRight	13
fullScreenButton	13
fullScreenTakeOver	13
backgroundColor	14
backgroundAlpha	14
preloaderType	14

Table of Contents

preloaderSize.....	14
preloaderColor	15
reflection	15
reflectionAlpha	15
reflectionRatio	16
volume	16
duration.....	17
time.....	17
8. Methods	18
start()	18
resume()	18
stop()	18
pause().....	18
seek().....	19
close().....	19
9. Events	20
VIDEO_CLICK	20
VIDEO_DOUBLE_CLICK.....	20
VIDEO_MOUSE_UP.....	20
VIDEO_MOUSE_DOWN.....	21
VIDEO_MOUSE_OVER.....	21
VIDEO_MOUSE_OUT	21
BEGIN	22
CLOSE.....	22
START.....	22
STOP	23
PAUSE.....	23
RESUME	23
COMPLETE	24

1. *Component Installation*

1. Double-Click on the WallVideoPlayer.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 WallVideoPlayer component onto the stage.
Give the component an instance name, “videoPlayer” for example. Then click on the Wall component and go to the Component Inspector panel. In the list of properties you will see a property ‘Video Player’. Set that property to the instance name of the WallVideoPlayer component, which is “videoPlayer”.
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. Also create a folder for the video files and name it “videos”.
4. For this example we’ll use an XML file to specify the location of the files. 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>
    <video>videos/trip.flv</video>
  </item>
  <item>
    <source>images/image2.jpg</source>
    <video>videos/concert.mp4</video>
  </item>
  <item>
    <source>images/image3.jpg</source>
    <video>videosList/cars.mov</video>
  </item>
  <item>
    <source>images/image4.jpg</source>
  </item>
  <item>
    <source>images/image5.jpg</source>
  </item>
</items>
```

The only required keywords are “source” and “video”. You need to use these keywords to specify the location of the images and videos you want to load.

You can also specify the id of a YouTube video.

```
<items>
  <item>
    <source>images/image1.jpg</source>
    <video>uFo-H5g6EJo</video>
  </item>
</items>
```

3. Setting the component's properties

The WallVideoPlayer component can be customized exactly the same way you customize the Wall component. You can set its properties using the Component Inspector or using Actionscript code.

Example:

```
import com.afcomponents.wall.WallVideoPlayer;

