JW Player for Flash

Release 5.2

www.longtailvideo.com

June 09, 2010

CONTENTS

CHAPTER

ONE

INTRODUCTION

The JW Player for Flash is the Internet's most popular and flexible media player. It supports playback of *any media type* the Adobe Flash Player can handle, both by using simple downloads, *HTTP Pseudostreaming* and *RTMP Streaming*.

The player supports various *playlist formats* and a wide range of *options* (flashvars) for changing its layout and behavior. Embedding the player in a webpage *is a breeze*.

1.1 API

For JavaScript developers, the player features an extensive *JavaScript API*. With this API, it is possible to both control the player (e.g. pause it) and respond to playback changes (e.g. when the video has ended).

1.2 Addons

Both the layout and the behavior of the player can be extended with a range of so-called AddOns. These AddOns are available on the LongTail Video website. There are three categories: skins, plugins and providers

1.2.1 Skins

Skins drastically change the looks of the player. They solely consist of an XML file and a bunch of PNG images, which makes *creating your own skins <skinning>* simple and fun.

A wide range of professional-looking skins can also be downloaded.

1.2.2 Plugins and providers

Plugins extend the functionality of the player, e.g. in the areas of analytics, advertising or viral sharing. Plugins are loaded from our plugin repository, making them extremely easy to install.

Providers are similar to plugins. They are externally loaded SWF files that can be installed with a single *option*. Whereas plugins are used to add functionality on top of the player, providers are used to extend the low-level playback functionality of the player, e.g. to support advanced features of a specific CDN or video portal. Providers are new to the 5.x player; a couple of are already available from our addons repository.

It is possible to create your own plugins and providers using Adobe Flash and actionscript, but this is not covered by these publisher-focused documents. Instead, visit developer.longtailvideo.com to learn more and download the plugin and/or provider SDK.

CHAPTER

TWO

EMBEDDING THE PLAYER

Like every other Flash object, the JW Player has to be embedded into the HTML of a webpage using specific embed codes. Overall, there are two methods for embedding Flash:

- Using a JavaScript (like SWFObject).
- Using a HTML tag (like *<embed>*).

We highly recommend the JavaScript method for Flash embedding. It can sniff if a browsers supports Flash, it ensures the player *JavaScript API* works and it avoids browser compatibility issues. Detailed instructions can be found below.

2.1 Upload

First, a primer on uploading. This may sound obvious, but for the JW Player to work on your website, you must upload the *player.swf* file from the download (or SVN checkout) to your webserver. If you want to play Youtube videos, you must also upload the **yt.swf** file - this is the bridge between the player and Youtube. No other files are needed.

Your *media files* and *playlists* can be hosted at any domain. Do note that *Crossdomain Security Restrictions* apply when loading these files from a different domain. In short, playing media files works, but loading playlists across domains will not work by default. Resolve this issue by hosting a *crossdomain.xml file*.

2.2 SWFObject

The preferred way to embed the JW Player on a webpage is JavaScript. There's a wide array of good, open source libraries available for doing so. We recommend **SWFObject**, the most widely used one. It has excellent documentation.

Before embedding any players on the page, make sure to include the *swfobject.js* script in the *<head>* of your HTML. You can download the script and host it yourself, or leverage the copy provided by Google:

```
<script type="text/javascript"
  src="http://ajax.googleapis.com/ajax/libs/swfobject/2.2/swfobject.js">
</script>
```

With the library set up, you can start embedding players. Here's an example:

```
<p id="container1">Please install the Flash Plugin</p>

<script type="text/javascript">
  var flashvars = { file:'/data/bbb.mp4',autostart:'true' };
```

```
  var params = { allowfullscreen:'true', allowscriptaccess:'always' };
  var attributes = { id:'player1', name:'player1' };

  swfobject.embedSWF('player.swf','container1','480','270','9.0.115','false',
    flashvars, params, attributes);
</script>
```

It's a fairly sizeable chunk of code that contains the embed *container*, *flashvars*, *params*, *attributes* and *instantiation*. Let's walk through them one by one:

- The *container* is the HTML element where the player will be placed into. It should be a block-level element, like a <p> or . If a user has a sufficient version of Flash, the text inside the container is removed and replaced by the videoplayer. Otherwise, the contents of the container will remain visible.
- The *flashvars* object lists your player *Configuration Options*. One option that should always be there is *file*, which points to the file to play. You can insert as many options as you want.
- The *params* object includes the Flash plugin parameters. The two parameters in the example (our recommendation) enable both the *fullscreen* and *JavaScript* functionality of Flash.
- The *attributes* object include the HTML attributes of the player. We recommend always (and only) setting an *id* and *name*, to the same value. This will be the *id* of the player instance if you use its *JavaScript API*.
- The *instantiation* is where all things come together and the actual player embedding takes place. These are all parameters of the SWFObject call:
  - The URL of the *player.swf*, relative to the page URL.
  - The ID of the container you want to embed the player into.
  - The width of the player, in pixels. Note the JW Player automatically stretches itself to fit.
  - The height of the player, in pixels. Note the JW Player automatically stretches itself to fit.
  - The required version of Flash. We highly recommend setting *9.0.115*. This is the first version that supports *MP4* and is currently installed at >95% of all computers. The only feature for which you might restricted to *10.0.0* is *RTMP dynamic streaming*.
  - The location of a Flash auto-upgrade script. We recommend to **not** use it. People that do not have Flash 9.0.115 either do not want or are not able (no admin rights) to upgrade.
  - Next, the *flashvars*, *params* and *attributes* are passed, in that order.

It is no problem to embed multiple players on a page. However, make sure to give each player instance a different container **id** and a different attributess **id** and **name**.

2.3 Embed tag

In cases where a JavaScript embed method is not possible (e.g. if your CMS does not allow including JavaScripts), the player can be embedded using plain HTML. There are various combinations of tags for embedding a SWF player:

- A single <*embed*> tag (for IE + other browsers).
- An <*object*> tag with nested <*embed*> tag (the first one for IE, the second for other browsers).
- An <*object*> tag with nested <*object*> tag (the first one for IE, the second for other browsers).

We recommend using the single <*embed*> tag. This works in all current-day browsers (including IE6) and provides the shortest codes. Here is an example embed code that does exactly the same as the SWFObject example above:

```
<embed
  flashvars="file=/data/bbb.mp4&autostart=true"
  allowfullscreen="true"
  allowscripaccess="always"
  id="player1"
  name="player1"
  src="player.swf"
  width="480"
  height="270"
/>
```

As you can see, most of the data of the SWFObject embed is also in here:

- The **container** is now the embed tag itself. The *fallback* text cannot be used anymore.
- The **flashvars** are merged into a single string, and loaded as an attribute. You should always concatenate the flashvars using so-called querystring parameter encoding: *flashvar1=value1&flashvar2=value2&....*
- The **params** each are individual attributes of the embed tag.
- The **attributes** also are individual attributes of the embed tag.
- The **instantiation** options (source, width, height) are attributes of the embed tag.

Note: As you can see, the Flash version reference is not in the embed tag: this is one of the drawbacks of this method: it's not possible to sniff for Flash and selectively hide it, e.g. if the flash version is not sufficient or if the device (iPad ...) doesn't support Flash.

CHAPTER

THREE

CONFIGURATION OPTIONS

Here's a list of all configuration options (flashvars) the player accepts. Options are entered in the *embed code* to set how the player looks and functions.

3.1 Encoding

First, a note on encoding. You must URL encode the three glyphs **? = &** inside flashvars, because of the way these flashvars are loaded into the player (as a querystring). The urlencoded values for these symbols are listed here:

- ? → %3F
- = → %3D
- & → %26

If, for example, your **file** flashvar is at the location *getplaylist.php?id=123&provider=flv*, you must encode the option to:

```
getplaylist.php%3Fid%3D123%26provider%3Dflv
```

The player will automatically URLdecode every option it receives.

3.2 Playlist properties

To load a playlist, only a single flashvar is required:

playlistfile (undefined)
: Location of an *XML playlist* to load into the player.

The following flashvars can be set instead of **playlistfile**. They are used to create a playlist with a single item. They set various properties of the *media item* to load (e.g. the source file or preview image or title). Those properties are:

duration (0)
: Duration of the file in seconds. Set this to present the duration in the controlbar before the video starts. It can also be set to a shorter value than the actual file duration. The player will restrict playback to only that section.

file (undefined)
: Location of the file or playlist to play, e.g. *http://www.mywebsite.com/myvideo.mp4*.

image (undefined)
: Location of a preview (poster) image; shown in display before the video starts.

mediaid (undefined)
Unique string (e.g. *9Ks83JsK*) used to identify this media file. Is used by certain plugins, e.g. for the targeting of advertisements. The player itself doesn't use this ID anywhere.

provider (undefined)
Set this flashvar to tell the player in which format (regular/streaming) the player is. By default, the **provider** is detected by the player based upon the file extension. If there is no suiteable extension, it can be manually set. The following provider strings are supported:

- **video**: progressively downloaded FLV / MP4 video, but also AAC audio. See *Media Support*.
- **sound**: progressively downloaded MP3 files. See *Media Support*.
- **image**: JPG/GIF/PNG images. See *Media Support*.
- **youtube**: videos from Youtube. See *Media Support*.
- **http**: FLV/MP4 videos using HTTP pseudo-streaming. See *HTTP Pseudostreaming*.
- **rtmp**: FLV/MP4/MP3 files or live streams using RTMP streaming. See *HTTP Pseudostreaming*.

Note: In addition to these built-in providers, it is possible to load custom providers into the JW Player, e.g. for specific CDN support. Custom providers are packed in a separate SWF file, much like a **plugin**.

A number of custom providers is available from our ' addons repository <http://www.longtailvideo.com/addons/>'_. Third party developers interested in building a custom provider should check our our developer site, which includes documentation and a MediaProvider SDK.

start (0)
Position in seconds where playback should start. This option works for *HTTP Pseudostreaming*, *RTMP Streaming* and the MP3 and Youtube *files*. It does not work for regular videos.

streamer (undefined)
Location of an RTMP or HTTP server instance to use for streaming. Can be an RTMP application or external PHP/ASP file. See *RTMP Streaming* and *HTTP Pseudostreaming*.

Note: Technically, any playlist item property is also available as an option. In practice though, the properties *author*, *date*, *description*, *link*, *tags* and *title* are not used anywhere if a single media file is loaded.

3.3 Layout

These flashvars control the looks of the player.

controlbar (bottom)
Position of the controlbar. Can be set to *bottom*, *over* and *none*.

dock (false)
set this to **true** to list plugin buttons in display. By default (*false*), plugin buttons are shown in the controlbar.

playlist (none)
Position of the playlist. Can be set to **bottom**, **right**, **left**, **over** or **none**.

playlistsize (180)
When the playlist is positioned below the display, this option can be used to change its height. When the playlist lives left or right of the display, this option represents its width. In the other cases, this option isn't needed.

skin (undefined)
Location of a so-called **skin**, a file with graphics that drastically changes the look of the player. There are two types of skins available:

- **XML/PNG skins**: These skins consist of an XML file with settings and a bunch of PNG images. The files are packed up in a ZIP, so they load fast over the internet. Building your own skin is extremely easy and can be done with any basic image and text editor. See *XML/PNG Skinning* for more info.
- **SWF skins**: These skins consist of a single SWF file, built using Adobe Flash. This type of skins has been supported since the 4.0 player. Since SWF skins can only be built using Flash (a $500+ package) and since this skinning model can easily break, SWF skins are considered deprecated in favor of PNG skins.

Our AddOns repository contains a list of available skins.

3.4 Behavior

These flashvars control the playback behavior of the player.

autostart (false)
Set this to *true* to automatically start the player on load.

bufferlength (1)
Number of seconds of the file that has to be loaded before the player starts playback. Set this to a low value to enable instant-start (good for fast connections) and to a high value to get less mid-stream buffering (good for slow connections).

id (undefined)

Unique identifier of the player in the HTML DOM. You only need to set this option if you want to use the *JavaScript API* and want to target Linux users.

The ID is needed by JavaScript to get a reference to the player. On Windows and Mac OS X, the player automatically reads the ID from the *id* and *name* attributes of the player's *HTML embed code* <*embedding*>. On Linux however, this functionality does not work. Setting the **id** option in addition to the HTML attributes will fix this problem.

item (0)
Playlist item that should start to play. Use this to start the player with a specific item instead of with the first item.

mute (false)
Mute the sounds on startup. Is saved in a cookie.

playerready (undefined)
By default, the player calls a *playerReady()* JavaScript function when it is initialized. This option is used to let the player call a different function after it's initialized (e.g. *registerPlayer()*).

plugins (undefined)
A powerful feature, this is a comma-separated list of plugins to load (e.g. **hd,viral**). Plugins are separate SWF files that extend the functionality of the player, e.g. with advertising, analytics or viral sharing features. Visit our addons repository to browse the long list of available plugins.

repeat (none)
What to do when the mediafile has ended. Has several options:

- **none**: do nothing (stop playback) whever a file is completed.
- **list**: play each file in the playlist once, stop at the end.
- **always**: continously play the file (or all files in the playlist).
- **single**: continously repeat the current file in the playlist.

shuffle (false)
Shuffle playback of playlist items. The player will randomly pick the items.

smoothing (true)
This sets the smoothing of videos, so you won't see blocks when a video is upscaled. Set this to **false** to disable the feature and get performance improvements with old computers / big files.

stretching (uniform)
Defines how to resize the poster image and video to fit the display. Can be:

- **none**: keep the original dimensions.
- **exactfit**: disproportionally stretch the video/image to exactly fit the display.
- **uniform**: stretch the image/video while maintaining its aspect ratio. There'll be black borders.
- **fill**: stretch the image/video while maintaining its aspect ratio, completely filling the display.

volume (90)
Startup audio volume of the player. Can be 0 to 100.

3.5 Logo

Unlicensed copies of the JW Player contain a small watermark that pops up when the player is buffering. In licensed copies of the player, this watermark is empty by default. It is possible to place your own watermark in the player using the following options:

logo.file (undefined)
Location of an external JPG, PNG or GIF image to be used as watermark. PNG images with transparency give the best results.

logo.link (undefined)
HTTP link to jump to when the watermark image is clicked. If it is not set, a click on the watermark does nothing.

logo.linktarget (_blank)
Link target for logo click. Can be *_self*, *_blank*, *_parent*, *_top* or a named frame.

logo.hide (true)
By default, the logo will automatically show when the player buffers and hide 5 seconds later. When this option is set *false*, the logo will stay visible all the time.

logo.position (bottom-left)
This sets the corner in which to display the watermark. It can be one of the following:

- **bottom-left**
- **bottom-right**
- **top-left**
- **top-right**

Note: Once again: the logo options can only be used for licensed players!

3.6 Colors

These options are available when either using no skin or when using skins built with the older SWF skinning model (these skins have the extension *.swf*). These color options will be deprecated once SWF skinning support is dropped in a future release.

backcolor (ffffff)
background color of the controlbar and playlist. This is white by default.

frontcolor (000000)
color of all icons and texts in the controlbar and playlist. Is black by default.

lightcolor (000000)
Color of an icon or text when you rollover it with the mouse. Is black by default.

screencolor (000000)
Background color of the display. Is black by default.

The four color flashvars must be entered using hexadecimal values, as is common for web colors (e.g. *FFCC00* for bright yellow).

3.7 Config XML

All options can be listed in an XML file and then fed to the player with a single option:

config (undefined)
location of a XML file with flashvars. Useful if you want to keep the actual embed codes short. Here's an example:

Here is an example of such an XML file:

```
<config>
  <file>files/bunny.mp4</file>
  <image>files/bunny.jpg</image>
  <repeat>true</repeat>
  <backcolor>333333</backcolor>
  <volume>40</volume>
  <controlbar>over</controlbar>
</config>
```

Options set in the embed code will overwrite those set in the config XML.

Note: Due to the *Crossdomain Security Restrictions* restrictions of Flash, you cannot load a config XML from one domain in a player on another domain. This issue can be circumvented by placing a *crossdomain.xml* file on the server that hosts your XML.

CHAPTER

FOUR

MEDIA SUPPORT

This document lists all media file formats the JW Player supports: video, sound, images and Youtube clips.

Single media files can be grouped using *playlists* and streamed over *http* or *rtmp* instead of downloaded. Both options do not change the set of supported media formats.

Note: The player always tries to recognize a file format by its extension. If no suitable extension is found, **the player will presume you want to load a playlist**! Work around this issue by setting the *provider option*.

4.1 Video

The player supports video (*provider=video*) in the following formats:

`H.264 ( .mp4, .mov, .f4v )`
: Video in either the MP4 or *Quicktime <http://en.wikipedia.org/wiki/Quicktime>* container format. These files must contain video encoded with the H.264 codec and audio encoded with the AAC codec. H264/AAC video is today's format of choice. It can also be played on a wide range of (mobile) devices.

  Note: If you cannot seek within an MP4 file be before it is completely downloaded, the cause of this problem is that the so-called MOOV atom (which contains the seeking information) is located at the end of your video. Check out this little application to parse your videos and fix it.

`FLV ( .flv )`
: Video in the Flash Video container format. These files can contain video encoded with both the ON2 VP6 codec and the Sorenson Spark codec. Audio must be in the MP3 codec. FLV is a slightly outdated format. It is also unique to Flash.

  Note: If the progress bar isn't running with your FLV file, or if your video dimensions are wrong, this means that your FLV file doesn't have metadata. Fix this by using the small tool from buraks.com.

`3GPP ( .3gp, .3g2 )`
: Video in the 3GPP container format. These files must contain video encoded with the H.263 codec and audio encoded with the AAC codec. Used widely for mobile phones because it is easy to decode. More and more devices switch to H264 though.

`AAC ( .aac, .m4a )`
: Audio encoded with the AAC codec. Indeed, this is not video! However, the player must use the **video** provider to playback this audio, since the **sound** provider only supports MP3. State of the art codec, widely supported.

4.2 Sound

The player supports sounds (*provider=sound*) in the following format:

MP3 (.mp3)
Audio encoded with the MP3 codec. Though not as good as AAC, MP3 is very widely used. It is also supported by nearly any device that can play audio.

Note: If you encounter too fast or too slow playback of MP3 files, it contains variable bitrate encoding or unsupported sample frequencies (eg 48Khz). Please stick to constant bitrate encoding and 44 kHz. The free iTunes software has an MP3 encoder built-in.

4.3 Images

The player supports images (*provider=image*) in the following formats:

JPEG (.jpg)
Images encoded with the JPEG algorythm. No transparency support.

PNG (.png)
Images encoded with the PNG algorythm. Supports transparency.

GIF (.gif)
Images encoded with the GIF algorythm. Supports transparency, but pixels can only be opaque or 100% transparent.

Note: The player does NOT support animated GIFs.

SWF (.swf)
Drawings/animations encoded in the Adobe Flash format. Supports transparency.

Note: Though SWF files load in the player, it is discouraged to use them. The player cannot read the duration and dimensions of SWF files. Custom scripts inside these SWF files might also interfere with (or break) playback.

4.4 Youtube

The player includes native support for playing back Youtube videos (*provider=youtube*). Youtube playback is automatically enabled when the **file** option is assigned to the URL of a Youtube video (e.g. *http://www.youtube.com/watch?v=WuQnd3d9IuA*).

The player uses the official Youtube API for this functionality, so this is definitely not a hack. Youtube officially support playback of its content in third-party players like the JW Player.

The Youtube API is accessed through a bridge, the separate **yt.swf** file included in the player download.

Note: In order for Youtube videos to play, you must upload the *yt.swf* file to the same directory as the *player.swf*.

4.5 Custom Providers

The JW Player has built-in support for two distinct streaming providers, *RTMP Streaming* and *HTTP Pseudo-Streaming*.

In addition to the built-in media support, it is possible to load custom media playback **providers** into the JW Player, e.g. to support specific features of a certain CDN. Custom providers are packed in a separate SWF file, much like a *plugin*.

A number of custom providers is available from our addons repository.

Third party developers interested in building a custom provider should check out our developer site, which includes documentation and and SDK for building providers.

CHAPTER

FIVE

PLAYLIST SUPPORT

First, note that playlist XML files are subject to the *Crossdomain Security Restrictions* of Flash. This means that a videoplayer on one domain cannot load a playlist from another domain. It can be fixed by placing a *crossdomain.xml* file at the server the playlist is loaded from.

If your playlist and player.swf are hosted on the same domain, these restrictions don't apply.

5.1 Supported XML Formats

That said, the following playlist formats are supported:

- ASX feeds
- ATOM feeds with Media extensions
- RSS feeds with iTunes extensions and Media extensions
- XSPF feeds

Here is an overview of all the tags of each format the player processes, and the property in the JW Player playlist they correspond to:

JW Player	XSPF	RSS	itunes:	media:	ASX	ATOM
author	creator	(none)	author	credit	author	(none)
date	(none)	pubDate	(none)	(none)	(none)	published
description	annotation	description	summary	description	abstract	summary
duration	duration	(none)	duration	content	duration	(none)
file	location	enclosure	(none)	content	ref	(none)
link	info	link	(none)	(none)	moreinfo	link
image	image	(none)	(none)	thumbnail	(none)	(none)
provider	(none)	(none)	(none)	(none)	(none)	(none)
start	(none)	(none)	(none)	(none)	starttime	(none)
streamer	(none)	(none)	(none)	(none)	(none)	(none)
tags	(none)	category	keywords	keywords	(none)	(none)
title	title	title	(none)	title	title	title

All **media:** tags can be embedded in a **media:group** element. A **media:content** element can also act as a container.

Here is an example playlist (with one video) in the most widely used format: *RSS* with *media:* extensions:

```
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss/">
  <channel>
    <title>Example playlist</title>
```

```
    <item>
      <title>Big Buck Bunny</title>
      <link>http://www.bigbuckbunny.org/</link>
      <description>Big Buck Bunny is a short animated film by the Blender Institute,
        part of the Blender Foundation.</description>
      <pubDate>Sat, 07 Sep 2002 09:42:31 GMT</pubDate>
      <media:content url="/videos/bbb.mp4" duration="33" />
      <media:thumbnail url="/thumbs/bbb.jpg" />
    </item>

  </channel>
</rss>
```

In order to load this playlist into the player, save it as an XML file, upload it to your webserver and point the player to it using the *playlistfile option*.