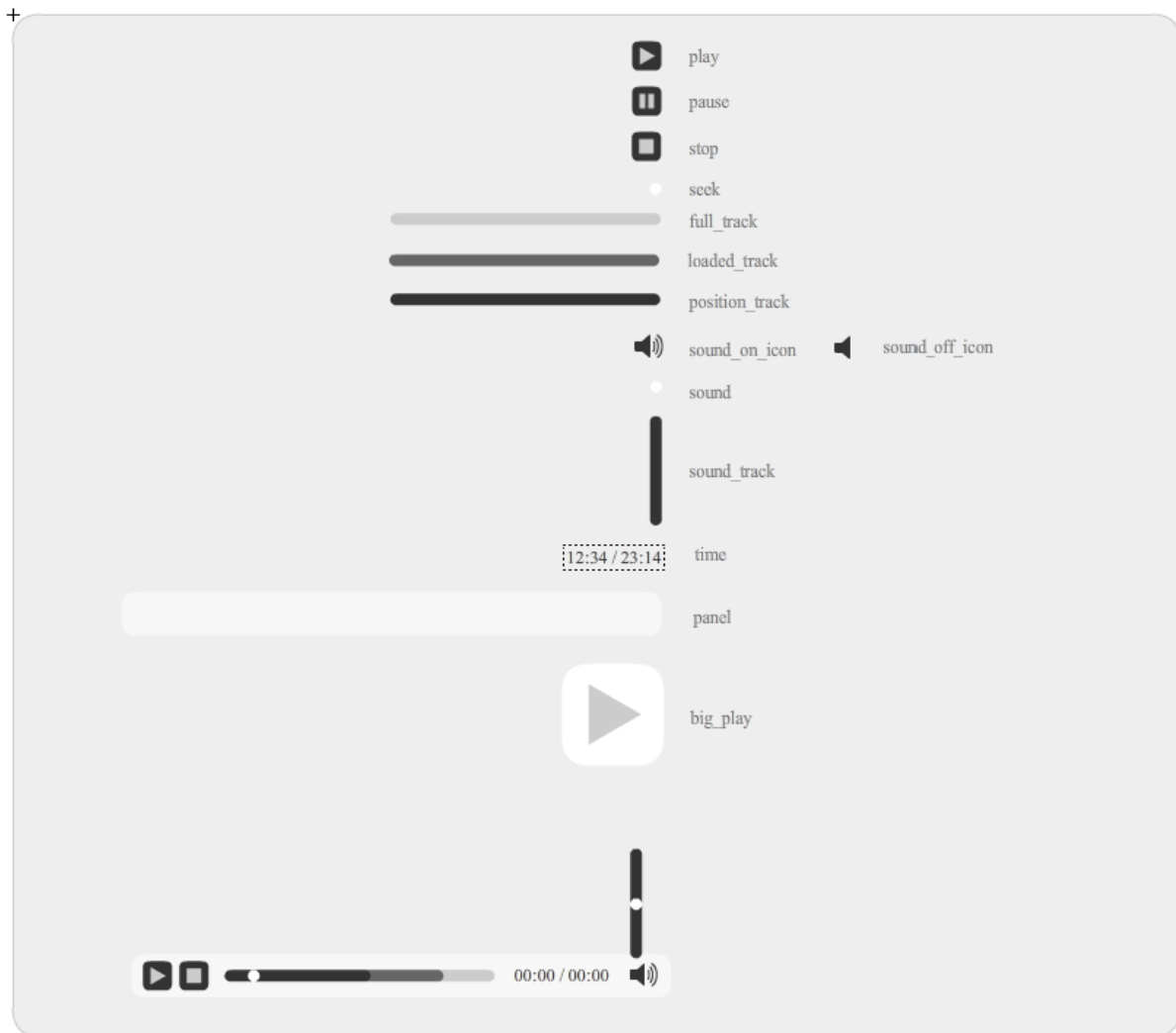
var myVideoPlayer:WallVideoPlayer = new WallVideoPlayer();
addChild(myVideoPlayer);
myVideoPlayer.border = true;
myVideoPlayer.marginBottom = 20;

myWall.videoPlayer = myVideoPlayer;
```

4. *Styling the component's assets*

You can customize the video player's assets the same way you customize the components included with Flash, like Button, Slider etc.

Drag an instance onto the stage and double-click on it to see its assets.



5. RTMP Streaming

If you want to stream video files from a RTMP server, you will have to specify the path to the directory where the application files are in the “server” attribute. In the “video” attribute only the name of the video file needs to be specified.

Example:

```
<items>
  <item>
    <source> myImage1.jpg</source>
    <video>concert.flv</video>
    <server>rtmp://localhost/myApplication</server>
  </item>
  <item>
    <source>image2.jpg</source>
    <video>trip.mp4</video>
    <server> rtmp://www.domain.com/videoApp</server>
  </item>
</items>
```

If you intend to use reflections for your videos, make sure the client application has access to raw video data.

In Flash Media Server, you can enable access to raw data by adding the following tag in the Application.xml file:

```
<VideoSampleAccess enabled="true"></VideoSampleAccess>
```

If you don't have an Application.xml file in your application's folder you will need to create one. After you create the file, add the following lines:

```
<Application>
  <Client>
    <Access>
      <VideoSampleAccess enabled="true"></VideoSampleAccess>
    </Access>
  </Client>
</Application>
```

6. *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 wallContainer:ComponentWrapper = new ComponentWrapper(myWall);

var myVideoPlayer:WallVideoPlayer = new WallVideoPlayer ();
var videoPlayerContainer:ComponentWrapper = new ComponentWrapper(myVideoPlayer);

var panel:Panel = new Panel();
panel.setStyle("verticalGap", 0);
addChild(panel);

panel.addChild(wallContainer);
panel.addChild(videoPlayerContainer);
```