5.2 JWPlayer Namespace

In order to enable all JW Player playlist properties for all feed formats, the player contains a **jwplayer** namespace. By inserting this into your feed, properties that are not supported by the feed format itself (such as the **streamer**) can be amended without breaking validation. Any of the entries listed in the above table can be inserted. Here's an example, of a video that uses *RTMP Streaming*:

```
<rss version="2.0" xmlns:jwplayer="http://developer.longtailvideo.com/">
  <channel>
    <title>Example RSS feed with jwplayer extensions</title>
    <item>
      <title>Big Buck Bunny</title>
      <jwplayer:file>videos/nPripu9l-60830.mp4</jwplayer:file>
      <jwplayer:streamer>rtmp://myserver.com/myApp/</jwplayer:streamer>
      <jwplayer:duration>34</jwplayer:duration>
    </item>
  </channel>
</rss>
```

Pay attention to the top level tag, which describes the JW Player namespace with the xmlns attribute. This must be available in order to not break validity.

5.3 Mixing namespaces

You can mix **jwplayer** elements with both the regular elements of a feed and elements from the mRSS and iTunes extensions. If multiple elements match the same playlist entry, the elements will be prioritized:

- Elements that are defined by the feed format (e.g. the *enclosure* in RSS) get the lowest priority.
- Elements defined by the *itunes* namespace rank third.
- Element defined by the *media* namespace (e.g. *media:content*) rank second.
- Elements defined by the *jwplayer* extension always gets the highest priority.

This feature allows you to set, for example, a specific video version or HTTP/RTMP streaming for the JW Player, while other feed aggregators will pick the default content. In the above example feed, we could insert a regular *enclosure* element that points to a download of the video. This would make the feed useful for both the JW Player and text-oriented aggregators such as Feedburner.

5.4 Adding properties

Certain plugins (e.g. *captions* and *hd*) and providers (*http* and *rtmp*) support item-specific configuration options. These are placed inside **jwplayer** tags as well, and are inserted like this:

```
<rss version="2.0" xmlns:jwplayer="http://developer.longtailvideo.com/">
  <channel>
    <title>Example RSS feed with playlistitem extensions</title>
    <item>
      <title>First video</title>
      <enclosure url="/files/bunny.flv" type="video/x-flv" length="1192846" />
      <jwplayer:provider>http</jwplayer:provider>
      <jwplayer:http.startparam>start</jwplayer:http.startparam>
      <jwplayer:captions.file>/files/captions_1.xml</jwplayer:captions.file>
    </item>

    <item>
      <title>Second Video</title>
      <enclosure url="/files/bunny.mp4" type="video/mp4" length="1192846" />
      <jwplayer:provider>http</jwplayer:provider>
      <jwplayer:http.startparam>starttime</jwplayer:http.startparam>
      <jwplayer:captions.file>/files/captions_2.xml</jwplayer:captions.file>
    </item>
  </channel>
</rss>
```

Notice that the **<jwplayer:http.startparam>** and **<jwplayer:captions.file>** properties are set differently for each of the playlist items.

CHAPTER

SIX

HTTP PSEUDOSTREAMING

Both MP4 and FLV videos can be played back with a mechanism called HTTP Pseudostreaming. This mechanism allows your viewers to seek to not-yet downloaded parts of a video. Youtube is an example site that offers this functionality. HTTP pseudostreaming is enabled by setting the *option provider=http* in your player.

HTTP pseudostreaming combines the advantages of straight HTTP downloads (it passes any firewall, viewers on bad connections can simply wait for the download) with the ability to seek to non-downloaded parts. The only drawbacks of HTTP Pseudostreaming compared to Flash's official *RTMP Streaming* are its reduced security (HTTP is easier to sniff than RTMP) and long loading times when seeking in large videos (> 15 minutes).

HTTP Pseudostreaming should not be confused with HTTP Dynamic Streaming. The latter is a brand-new mechanism currently being developed by Adobe that works by chopping up the original video in so-called *chunks* of a few seconds each. The videoplayer seamlessly glues these chunks together again. The JW Player does **not yet** support HTTP Dynamic Streaming.

6.1 Servers

HTTP Pseudostreaming does not work by default on any webserver. A serverside module is needed to enable it. Here are the two most widely used (and open source) modules for this:

- The H264 streaming module for Apache, Lighttpd, IIS and NginX. It supports MP4 videos.
- The FLV streaming module for Lighttpd. It supports FLV videos.

Several CDN's (Content Delivery Networks) support HTTP Pseudostreaming as well. We have done succesfull tests with Bitgravity, CDNetworks, Edgecast and Limelight.

Instead of using a serverside module, pseudostreaming can also be enabled by using a serverside script (in e.g. PHP or .NET). We do not advise this, since such a script consumes a lot of resources, has security implications and can only be used with FLV files. A much-used serverside script for pseudostreaming is Xmoov-PHP.

6.2 Mechanism

Under the hood, HTTP pseudostreaming works as follows:

When the video is initially loaded, the player reads and stores a list of *seekpoints* as part of the video's metadata. These seekpoints are offsets in the video (both in seconds and in bytes) at which a new *keyframe* starts. At these offsets, a request to the server can be made.

When a user seeks to a not-yet-downloaded part of the video, the player translates this seek to the nearest seekpoint. Next, the player does a request to the server, with the seekpoint offset as a parameter. For FLV videos, the offset is always provided in bytes:

```
http://www.mywebsite.com/videos/bbb.flv?start=219476905
```

For MP4 videos, the offset is always provided in seconds:

```
http://www.mywebsite.com/videos/bbb.mp4?starttime=30.4
```

The server will return the video, starting from the offset. Because the first frame in this video is a keyframe, the player is able to correctly load and play it. Should the server have returned the video from an arbitrary offset, the player would not be able to pick up the stream and the display would only show garbage.

Note: Some FLV encoders do not include seekpoints metadata when encoding videos. Without this data, HTTP Pseudostreaming will not work. If you suspect your videos to not have metadata, use our Metaviewer plugin to inspect the video. There should be a *seekpoints* or *keyframes* list. If it is not there, use the FLVMDI tool to parse your FLV videos and inject this metadata.

6.3 Startparam

When the player requests a video with an offset, it uses *start* as the default offset parameter:

```
http://www.mywebsite.com/videos/bbb.flv?start=219476905
http://www.mywebsite.com/videos/bbb.mp4?start=30.4
```

This name is most widely used by serverside modules and CDNs. However, sometimes a CDN uses a different name for this parameter. In that case, use the option *http.startparam* to set a custom offset parameter name. Here are some examples of CDNs that use a different name:

- The H264 streaming module uses *http.startparam=starttime* for MP4 videos.
- Bitgravity uses *http.startparam=apstart* for FLV videos and *http.startparam=starttime* for MP4 videos.
- Edgecast uses *http.startparam=ec_seek* for FLV videos.
- Limelight uses *http.startparam=fs* for FLV videos.

Here's what an example SWFObject *embed code* looks like when both HTTP Pseudostreaming and a custom start parameter is enabled:

```
<div id='container'>The player will be placed here</div>

<script type="text/javascript">
  var flashvars = {
    file:'http://bitcast-a.bitgravity.com/botr/bbb.mp4',
    provider:'http',
    'http.startparam':'starttime'
  };

  swfobject.embedSWF('player.swf','container','480','270','9.0.115','false', flashvars,
   {allowfullscreen:'true',allowscriptaccess:'always'},
   {id:'jwplayer',name:'jwplayer'}
  );
</script>
```

6.4 Playlists

HTTP Pseudostreaming can also be enabled in playlists, by leveraging the *JWPlayer namespace*. Both the *provider* and *http.startparam* options can be set for every entry in a playlist. In this case, you don't have to set them in the embed code (just point the *file* to your playlist).

Here's an example, an RSS feed with a single video:

```
<rss version="2.0" xmlns:jwplayer="http://developer.longtailvideo.com/">
  <channel>
    <title>Playlist with HTTP Pseudostreaming</title>

    <item>
      <title>Big Buck Bunny</title>
      <description>Big Buck Bunny is a short animated film by the Blender Institute,
         part of the Blender Foundation.</description>
      <enclosure url="http://myserver.com/botr/bbb.mp4" type="video/mp4" length="3192846" />
      <jwplayer:provider>http</jwplayer:provider>
      <jwplayer:http.startparam>apstart</jwplayer:http.startparam>

    </item>
  </channel>
</rss>
```

Instead of the *enclosure* element, you can also use the *media:content* or *jwplayer:file* element. More info in *Playlist Support*.

Note: Do not forget the **xmlns** at the top of the feed. It is needed by the player (and any other feed reader you might use) to understand the *jwplayer:* elements.

6.5 Bitrate Switching

Like *RTMP Streaming*, HTTP Pseudostreaming includes the ability to dynamically adjust the video quality for each individual viewer. We call this mechanism *bitrate switching*.

To use bitrate swiching, you need multiple copies of your MP4 or FLV video, each with a different quality (dimensions and bitrate). These multiple videos are loaded into the player using an mRSS playlist (see example below). The player recognizes the various *levels* of your video and automatically selects the highest quality one that:

- Fits the *bandwidth* of the server » client connection.
- Fits the *width* of the player's display (or, to be precise, is not more than 20% larger).

As a viewer continues to watch the video, the player re-examines its decision (and might switch) in response to certain events:

- On **startup**, immediately after it has calculated the bandwidth for the first time.
- On a **fullscreen** switch, since the *width* of the display then drastically changes. For example, when a viewer goes fullscreen and has sufficient bandwidth, the player might serve an HD version of the video.
- On every **seek** in the video. Since the player has to rebuffer-the stream anyway, it takes the opportunity to also check if bandwidth conditions have not changed.

Note that the player will not do a bandwidth switch if extreme bandwidth changes cause the video to re-buffer. In practice, we found such a heuristic to cause continous switching and an awful viewing experience. *RTMP Streaming* on the other hand, is able to switch seamlessly in response to bandwidth fluctuations.

6.5.1 Example

Here is an example bitrate switching playlist (only one item). Note that it is similar to a *regular* HTTP Pseudostreaming playlist, with the exception of the multiple video elements per item. The mRSS extension is the only way to provide these multiple elements including *bitrate* and *width* attributes:

```
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss/"
  xmlns:jwplayer="http://developer.longtailvideo.com/">
  <channel>
    <title>Playlist with HTTP Bitrate Switching</title>

    <item>
      <title>Big Buck Bunny</title>
      <description>Big Buck Bunny is a short animated film by the Blender Institute,
         part of the Blender Foundation.</description>
      <media:group>
        <media:content bitrate="1800" url="http://myserver.com/bbb-486.mp4"  width="1280" />
        <media:content bitrate="1100" url="http://myserver.com/bbb-485.mp4" width="720"/>
        <media:content bitrate="700" url="http://myserver.com/bbb-484.mp4" width="480" />
        <media:content bitrate="400" url="http://myserver.com/bbb-483.mp4" width="320" />
      </media:group>
      <jwplayer:provider>http</jwplayer:provider>
      <jwplayer:http.startparam>apstart</jwplayer:http.startparam>
    </item>

  </channel>
</rss>
```

Some hints:

- The *bitrate* attributes must be in kbps, as defined by the mRSS spec. The *width* attribute is in pixels.
- It is recommended to order the streams by quality, the best one at the beginning. Most RSS readers will pick this one.
- The four levels displayed in this feed are actually what we recommend for bitrate switching of widescreen MP4 videos. For 4:3 videos or FLV videos, you might want to increase the bitrates or decrease the dimensions a little.
- Some publishers only modify the bitrate when encoding multiple levels. The player can work with this, but modifying both the bitrate + dimensions allows for more variation between the levels (and re-use of videos, e.g. the smallest one for streaming to phones).
- The *media:group* element here is optional, but it organizes the video links a little.

CHAPTER

SEVEN

RTMP STREAMING

RTMP (Real Time Messaging Protocol) is a system for delivering on-demand and live media to Adobe Flash applications (like the JW Player). RTMP supports video in FLV and H.264 (MP4/MOV/F4V) *formats* and audio in MP3 and AAC (M4A) *formats*. RTMP offers several advantages over regular HTTP video downloads:

- RTMP can do live streaming - people can watch your video while it is being recorded.
- With RTMP, viewers can seek to not-yet-downloaded parts of a video. This is especially useful for longer-form content (> 10 minutes).
- Videos delivered over RTMP (and its secure brother, RTMPE) are harder to steal than videos delivered over regular HTTP.

However, do note that RTMP has its disadvantages too. Especially since the introduction of *HTTP Pseudostreaming* (which also offer seeking to not-yet-downloaded parts), RTMP is not the only option for professional video delivery. Some drawbacks to be aware of:

- RTMP is a different protocol than HTTP and is sent over a different port (1935 instead of 80). Therefore, RTMP is frequently blocked by (corporate) firewalls. This can be circumvented by using RTMPT (tunneled), but this comes at a performance cost (longer buffer times - 2x in our experience).
- RTMP is a *true* streaming protocol, which means that the bandwidth of the connection must always be larger than the datarate of the video. If the connection drops for just a few seconds, the stream will stutter. If the connection overall is just a little less than the video datarate, the video will not play at all. With *HTTP Pseudostreaming* on the other hand, people can simply wait until more of the video is downloaded.

The JW Player supports a wide array of features of the RTMP protocol.

7.1 Servers

In order to use RTMP, your webhoster or CDN needs to have a dedicated RTMP webserver installed. There are three major offerings, all supported by the JW Player:

- The Flash Media Server from Adobe is the de facto standard. Since Flash is also developed by Adobe, new video functionalities always find their way in FMS first.
- The Wowza Media Server from Wowza is a great alternative, because it includes support for other streaming protocols than RTMP (for e.g. Shoutcast, the iPhone or Silverlight).
- The Red5 Media Server is an open-source RTMP alternative. It lags in features (e.g. no dynamic streaming), but has an active and open community of developers.

RTMP servers are not solely used for one-to-many media streaming. They include support for such functionalities as video conferencing, document sharing and multiplayer games. Each of these functionalities is separately set up on the

server in what is called an *application*. Every application has its own URL (typically a subfolder of the root). For example, these might be the path to both an on-demand streaming and live streaming application on your webserver:

```
rtmp://www.myserver.com/ondemand/
rtmp://www.myserver.com/live/
```

7.2 Options

To play an RTMP stream in the player, both the *streamer* and *file options* must be set. The *streamer* is set to the server + path of your RTMP application. The *file* is set to the internal URL of video or audio file you want to stream. Here is an example *embed code*:

```
<div id='container'>The player will be placed here</div>

<script type="text/javascript">
  var flashvars = {
    file:'library/clip.mp4',
    streamer:'rtmp://www.myserver.com/ondemand/'
  };

  swfobject.embedSWF('player.swf','container','480','270','9.0.115','false', flashvars,
   {allowfullscreen:'true',allowscriptaccess:'always'},
   {id:'jwplayer',name:'jwplayer'}
  );
</script>
```

Note that the documentation of RTMP servers tell you to set the *file* option in players like this:

- For FLV video: **file=clip** (without the *.flv* extension).
- For MP4 video: **file=mp4:clip.mp4** (with *mp4:* prefix).
- For MP3 audio: **file=mp3:song.mp3** (with *mp3:* prefix).
- For AAC audio: **file=mp4:song.aac** (with *mp4:* prefix).

You do not have to do this with the JW Player, since the player takes care of stripping the extension or adding the prefix. If you do add the prefix yourself, the player will recognize it and not modify the URL.

Additionally, the player will leave querystring variables (e.g. for certain CDN security mechanisms) untouched. It basically ignores everything after the **?** character. However, because of the way options are *loaded* into Flash, it is not possible to plainly use querystring delimiters (*?*, *=*, *&*) inside the *file* or *streamer* option. This issue can be circumvented by *URL encoding these characters*.

7.3 Playlists

RTMP streams can also be included in playlists, by leveraging the *JWPlayer namespace*. The *streamer* option should be set for every RTMP entry in a playlist. You don't have to set them in the embed code (just point the *file* option to your playlist).

Here's an example, an RSS feed with an RTMP video and audio clip:

```
<rss version="2.0" xmlns:jwplayer="http://developer.longtailvideo.com/">
  <channel>
    <title>Playlist with RTMP streams</title>

    <item>
      <title>Big Buck Bunny</title>
      <description>Big Buck Bunny is a short animated film by the Blender Institute,
         part of the Blender Foundation.</description>
      <enclosure url="files/bbb.mp4" type="video/mp4" length="3192846" />
      <jwplayer:streamer>rtmp://myserver.com/ondemand</jwplayer:streamer>
    </item>

    <item>
      <title>Big Buck Bunny (podcast)</title>
      <description>Big Buck Bunny is a short animated film by the Blender Institute,
         part of the Blender Foundation.</description>
      <enclosure url="files/bbb.mp3" type="audio/mp3" length="3192846" />
      <jwplayer:streamer>rtmp://myserver.com/ondemand</jwplayer:streamer>
    </item>

  </channel>
</rss>
```

Instead of the *enclosure* element, you can also use the *media:content* or *jwplayer:file* element. You could even set the *enclosure* to a regular http download of the video ánd *jwplayer:file* to the RTMP stream. That way, this single feed is useful for both regular RSS readers and the JW Player. More info in *Playlist Support*.

Note: Do not forget the **xmlns** at the top of the feed. It is needed by the player (and any other feed reader you might use) to understand the *jwplayer:* elements.

7.4 Live Streaming

A unique feature of RTMP is the ability to do live streaming, e.g. of presentations, concerts or sports events. Next to the player and an RTMP server, one then also needs a small tool to *ingest* (upload) the live video into the server. There's a bunch of such tools available, but the easiest to use is the (free) Flash Live Media Encoder. It is available for Windows and Mac.

A live stream can be embedded in the player using the same options as an on-demand stream. The only difference is that a live stream has no file extension. Example:

```
<div id='container'>The player will be placed here</div>

<script type="text/javascript">
  var flashvars = {
    file:'livepresentation',
    streamer:'rtmp://www.myserver.com/live/'
  };

  swfobject.embedSWF('player.swf','container','480','270','9.0.115','false', flashvars,
   {allowfullscreen:'true',allowscriptaccess:'always'},
   {id:'jwplayer',name:'jwplayer'}
  );
</script>
```

7.4.1 Subscribing

When streaming live streams using the Akamai or Limelight CDN, players cannot simply connect to the live stream. Instead, they have to *subscribe* to it, by sending an **FCSubscribe call** to the server. The JW Player includes support for this functionality. Simply add the *rtmp.subscribe=true* option to your embed code to enable:

```
<div id='container'>The player will be placed here</div>

<script type="text/javascript">
  var flashvars = {
    file:'livepresentation',
    streamer:'rtmp://www.myserver.com/live/',
    'rtmp.subscribe':'true'
  };

  swfobject.embedSWF('player.swf','container','480','270','9.0.115','false', flashvars,
   {allowfullscreen:'true',allowscriptaccess:'always'},
   {id:'jwplayer',name:'jwplayer'}
  );
</script>
```

7.4.2 DVR Live Streaming

Flash Media Server 3.5, introduced DVR live streaming - the ability to pause and seek in a live stream. This functionality is supported by the JW Player. It can be enabled by setting the option **rtmp.dvr=true**.

By default, a DVR stream acts like a regular on-demand stream, the only difference being that the *duration* of the stream keeps increasing. This leads to a slightly awkward user experience, since the time scrubber in the controlbar keeps bouncing around in one position instead of moving to the right.

To solve this issue, also set the *duration* option to the total duration of your live event (or, to be safe, a few minutes longer). That way the time scrubber will function normally. The *live head* of the event is then indicated by the download progress bar in the player. If a user seeks beyond that point, he will automatically get pushed to that head. Here's an example of DVR Live Streaming with duration (3600 seconds is 1 hour):

```
<div id='container'>The player will be placed here</div>

<script type="text/javascript">
  var flashvars = {
    file:'livepresentation',
    streamer:'rtmp://www.myserver.com/live/',
    'rtmp.dvr':'true',
    'duration':'3600'
  };

  swfobject.embedSWF('player.swf','container','480','270','9.0.115','false', flashvars,
   {allowfullscreen:'true',allowscriptaccess:'always'},
   {id:'jwplayer',name:'jwplayer'}
  );
</script>
```

Note: DVR Live Streaming only works in combination with Adobe's Live Media Encoder and an RTMP server that has DVR enabled.

7.5 Dynamic Streaming

Like with *HTTP Pseudostreaming*, RTMP Streaming includes the ability to dynamically optimize the video quality for each individual viewer. Adobe calls this mechanism *dynamic streaming*. This functionality is supported for FMS 3.5+ and Wowza 2.0+.

To use dynamic streaming, you need multiple copies of your MP4 or FLV video, each with a different quality (dimensions and bitrate). These multiple videos are loaded into the player using an mRSS playlist (see example below). The player recognizes the various *levels* of your video and automatically selects the highest quality one that:

- Fits the *bandwidth* of the server » client connection.
- Fits the *width* of the player's display (or, to be precise, is not more than 20% larger).

As a viewer continues to watch the video, the player re-examines its decision (and might switch) in response to certain events:

- On a **bandwidth** increase or decrease - the bandwidth is re-calculated at an interval of 2 seconds.
- On a **resize** of the player. For example, when a viewer goes fullscreen and has sufficient bandwidth, the player might serve an HD version of the video.

Unlike with *HTTP Pseudostreaming*, a dynamic streaming switch is unobtrusive. There'll be no re-buffering or audible/visible hickup. It does take a few seconds for a switch to occur in response to a bandwidth change / player resize, since the server has to wait for a *keyframe* to do a smooth switch and the player always has a few seconds of the old stream in its buffer. To keep stream switches fast, make sure your videos are encoded with a small (2 to 4 seconds) keyframe interval.

Note: So far, we have not been able to combine dynamic streaming with live streaming. This functionality is highlighted in documentation from Adobe and Wowza, but in our tests we found that the bandwidth the player receives never exceeds the bandwidth of the level that currently plays. In other words: the player will never switch to a higher quality stream than the one it starts with. This seems to be a bug in the Flash plugin, since both FMS and Wowza have this issue.