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.videoPlayer = myVideoPlayer;
myVideoPlayer.autoPlay = true;
```

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

7. *Properties*

scaleMode

Component Inspector Name: Scale Mode.

Type: String.

Default Value: "exactFit".

Available Values: "exactFit", "maintainAspectRatio" and "noScale";

Description: The scaling of the video. "exactFit" will resize the video to the maximum width and height. "maintainAspectRatio" will resize the video to the maximum width or height, maintaining the aspect ratio. "noScale" will ignore the maximum width and height, maintaining the video at the original dimensions.

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

autoPlay

Component Inspector Name: Auto Play.

Type: Boolean.

Default Value: false.

Description: Indicates whether the video will play immediately after the transition of the slide is complete.

Example: `myWallVideoPlayer.autoPlay = true;`

autoReplay

Component Inspector Name: Auto Replay.

Type: Boolean.

Default Value: false.

Description: Indicates whether the video will be replayed after reaching the end.

Example: `myWallVideoPlayer.autoReplay = true;`

bufferTime

Component Inspector Name: Buffer Time.

Type: Number.

Default Value: 1.

Description: Indicates the number of seconds assigned to the buffer.

Example: `myWallVideoPlayer.bufferTime = 5;`

skins

Component Inspector Name: Skins.

Type: String.

Default Value: "".

Description: The path to the SWF file that contains the skin assets. You must use this property when you use the component in FLEX.

Example: `myWallVideoPlayer.skins = "assets/skins/mySkin.swf";`

fadeControls

Component Inspector Name: Fade Controls.

Type: Boolean.

Default Value: true.

Description: Indicates whether the player controls will fade out.

Example: `myWallVideoPlayer.fadeControls = false;`

fadeInDuration

Component Inspector Name: Fade In Duration.

Type: Number.

Default Value: 0.5.

Description: The duration of the fade in.

Example: `myWallVideoPlayer.fadeInDuration = 2;`

fadeOutDuration

Component Inspector Name: Fade Out Duration.

Type: Number.

Default Value: 1.

Description: The duration of the fade out.

Example: `myWallVideoPlayer.fadeOutDuration = 2;`

fadeOutDelay

Component Inspector Name: Fade Out Delay.

Type: Number.

Default Value: 1.

Description: The delay of the fade out.

Example: `myWallVideoPlayer.fadeOutDelay = 2;`

initialFadeOutDelay

Component Inspector Name: Initial Fade Out Delay.

Type: Number.

Default Value: 3.

Description: The delay of the initial fade out.

Example: `myWallVideoPlayer.initialFadeOutDelay = 2;`

playButtonAlpha

Component Inspector Name: Play Button Alpha.

Type: Number.

Default Value: 0.8.

Description: The alpha of the initial play button.

Example: `myWallVideoPlayer.playButtonAlpha = 1.0;`

playButtonScale

Component Inspector Name: Play Button Scale.

Type: Number.

Default Value: 2.

Description: Indicates how much the play button will be scaled after it is pressed.

Example: `myWallVideoPlayer.playButtonScale = 1.5;`

smoothing

Component Inspector Name: Smoothing.

Type: Boolean.

Default Value: false.

Description: Indicates whether the videos will be smoothed.

Example: `myWallVideoPlayer.smoothing = true;`

marginBottom

Component Inspector Name: Margin Bottom.

Type: Number.

Default Value: 10.

Description: The bottom margin of the player controls.

Example: `myWallVideoPlayer.marginBottom = 20;`

marginLeft

Component Inspector Name: Margin Left.

Type: Number.

Default Value: 10.

Description: The left margin of the player's controls.

Example: `myWallVideoPlayer.marginLeft = 20;`

marginRight

Component Inspector Name: Margin Right.

Type: Number.

Default Value: 10.

Description: The right margin of the player's controls.

Example: `myWallVideoPlayer.marginRight = 20;`

fullScreenMarginBottom

Component Inspector Name: Margin Bottom.

Type: Number.

Default Value: 10.

Description: The bottom margin of the player controls in full screen mode.

Example: `myWallVideoPlayer.fullScreenMarginBottom = 20;`

fullScreenMarginLeft

Component Inspector Name: Margin Left.

Type: Number.

Default Value: 10.

Description: The left margin of the player's controls in full screen mode.

Example: `myWallVideoPlayer.fullScreenMarginLeft = 20;`

fullScreenMarginRight

Component Inspector Name: Margin Right.

Type: Number.

Default Value: 10.

Description: The right margin of the player's controls in full screen mode.

Example: `myWallVideoPlayer.fullScreenMarginRight = 20;`

fullScreenButton

Component Inspector Name: Full Screen Button.

Type: Boolean.

Default Value: true.

Description: Indicates whether the "full screen" button will be displayed.

Example: `myWallVideoPlayer.fullScreenButton = false;`

fullScreenTakeOver

Component Inspector Name: Full Screen Take Over.

Type: Boolean.

Default Value: true.

Description: Indicates whether in full screen mode the video will be on top of all content from the stage.

Example: `myWallVideoPlayer.fullScreenTakeOver = false;`

backgroundColor

Component Inspector Name: Background Color.

Type: uint.

Default Value: 0x000000.

Description: The color of the video background in full screen mode.

Example: myWallVideoPlayer.backgroundColor = 0x0000FF;

backgroundAlpha

Component Inspector Name: Background Alpha.

Type: Number.

Default Value: 1.

Description: The alpha of the video background in full screen mode.

Example: myWallVideoPlayer.backgroundAlpha = 0.5;

preloaderType

Component Inspector Name: 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: myWallVideoPlayer.preloaderType = "circular2";

preloaderSize

Component Inspector Name: Preloader Size.

Type: Number.

Default Value: 20.

Description: The radius of the preloader.

Example: `myWallVideoPlayer.preloaderSize = 15;`

preloaderColor

Component Inspector Name: Preloader Color.

Type: uint.

Default Value: 0xFFFFFFFF.

Description: The color of the preloader.

Example: `myWallVideoPlayer.preloaderColor = 0x0000FF;`

reflection

Component Inspector Name: Reflection.

Type: Boolean.

Default Value: false.

Description: Indicates whether video reflection will be displayed.

Example: `myWallVideoPlayer.reflection = true;`

reflectionAlpha

Component Inspector Name: Reflection Alpha.

Type: Array.

Default Value: [0.7, 0].

Description: An array indicating the transparency of the reflection at its top and at its bottom.

Example: `myWallVideoPlayer.reflectionAlpha = [1, 0];`

reflectionRatio

Component Inspector Name: Reflection Ratio.

Type: Array.

Default Value: [0, 127].

Description: An array indicating the ratio of the reflection.

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

volume

Component Inspector Name: NA.

Type: Number.

Default Value: 0.5.

Description: A number between 0 and 1 representing the volume of the video.

Example: `myWallVideoPlayer.volume = 0.4;`

Read-only properties

duration

Type: Number.

Description: Returns the duration of the video.

Example:

```
trace(myWallVideoPlayer.duration);
```

time

Type: Number.

Description: Returns the current time of the video.

Example:

```
trace(myWallVideoPlayer.time);
```

8. *Methods*

start()

Implementation: start():void

Description: Starts the video playback. Calling this method is the equivalent of clicking the big play button.

Example:

```
myWallVideoPlayer.start();
```

resume()

Implementation: resume():void

Description: Resumes the currently playing video.

Example:

```
myWallVideoPlayer.resume();
```

stop()

Implementation: stop():void

Description: Stops the currently playing video.

Example:

```
myWallVideoPlayer.stop();
```

pause()

Implementation: pause():void

Description: Pauses the currently playing video.

Example:

```
myWallVideoPlayer.pause();
```

seek()

Implementation: seek(value:Number):void

Description: Seeks the currently playing video to the specified time value.

Example:

```
myWallVideoPlayer.seek(100);
```

close()

Implementation: close():void

Description: Closes the currently playing video.

Example:

```
myWallVideoPlayer.close();
```

9. Events

VIDEO_CLICK

Description: Dispatched when a video was clicked.

Example:

```
import com.afcomponents.events.VideoPlayerEvent;
myWallVideoPlayer.addEventListener(VideoPlayerEvent.VIDEO_CLICK, eventHandler);

function eventHandler(event:VideoPlayerEvent):void{
    trace(event.type);
}
```

VIDEO_DOUBLE_CLICK

Description: Dispatched when a video was double-clicked.

Example:

```
import com.afcomponents.events. VideoPlayerEvent;
myWall.addEventListener(VideoPlayerEvent.VIDEO_DOUBLE_CLICK, eventHandler);

function eventHandler(event: VideoPlayerEvent):void
{
    trace(event.type);
}
```

VIDEO_MOUSE_UP

Description: Dispatched when the mouse is released over a video.

Example:

```
import com.afcomponents.events.IVideoPlayerEvent;
myWallVideoPlayer.addEventListener(VideoPlayerEvent.VIDEO_MOUSE_UP, eventHandler);
```

```
function eventHandler(event: VideoPlayerEvent):void
{
    trace(event.type);
}
```

VIDEO_MOUSE_DOWN

Description: Dispatched when the mouse is pressed over a video.

Example:

```
import com.afcomponents.events. VideoPlayerEvent;
myWallVideoPlayer.addEventListener(VideoPlayerEvent.VIDEO_MOUSE_DOWN, eventHandler);

function eventHandler(event: VideoPlayerEvent):void
{
    trace(event.type);
}
```

VIDEO_MOUSE_OVER

Description: Dispatched when the mouse pointer is moved over a video.

Example:

```
import com.afcomponents.events. VideoPlayerEvent;
myVideoPlayerWall.addEventListener(VideoPlayerEvent.VIDEO_MOUSE_OVER, eventHandler);

function eventHandler(event: VideoPlayerEvent):void
{
    trace(event.type);
}
```

VIDEO_MOUSE_OUT

Description: Dispatched when the mouse pointer is moved away from a video.

Example:

```
import com.afcomponents.events. VideoPlayerEvent;

myWallVideoPlayer.addEventListener(VideoPlayerEvent.VIDEO_MOUSE_OUT, eventHandler);
```

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

BEGIN

Description: Dispatched when the initial play button was clicked.

Example:

```
import com.afcomponents.events. VideoPlayerEvent;
myWallVideoPlayer.addEventListener(VideoPlayerEvent.BEGIN, eventHandler);

function eventHandler(event: VideoPlayerEvent):void
{
    trace("begin");
}
```

CLOSE

Description: Dispatched when the video player is closed, after the user moved to a new slide or after the close() method was called.

Example:

```
import com.afcomponents.events. VideoPlayerEvent;
myWallVideoPlayer.addEventListener(VideoPlayerEvent.CLOSE, eventHandler);

function eventHandler(event: VideoPlayerEvent):void
{
    trace("close");
}
```

START

Description: Dispatched when the playback initially starts to play.

Example:

```
import com.afcomponents.events. VideoPlayerEvent;
myWallVideoPlayer.addEventListener(VideoPlayerEvent.START, eventHandler);
```

```
function eventHandler(event: VideoPlayerEvent):void
{
    trace("start");
}
```

STOP

Description: Dispatched when the stop button is pressed or when the playback is complete.

Example:

```
import com.afcomponents.events. VideoPlayerEvent;
myWallVideoPlayer.addEventListener(VideoPlayerEvent.STOP, eventHandler);

function eventHandler(event: VideoPlayerEvent):void
{
    trace("stop");
}
```

PAUSE

Description: Dispatched when the pause button is.

Example:

```
import com.afcomponents.events. VideoPlayerEvent;
myWallVideoPlayer.addEventListener(VideoPlayerEvent.PAUSE, eventHandler);

function eventHandler(event: VideoPlayerEvent):void
{
    trace("pause");
}
```

RESUME

Description: Dispatched when the playback is resumed.

Example:

```
import com.afcomponents.events. VideoPlayerEvent;

myWallVideoPlayer.addEventListener(VideoPlayerEvent.RESUME, eventHandler);
```

```
function eventHandler(event: VideoPlayerEvent):void
{
    trace("resume");
}
```

COMPLETE

Description: Dispatched when the playback is complete.

Example:

```
import com.afcomponents.events. VideoPlayerEvent;
myWallVideoPlayer.addEventListener(VideoPlayerEvent.COMPLETE, eventHandler);

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