7.5.1 Example

Here is an example dynamic streaming playlist (only one item). It is similar to a regular RTMP Streaming playlist, with the exception of the multiple video elements per item. The mRSS extension is the only way to provide these multiple elements including *bitrate* and *width* attributes:

```
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss/"
  xmlns:jwplayer="http://developer.longtailvideo.com/">
  <channel>
    <title>Playlist with RTMP Dynamic Streaming</title>

    <item>
      <title>Big Buck Bunny</title>
      <description>Big Buck Bunny is a short animated film by the Blender Institute,
         part of the Blender Foundation.</description>
      <media:group>
        <media:content bitrate="1800" url="videos/Qvxp3Jnv-486.mp4"  width="1280" />
        <media:content bitrate="1100" url="videos/Qvxp3Jnv-485.mp4" width="720"/>
        <media:content bitrate="700" url="videos/Qvxp3Jnv-484.mp4" width="480" />
        <media:content bitrate="400" url="videos/Qvxp3Jnv-483.mp4" width="320" />
      </media:group>
      <jwplayer:streamer>rtmp://www.myserver.com/ondemand/</jwplayer:streamer>
    </item>
```

```
  </channel>
</rss>
```

Some hints:

- The *bitrate* attributes must be in kbps, as defined by the mRSS spec. The *width* attribute is in pixels.
- It is recommended to order the streams by quality, the best one at the beginning.
- The four levels displayed in this feed are actually what we recommend for bitrate switching of widescreen MP4 videos. For 4:3 videos or FLV videos, you might want to increase the bitrates or decrease the dimensions a little.
- Some publishers only modify the bitrate when encoding multiple levels. The player can work with this, but modifying both the bitrate + dimensions allows for more variation between the levels (and re-use of videos, e.g. the smallest one for streaming to mobile phones).
- The *media:group* element here is optional, but it organizes the video links a little.

7.6 Load Balancing

For high-volume publishers who maintain several RTMP servers, the player supports load-balancing by means of an intermediate XML file. This is used by e.g. the Highwinds and Streamzilla CDNs. Load balancing works like this:

- The player first requests the XML file (typically from a single *master* server).
- The server returns the XML file, which includes the location of the RTMP server to use (typically the server that's least busy).
- The player parses the XML file, connects to the server and starts the stream.

7.6.1 Example

Here's an example of such an XML file. It is in the SMIL format:

```
<smil>
  <head>
    <meta base="rtmp://server1234.mycdn.com/ondemand/" />
  </head>
  <body>
    <video src="library/myVideo.mp4" />
  </body>
</smil>
```

Here's an example embed code for enabling this functionality in the player. Note the *provider=rtmp option* is needed in addition to *rtmp.loadbalance*, since otherwise the player thinks the XML file is a playlist.

```
<div id='container'>The player will be placed here</div>

<script type="text/javascript">
  var flashvars = {
    file:'http://www.mycdn.com/videos/myVideo.mp4.xml',
    provider:'rtmp',
    'rtmp.loadbalance':'true'
  };

  swfobject.embedSWF('player.swf','container','480','270','9.0.115','false', flashvars,
```

```
    {allowfullscreen:'true',allowscriptaccess:'always'},
    {id:'jwplayer',name:'jwplayer'}
  );
</script>
```

7.6.2 Playlists

RTMP Load balancing in playlists works in a similar fashion: the *provider=rtmp* and *rtmp.loadbalance=true* options can be set for every entry in the playlist that uses loadbalancing. Here's an example with one item:

```
<rss version="2.0" xmlns:jwplayer="http://developer.longtailvideo.com/">
  <channel>
    <title>Playlist with RTMP loadbalancing</title>

    <item>
      <title>Big Buck Bunny (podcast)</title>
      <description>Big Buck Bunny is a short animated film by the Blender Institute,
         part of the Blender Foundation.</description>
      <enclosure url="http://www.mycdn.com/videos/bbb.mp3.xml" type="text/xml" length="185" />
      <jwplayer:provider>rtmp</jwplayer:provider>
      <jwplayer:rtmp.loadbalance>true</jwplayer:rtmp.loadbalance>
    </item>

  </channel>
</rss>
```

See the playlist section above for more information on format and element support.

Note: A combination of load balancing + dynamic streaming is not possible yet. We are working on such a functionality, which will be included in a future version of the player.

CHAPTER

EIGHT

XML/PNG SKINNING

With skins, you can customize the face of your JW player. You can alter the design of any of the player's four component parts – ControlBar, Display, Dock and Playlist, as well as skinning-enabled plugins. Before JW Player 5, it was only possible to build skins using Adobe Flash (which was difficult and error-prone). Now, with JW Player 5, designers can build skins in their graphical editor of choice, and save the visual elements as bitmaps. This allows for rapid prototyping without the need to compile a swf file, and opens up skinning to designers who don't have Flash experience or software.

8.1 Supported Graphics Formats

JW Player 5 will accept most commonly used bitmap image formats including:

- JPG
- GIF (*Allows Transparency*)
- PNG (8-Bit) (*Allows Transparency*)
- PNG (24-Bit) (*Allows Transparency and Partial Transparency*)

Examples in this guide will use the PNG file format. It is the preferred format for creating slick skins due to its partial transparency support.

JW Player 5.2 and up support the use of SWF assets in the XML skinning format. However, we recommend that designers restrict themselves to the bitmap formats above, since skins created using SWF assets will not compatible with the JW Player for HTML5.

Note: Animated gif files are not supported.

8.2 The XML Document

The XML (Extensible Markup Language) file, or document, contains all the settings for your skin – the color settings for text and dock elements; margins and font-sizes for the ControlBar; and paths to images for every element in the skin.

A player skin consists of its own settings and its components. Here is an example of an XML document before the elements have been defined:

8.2.1 Basic XML Structure

```
<?xml version="1.0"?>
<skin version="1.1" name="SkinName" author="http://www.yoursite.com/">
   <components>
      <component name="controlbar">
         <settings>
            <setting name="..." value="..." />
         </setting>
         <layout>
            ...
         </layout>
         <element name="..." src="..." />
         <element name="..." src="..." />
         <element name="..." src="..." />
      </component>
      <component name="display">
         <settings>
            <setting name="..." value="..." />
         </setting>
         <element name="..." src="..." />
         <element name="..." src="..." />
         <element name="..." src="..." />
      </component>
      <component name="dock">
         <settings>
            <setting name="..." value="..." />
         </setting>
         <element name="..." src="..." />
         <element name="..." src="..." />
         <element name="..." src="..." />
      </component>
      <component name="playlist">
         <settings>
            <setting name="..." value="..." />
         </setting>
         <element name="..." src="..." />
         <element name="..." src="..." />
         <element name="..." src="..." />
      </component>
   </components>
</skin>
```

8.2.2 Beginning Your XML Skin

The opening declaration of your XML document declares that it *IS* an XML document, and establishes that this is a JW Player skin. Inside the skin element are two attributes: *name* and *author*.

```
<?xml version="1.0"?>
<skin version="1.1" name="SkinName" author="http://www.yoursite.com/">
```

You can replace these with your skin's name and your website, or your own name if you'd prefer not to have your URL in the *author* attribute.

8.2.3 Linking to Images

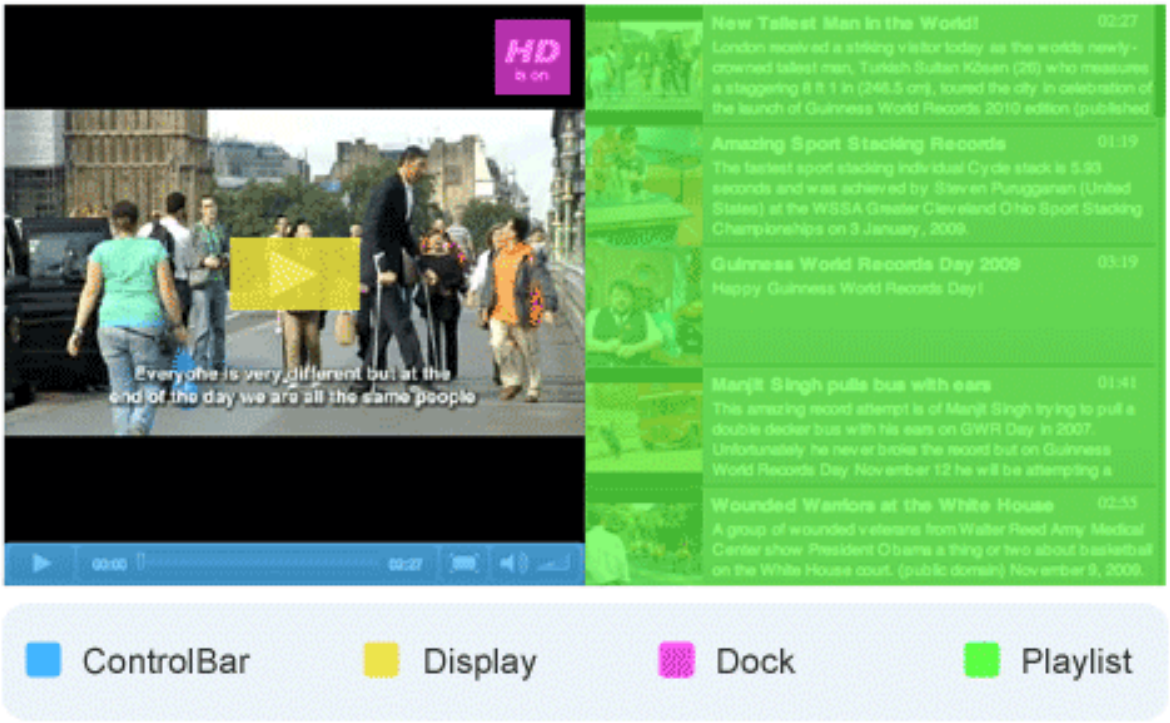
Images must reside in a subdirectory corresponding to their parent container of the skin's folder. For instance, Controlbar images should reside in the *controlbar* subdirectory.

8.2.4 Component sections

The player's controls are broken into four components. Each of these is defined in a **<component>** tag, and are all placed inside of the skin's **<components>** block. The player controls are:

- controlbar
- display
- dock
- playlist

In addition to player controls, it is also possible to define skinning elements for skinnable plugins as well. These would be placed in another **<component>** tag, with the *name* attribute corresponding to the id of the plugin.



8.3 The Controlbar

The ControlBar component is used more than any of the other JW Player skin components. It controls video playback, shows you your point in time, toggles to full-screen mode and allows you to control the volume.

8.3.1 Controlbar XML Syntax

```
<component name="controlbar">

   <settings>
      <setting name="backgroundcolor" value="0x000000"/>
      <setting name="margin" value="10" />
      <setting name="font" value="_sans" />
      <setting name="fontsize" value="10" />
      <setting name="fontcolor" value="0x000000" />
      <setting name="fontstyle" value="normal" />
      <setting name="fontweight" value="normal" />
      <setting name="buttoncolor" value="0xFFFFFF" />
   </settings>

   <elements>
      <element name="background" src="file.png" />
      <element name="capLeft" src="file.png" />
      <element name="capRight" src="file.png" />
      <element name="divider" src="file.png" />
      <element name="playButton" src="file.png" />
      <element name="playButtonOver" src="file.png" />
      <element name="pauseButton" src="file.png" />
      <element name="pauseButtonOver" src="file.png" />
      <element name="timeSliderRail" src="file.png" />
      <element name="timeSliderBuffer" src="file.png" />
      <element name="timeSliderProgress" src="file.png" />
      <element name="timeSliderThumb" src="file.png" />
      <element name="fullscreenButton" src="file.png" />
      <element name="fullscreenButtonOver" src="file.png" />
      <element name="normalscreenButton" src="file.png" />
      <element name="normalscreenButtonOver" src="file.png" />
      <element name="muteButton" src="file.png" />
      <element name="muteButtonOver" src="file.png" />
      <element name="unmuteButton" src="file.png" />
      <element name="unmuteButtonOver" src="file.png" />
      <element name="volumeSliderRail" src="file.png" />
      <element name="volumeSliderBuffer" src="file.png" />
      <element name="volumeSliderProgress" src="file.png" />
      ...
   </elements>

   <layout>
      ...
   </layout>
</component>
```

8.3.2 Controlbar Settings

In the example above, you will notice the bit of code containing the settings element for the ControlBar component. Here is a list of the Controlbar settings, along with their default values:

backgroundcolor (undefined)
: Color to display underneath the controlbar. If the controlbar elements are transparent or semi-transparent, this color will show beneath those elements. If this is not set, the Flash stage will be visible beneath the controlbar.

margin (0)

This is the margin which will wrap around the controlbar when the player is fullscreen mode, or when the player's *controlbar* setting is set to **over**. The value is in pixels.

font (_sans)
The font face for the Controlbar's text fields, **elapsed** and **duration**. (*_sans*, *_serif*, *_typewriter*)

fontsize (10)
The font size of the Controlbar's text fields.

fontweight (normal)
The font weight for the Controlbar's text fields. (*normal*, *bold*)

fontstyle (normal)
The font style for the Controlbar's text fields. (*normal*, *italic*)

fontcolor (undefined)
The color for the Controlbar's text fields.

buttoncolor (undefined)
The color for any custom Controlbar icons.

8.3.3 Controlbar Elements

The controlbar contains a single background element:

background
The background is a graphic which stretches horizontally to fit the width of the Controlbar. *capLeft* and *capRight* (see below) are placed to the left and right of the background.

The Controlbar has a few elements which allow you to add space between elements. They are non-functioning bitmaps meant to give space to the right and left edges of the Controlbar.

capLeft
The left cap graphic to your controlbar skin

capRight
The right cap graphic to your controlbar skin

divider
A separator element between buttons and sliders. *(this same element can appear multiple times)*

Note: JW Player 5.1 and below will fail to load without the **capLeft**, **capRight** and **volumeSlider** elements in the XML File. This issue was resolved in version 5.2.

Next, there are the buttons. Controlbar buttons have two states. The **button** state is visible when the mouse is not hovering over the button. The **buttonOver** state – which should have the same dimensions as **button** – is shown when the user hovers the mouse above the button. Here's a list of all buttons with their states:

- **playButton /** playButtonOver
- **pauseButton /** pauseButtonOver
- **prevButton /** prevButtonOver
- **nextButton /** nextButtonOver
- **stopButton /** stopButtonOver
- **fullscreenButton /** fullscreenButtonOver
- **normalscreenButton /** normalscreenButtonOver
- **muteButton /** muteButtonOver

- **unmuteButton /** unmuteButtonOver
- **blankButton /** blankButtonOver

The **blankButton** element is used when plugins insert additional buttons into the Controlbar. This element should simply be a button background; the foreground element will be added by the plugins.

Certain buttons replace each other depending on the state of the JW Player. For instance, when a video is playing, the **playButton** is replaced by the **pauseButton** element. Toggle button pairs:

- **playButton /** pauseButton
- **fullscreenButton /** normalscreenButton
- **muteButton /** unmuteButton

Next to the caps and buttons, there's the two sliders (for time and volume). The **timeSlider** is a unique block built using several elements stacked on top of each other. Of those elements, three of them automatically scale to a width based on the free space in the player. Those elements are:

`timeSliderRail`
: the *background* graphic which serves as the frame for the timeSlider

`timeSliderBuffer`
: the file's buffer indicator

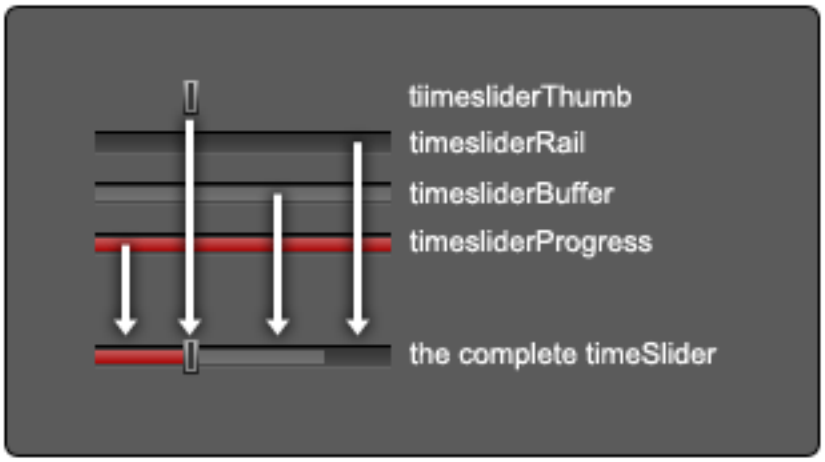
`timeSliderProgress`
: the file's progress indicator

With that in mind it is important to design your elements to gracefully scale horizontally. The **timeSliderBuffer** and **timeSliderProgress** elements dynamically scale to indicate a percentage of progress of the total file length. Additional, non-scaling **timeSlider** elements are:

`timeSliderThumb`
: serves as a handle which can be dragged across the progress bar to allow the user to specify a seek position.

`timeSliderCapLeft`
: Left-hand end-cap, placed to the left of the **timeSliderRail** element.

`timeSliderCapRight`
: Right-hand end-cap, placed to the right of the **timeSliderRail** element.



The **volumeSlider** element is quite similar to the **timeSlider**, except that it does not scale automatically. It will be as large as graphics you produce.

volumeSliderRail
: the **background** graphic which serves as the frame for the volumeSlider

volumeSliderBuffer
: this shows the potential volume the slider can have.

volumeSliderProgress
: this is shows the current level at which the volumeSlider is set.

volumeSliderThumb
: the handle to slide the volume, also indicates the volume level.

volumeSliderCapLeft
: Left-hand end-cap, placed to the left of the **volumeSliderRail** element.

volumeSliderCapRight
: Right-hand end-cap, placed to the right of the **volumeSliderRail** element.

  the handle to slide the volume, also indicates the volume level.

Note: JW Player 5.1's skinning model will add 5 pixels of padding to each side of the **volumeSlider** if no end-caps are specified. JW Player 5.1 and below will fail to load without the **volumeSliderRail** element in the XML file.

Two text fields can be laid out in the controlbar:

elapsed
: Amount of time elapsed since the start of the video (format: mm:ss)

duration
: Duration of the currently playing video (format: mm:ss)

8.4 Controlbar Layout

The controlbar's components (*buttons*, *text fields*, *sliders* and *dividers*) are laid out according to a block of XML code in the Controlbar section.

8.4.1 Layout XML Syntax

Inside the controlbar's **<component>** block, you can insert an optional **<layout>** block which allows you to override the default controlbar layout.

```
<layout>
   <group position="left">
      <button name="play" />
```

```
      <divider />
      <button name="prev" />
      <divider />
      <button name="next" />
      <divider />
      <button name="stop" />
      <divider />
      <text name="duration" />
      <divider />
   </group>
   <group position="center">
      <slider name="time" />
   </group>
   <group position="right">
      <text name="elapsed" />
      <divider />
      <button name="blank" />
      <divider />
      <button name="mute" />
      <slider name="volume" />
      <divider />
      <button name="fullscreen" />
   </group>
</layout>
```

8.4.2 Layout Groups

The Controlbar's layout is made up of three groupings, *left*, *right* and *center*.

- **Left**:Elements placed in the **<group position="left">** tag will be placed left to right and be left-aligned.
- **Center**: Elements placed in the **<group position="center">** tag will be placed between the *left* and *right* groups. Furthermore, if the **timeSlider** element is placed here, it will be stretched to any space not assigned to other elements.
- **Right**: Elements placed in the **<group position="right">** tag will be placed left to right and be right-aligned.

8.4.3 Layout Elements

The **<group>** tag can contain the following elements:

<button name="..." />
: Used to place the Controlbar button elements described above. For example, the **play** button would appear as **<button name="play" />**

<text name="..." />
: Used to place the Controlbar text elements, **elapsed** and **duration**.

<slider name="..." />
: Used to place the Controlbar slider elements, **timeSlider** and **volumeSlider**.

<divider />
: Used to place the **divider** element. This tag can define two optional attributes (only one attribute may be used at a time):

- *element*: Allows an arbitrary element to be placed between other elements. If no *element* or *width* attribute is set, the default **divider** graphic is used. Example:

```
<divider element="alternate_divider" />
```

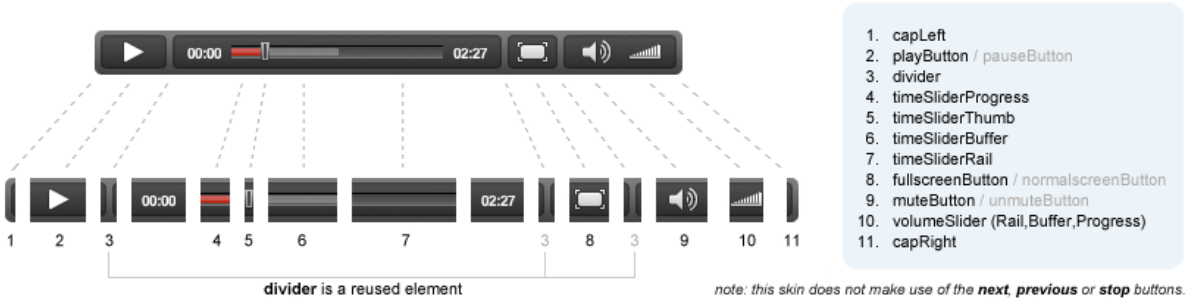
- *width*: If this attribute is set, the specified number of pixels will be placed into the layout. No graphical element will be used; the controlbar's **background** element will be visible. Example:

```
<divider width="10" />
```

8.4.4 Default Controlbar Layout

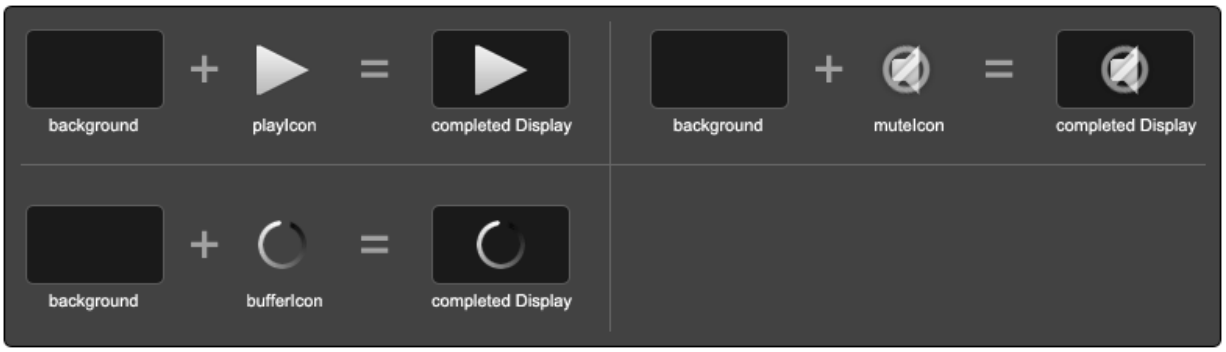
If no **<layout>** block is included in the skin, the player will use a default layout, based on which skin elements have been defined in the **<elements>** block. The elements will be layed out in the following order:

- capLeft
- playButton/pauseButton
- prevButton
- nextButton
- stopButton
- divider
- elapsedText
- timeSliderCapLeft
- timeSliderRail/timeSliderBuffer/timeSliderProgress/timeSliderThumb
- timeSliderCapRight
- durationText
- divider (*reused element*)
- blankButton
- divider (*reused element*)
- fullscreenButton/normalscreenButton
- divider (*reused element*)
- muteButton/unmuteButton
- volumeSliderCapLeft
- volumeSliderRail/volumeSliderBuffer/volumeSliderProgress/volumeSliderThumb
- volumeSliderCapRight
- capRight



8.5 The Display

The display largely consists of the buttons you see in the middle of the player. You see the familiar triangular **play** icon before the movie is playing, and also when you pause. When the user has muted the player, the **Mute** icon appears. Display Icons come in two parts: a global background element to every icon, and the icon itself, which is programmatically centered over the background layer. All images must reside in the *display* subdirectory of the skin.



Note: By default, the **bufferIcon** will slowly rotate clockwise. There are settings to influence this rotation.

8.5.1 Display XML Syntax

```
<component name="display">
   <settings>
      <settings>
         <setting name="backgroundcolor" value="0x000000" />
         <setting name="bufferrotation" value="15" />
         <setting name="bufferinterval" value="100" />
      </settings>
   </settings>
   <elements>
      <element name="background" src="file.png" />
      <element name="playIcon" src="file.png" />
      <element name="muteIcon" src="file.png" />
      <element name="bufferIcon" src="file.png" />
   </elements>
</component>
```

8.5.2 Display Settings

Here is a list of Display settings, along with their default values:

backgroundcolor (0x000000)
: This is the color of the player's display window, which appears behind any playing media.

bufferrotation (15)
: The number of degrees the buffer icon is rotated per rotation. A negative value will result in the buffer rotating counter-clockwise.

bufferinterval (100)
: The amount of time, in milliseconds between each buffer icon rotation.

8.6 The Dock

Dock Icons elements sit at the top right corner of your player and can be both informative or functional. For instance, if you've installed the HD plugin, once you've toggled High Definition Playback to ON, a small HD Dock Icon will appear in top corner of your player, letting you know you're watching in HD.

8.6.1 Dock XML Syntax

```
<component name="dock">
   <settings>
      <setting name="fontcolor" value="0x000000" />
   </settings>
   <elements>
      <element name="button" src="file.png" />
      <element name="buttonOver" src="file.png" />
   </elements>
</component>
```

8.6.2 Dock Settings

The dock's *settings* block contains only one setting:

fontcolor (0x000000)
: The color for the Dock buttons' text fields.

8.6.3 Dock elements

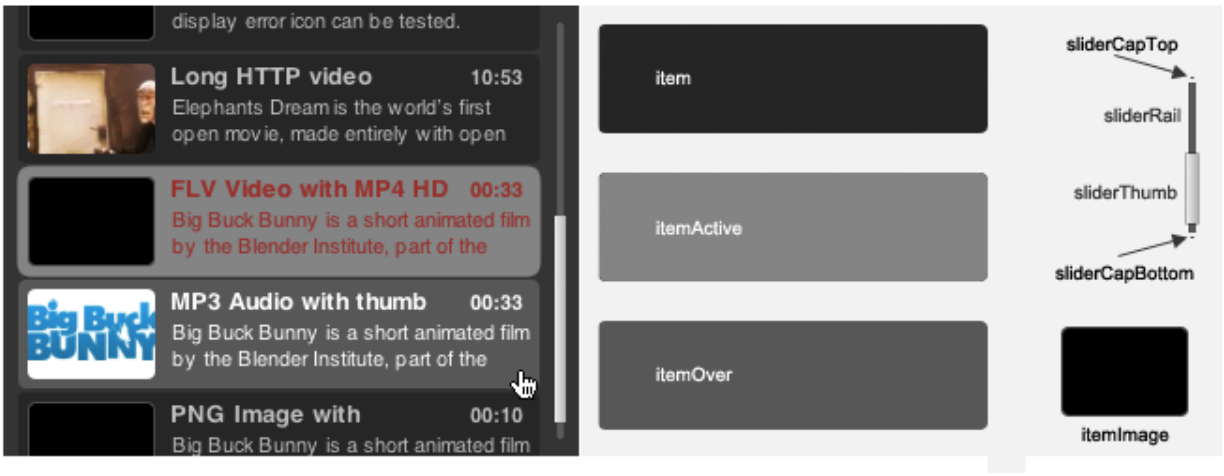
The Dock only has two elements:

button
: The background image of a dock button when the mouse is not rolled over it.

buttonOver
: The background image of a dock button when the mouse is rolled over it. Is not required.

8.7 The Playlist

There are two main Playlist skin elements; Playlist Items, and the Playlist Slider. Item graphics scale horizontally and are placed behind the description/thumbnail of videos in your playlist. The slider is a vertical scrollbar rail and handle (thumb), with optional top and bottom endcaps.



8.7.1 Playlist XML Syntax

```
<component name="playlist">
   <settings>
      <setting name="fontcolor" value="0x999999" />
      <setting name="overcolor" value="0xFFFFFF" />
      <setting name="activecolor" value="0x990000" />
      <setting name="backgroundcolor" value="0x000000"/>
      <setting name="font" value="_sans" />
      <setting name="fontsize" value="12" />
      <setting name="fontweight" value="normal" />
      <setting name="fontstyle" value="normal" />
   </settings>

   <elements>
      <element name="background" src="background.png" />
      <element name="item" src="item.png" />
      <element name="itemOver" src="itemOver.png" />
      <element name="itemActive" src="itemActive.png" />
      <element name="itemImage" src="itemImage.png" />
```

```
      <element name="sliderRail" src="sliderRail.png" />
      <element name="sliderThumb" src="sliderThumb.png" />
      <element name="sliderCapTop" src="sliderCapTop.png" />
      <element name="sliderCapBottom" src="sliderCapBottom.png" />
   </elements>
</component>
```

8.7.2 Playlist Settings

Here is a list of the settings that can be placed in the playlist's **<settings>** block, along with their default values:

fontcolor (undefined)
The color for the playlist's text fields.

overcolor (undefined)
The color for the playlist item's text fields when the mouse is hovering over an item.

activecolor (undefined)
The color for the playlist item's text fields when that item is the currently active item.

backgroundcolor (undefined)
The playlist's background color.

font (_sans)
Font used for the playlist's text fields (*_sans*, *_serif*, *_typewriter*)

fontsize (undefined)
Font size for the playlist's text fields. By default, the playlist item's title has a fontsize of 13 pixels; the rest of the fields are 11 pixels. If **fontsize** is set, all text fields will have the same font size.

fontstyle (normal)
Can be used to set the font style for the playlist's text fields (*normal*, *italic*)

fontweight (normal)
Can be used to set the font weight for the playlist's text fields (*normal*, *bold*)

8.7.3 Playlist elements

The following Playlist elements are available:

background
The *background* element serves as the default background of the playlist if there are fewer elements than the height of the playlist. It stretches in both the X and Y direction.

item
Background graphic for each playlist item. Stretch to the width of the playlist, minus the slider width (if necessary).

itemOver
Over state for **item**. Replaces **item** whenever the user mouses over.

itemImage
Image placeholder. This element is visible when the playlist item does not have an image associated with it. If the playlist item image is present, **itemImage**'s shape serves as a mask around the playlist item image. If the playlist item image has any transparency, **itemImage** will be visible behind it.

itemActive
: Active state for **item**. Replaces **item** whenever the corresponding playlist item is the currently playing/loaded playlist item.

sliderRail
: Background of the vertical slider. When the playlist's slider is visible, **sliderRail** is stretched to the height of the playlist, minus the height of any end caps.

sliderThumb
: Draggable thumb for the vertical slider. This element is stretched vertically, and is proportional to the visible area of the playlist versus its total size. For example, if 50% of the playlist items are currently visible in the playlist, the thumb will be 50% of the height of the playlist.

sliderCapTop
: Top end cap for the playlist slider. Placed above **sliderRail**.

sliderCapBottom
: Bottom end cap for the playlist slider. Placed below **sliderRail**.

8.8 Plugins

Some plugins allow their elements to be skinned as well. If so, you can place those elements directly in your skin, the same way you skin built-in player components. The *name* attribute must match the plugin's *id*. All plugin elements must be placed in a folder whose name also matches the plugin's *id*.

In the following example, the skin defines the HD plugin's two skinnable elements:

```
<component name="hd">
   <elements>
      <element name="dockIcon" src="dockIcon.png" />
      <element name="controlbarIcon" src="controlbarIcon.png" />
   </elements>
</component>
```

8.9 Packaging your Skin

Packaging your skin is as easy as zipping the skin XML file along with the subfolders containing the component graphics.

8.9.1 Zip File Structure

The XML file should named the same as the skin itself. For example, a skin called *myskin* would contain an XML file called *myskin.xml*, and would be zipped into *myskin.zip*. All images belong in their corresponding folders and reside on the same level as the XML file.

- *skin_name*.xml
- controlbar (folder with images)
- display (folder with images)
- dock (folder with images)
- playlist (folder with images)

Once you have zipped everything up, using a skin is a matter of:

- Uploading the skin to your server
- Setting the *skin option* in your player's *embed code* to the URL of the ZIP file.

8.9.2 Example skins

A number of example skins can be freely downloaded from our addons repository. Feel free to tweak any of these skins to make them fit your site design.

CHAPTER

NINE

JAVASCRIPT API

The JW Player for Flash supports a flexible JavaScript API. It is possible to read the config/playlist variables of the player, send events to the player (e.g. to pause or load a new video) and listen (and respond) to player events. A small initialization routine is needed to connect your apps to the player.

9.1 Initialization

Please note that the player will **NOT** be available the instant your HTML page is loaded and the first JavaScript is executed. The SWF file (90k) has to be loaded and instantiated first! You can catch this issue by defining a simple global JavaScript function. By default, it is called *playerReady()* and every player that's successfully instantiated will call it.

```
var player;
function playerReady(object) {
  alert('the player is ready');
  player = document.getElementById(object.id);
};
```

The *object* the player sends to the function contains the following properties:

id
: ID of the player (the <*embed*> code) in the HTML DOM. Use it to get a reference to the player with *getElementById()*.

version
: Exact version of the player in MAJOR.MINOR.REVISION format *e.g. 5.2.1065*.

client
: Plugin version and platform the player uses, e.g. *FLASH WIN 10.0.47.0*.

id (undefined)
: Unique identifier of the player in the HTML DOM. You only need to set this option if you want to use the *JavaScript API* and want to target Linux users.

Note: On Windows and Mac OS X, the player automatically reads its *ID* from the *id* and *name* attributes of the player's *HTML embed code* <*embedding*>. On Linux however, this functionality **does not work**. If you target Linux users with your scripting, you can circumvent this issue by including an ref:*id option* <*options-behavior*> in your list of flashvars in the embed code.

9.1.1 Custom playerready

It is possible to ask the player to call a different javascript function after it completes its initialization. This can be done with an *option* called **playerready**. Here is an example SWFObject :ref:‘ embed code <embedding>‘ using the function *registerPlayer()*:

```
<p id="container1">You don’t have Flash ...</p>

<script type="text/javascript">
  var flashvars = { file:’/data/bbb.mp4’,playerready:’registerPlayer’ };
  var params = { allowfullscreen:’true’, allowscriptaccess:’always’ };
  var attributes = { id:’player1’, name:’player1’ };
  swfobject.embedSWF(’player.swf’,’container1’,’480’,’270’,’9.0.115’,’false’,
    flashvars, params, attributes);

  var player;
  function registerPlayer(obj) {
    alert(’The player with ID ’+obj.id + ’is ready!’);
    player = document.getElementById(obj.id);
  };
</script>
```

9.1.2 No playerready

If you are not interested in calling the player immediately after the page loads, you won’t need the *playerReady()* function. You can then simply use the ID of the embed/object tag that embeds the player to get a reference. So for example with this embed tag:

```
<embed id="myplayer" name="myplayer" src="/upload/player.swf" width="400" height="200" />
```

You can get a pointer to the player with this line of code:

```
var player = document.getElementById(’myplayer’);
```

Note: Note you must add both the **id** and **name** attributes in the <*embedding*> in order to get back an ID in all browsers.

9.2 Reading variables

There’s two variable calls you can make through the API: *getConfig()* and *getPlaylist()*.

9.2.1 getConfig()

getConfig() returns an object with state variables of the player. For example, here we request the current audio volume, the current player width and the current playback state:

```
var volume = player.getConfig().volume;
var width = player.getConfig().width;
var state = player.getConfig().state;
```

Here’s the full list of state variables:

bandwidth
Current bandwidth of the player to the server, in kbps (e.g. *1431*). This is only available for the :ref:video <mediaformats>‘, *http* and *rtmp* providers.

fullscreen
Current fullscreen state of the player, as boolean (e.g. *false*).

height
Current height of the player, in pixels (e.g. *270*).

item
Currently active (playing, paused) playlist item, as zero-index (e.g. *0*). Note that *0* means the first playlistitem is playing and *1* means the second one is playing.

level
Currently active bitrate level, in case multipe bitrates are supplied to the player. This is only useful for *HTTP Pseudostreaming* and *RTMP Streaming*. Note that *0* always refers to the highest quality bitrate.

position
current playback position, in seconds (e.g. *13.2*).

state
Current playback state of the player, as an uppercase string. It can be one of the following:

- *IDLE*: The current playlist item is not loading and not playing.
- *BUFFERING*: the current playlistitem is loading. When sufficient data has loaded, it will automatically start playing.
- *PLAYING*: the current playlist item is playing.
- *PAUSED*: playback of the current playlistitem is not paused by the player.

mute
Current audio mute state of the player, as boolean (e.g. *false*).

volume
Current audio volume of the player, as a number from 0 to 100 (e.g. *90*).

width
Current width of the player, in pixels (e.g. *480*).

Note: In fact, all the *Configuration Options* will be available in the response to *getConfig()*. In certain edge cases, this might be useful, e.g. when you want to know if the player did **autostart** or not.

9.2.2 getPlaylist()

getPlaylist() returns the current playlist of the player as an array. Each entry of this array is in turn again a hashmap with all the *playlist properties* the player recognizes. Here's a few examples:

```
var playlist = player.getPlaylist();
alert("There are " + playlist.length + " videos in the playlist");
alert("The title of the first entry is " + playlist[0].title);
alert("The poster image of the second entry is " + playlist[1].image);
alert("The media file of the third entry is " + playlist[2].file);
alert("The media provider of the fourth entry is " + playlist[3].provider);
```

Playlist items can contain properties supported by the provider. Examples of such properties are:

- **http.startparam**, when using the *HTTP provider*.
- **rtmp.loadbalance**, when using the *RTMP provider*.

Playlist items can also contain properties supported by certain plugins. Examples of such properties are:

- **hd.file**, which is used by the HD plugin.
- **captions.file**, which is used by the Captions plugin.

More information, and the full list of 12 default playlist properties, can be found in *Playlist Support*.

9.3 Sending events

The player can be controlled from JavaScript by sending events (e.g. to pause it or change the volume). Sending events to the player is done through the *sendEvent()* call. Some of the event need a parameter and some don't. Here's a few examples:

```
// this will toggle playback.
player.sendEvent("play");
// this sets the volume to 90%
player.sendEvent("volume","true");
// This loads a new video in the player
player.sendEvent('load','http://www.mysite.com/videos/bbb.mp4');
```

Here's the full list of events you can send, plus their parameters:

item (index:Number)
: Start playback of a specific item in the playlist. If *index* isn't set, the current playlistitem will start.

link (index:Number)
: Navigate to the *link* of a specific item in the playlist. If *index* is not set, the player will navigate to the link of the current playlistitem.

load (url:String)
: Load a new media file or playlist into the player. The *url* must always be sent.

mute (state:Boolean)
: Mute or unmute the player's sound. If the *state* is not set, muting will be toggled.

next
: Jump to the next entry in the playlist. No parameters.

play (state:Boolean)
: Play (set *state* to *true*) or pause (set *state* to *false*) playback. If the *state* is not set, the player will toggle playback.

prev
: Jump to the previous entry in the playlist. No parameters.

seek (position:Number)
: Seek to a certain position in the currently playing media file. The *position* must be in seconds (e.g. *65* for one minute and five seconds).

  Note: Seeking does not work if the player is in the *IDLE* state. Make sure to check the *state* variable before attempting to seek. Additionally, for the *video* media *provider*, the player can only seek to portions of the video that are already loaded. Other media providers do not have this additional restriction.

stop
: Stop playback of the current playlist entry and unload it. The player will revert to the *IDLE* state and the poster image will be shown. No parameters.

volume (percentage:Number)
: Change the audio volume of the player to a certain percentage (e.g. *90*). If the player is muted, it will automatically be unmuted when a volume event is sent.

Note: Due to anti-phishing restrictions in the Adobe Flash runtime, it is not possible to enable/disable fullscreen playback of the player from JavaScript.

9.4 Setting listeners

In order to let JavaScript respond to player updates, you can assign listener functions to various events the player fires. An example of such event is the *volume* one, when the volume of the player is changed. The player will call the listener function with one parameter, a *key:value* populated object that contains more info about the event.

In the naming of the listener functions, the internal architecture of the JW Player sines through a little. Internally, the player is built using a Mode-View-Controller design pattern:

- The *Model* takes care of the actual media playback. It sends events to the View.
- The *View* distributes all events from the Model to the plugins and API. It also collects all input from the plugins and API.
- The *Controller* receives and checks all events from the View. In turn, it sends events to the Model.

Basically, the events from the View are those you send out using the *sendEvent()* API function. With two other API functions, you can listen to events from the Model (playback updates) and Controller (control updates). These API functions are *addModelListener()* and *addControllerListener()*. Here's a few examples:

```
function stateTracker(obj) {
   alert('the playback state is changed from '+obj.oldstate+' to '+obj.newstate);
};
player.addModelListener("state","stateTracker");

function volumeTracker(obj) {
   alert('the audio volume is changed to: '+obj.percentage'+ percent');
};
player.addControllerListener("volume","volumeTracker");
```

If you only need to listen to a certain event for a limited amount of time (or just once), use the *removeModelListener()* and removeControllerListener()* functions to unsubscribe your listener function. The syntax is exactly the same:

```
player.removeModelListener("state","stateTracker");
player.removeControllerListener("volume","volumeTracker");
```

Note: You MUST string representations of a function for the function parameter!

9.4.1 Model events

Here's an overview of all events the *Model* sends. Note that the data of every event contains the *id*, *version* and *client* parameters that are also sent on *playerReady*.

error
: Fired when a playback error occurs (e.g. when the video is not found or the stream is dropped). Data:

  •*message* (String): the error message, e.g. *file not found* or *no suiteable playback codec found*.

loaded
: Fired while the player is busy loading the currently playing media item. This event is never sent for *RTMP Streaming*, since that protocol does not preload content. Data:

  •*loaded* (Number): the number of bytes of the media file that are currently loaded.

- *total* (Number): the total filesize of the media file, in bytes.
- *offset* (Number): the byte position of the media file at which loading started. This is always 0, except when using *HTTP Pseudostreaming*.

meta

Fired when metadata on the currently playing media file is received. The exact metadata that is sent with this event varies per individual media file. Here are some examples:

- *duration* (Number) : sent for *video*, *youtube*, *http* and *rtmp* media. In seconds.
- *height* (Number): sent for all media providers, except for *youtube*. In pixels.
- *width* (Number): sent for all media providers, except for *youtube*. In pixels.
- Codecs, framerate, seekpoints, channels: sent for *video*, *http* and *rtmp* media.
- TimedText, captions, cuepoints: additional metadata that is embedded at a certain position in the media file. Sent for *video*, *http* and *rtmp* media.
- ID3 info (genre, name, artist, track, year, comment): sent for MP3 files (the *sound media provider*).

Note: Due to the *Crossdomain Security Restrictions* restrictions of Flash, you cannot load a ID3 data from an MP3 on one domain in a player on another domain. This issue can be circumvented by placing a *crossdomain.xml* file on the server that hosts your MP3s.

state

Fired when the playback state of the video changes. Data:

- *oldstate* ('IDLE','BUFFERING','PLAYING','PAUSED','COMPLETED'): the previous playback state.
- *newstate* ('IDLE','BUFFERING','PLAYING','PAUSED','COMPLETED'): the new playback state.

Note: You will not be able to check if a video is completed by polling for *getConfig().state*. The player will only be in the COMPLETED state for a very short time, before jumping to IDLE again. Always use *addModelListener('state',...)* if you want to check if a video is completed.

time

Fired when the playback position is changing (i.e. the media file is playing). It is fired with a resolution of 1/10 second, so there'll be a lot of events! Data:

- *duration* (Number): total duration of the media file in seconds, e.g. *150* for two and a half minutes.
- *position* (Number): current playback position in the file, in seconds.

9.4.2 Controller events

Here's an overview of all events the *Controller* sends. Note that the data of every event contains the *id*, *version* and *client* parameters that are also sent on *playerReady*.

item

Fired when the player switches to a new playlist entry. The new item will immediately start playing. Data:

- *index* (Number): playlist index of the media file that starts playing.

mute

Fired when the player's audio is muted or unmuted. Data:

- *state* (Boolean): the new mute state. If *true*, the player is muted.

play

Fired when the player toggles playback (playing/paused). Data:

•*state* (Boolean): the new playback state. If *true*, the player plays. If *false*, the player pauses.

playlist
Fired when a new playlist (a single file is also pushed as a playlist!) has been loaded into the player. Data:

•*playlist* (Array): The new playlist. It has exactly the same structure as the return of the *getPlaylist()* call.

resize
Fired when the player is resized. This includes entering/leaving fullscreen mode. Data:

•*fullscreen* (Boolean): The new fullscreen state. If *true*, the player is in fullscreen.

•*height* (Number): The overall height of the player.

•*width* (Number): The overall width of the player.

seek
Fired when the player is seeking to a new position in the video/sound/image. Parameters:

•*position* (Number): the new position in the file, in seconds (e.g. *150* for two and a half minute).

stop
Fired when the player stops loading and playing. The playback state will turn to *IDLE* and the position of a video will be set to 0. No data.

volume
Fired when the volume level is changed. Data:

•*percentage* (Number): new volume percentage, from 0 to 100 (e.g. *90*).

CHAPTER

TEN

CROSSDOMAIN SECURITY RESTRICTIONS

The Adobe Flash Player contains a crossdomain security mechanism, similar to JavaScript's Cross-Site Scripting restrictions. Flash's security model denies certain operations on files that are loaded from a different domain than the *player.swf*. Roughly speaking, three basic operations are denied:

- Loading of XML files (such as *playlists* and *configs*).
- Loading of SWF files (such as *SWF skins*).
- Accessing raw data of media files (such as *ID3 metadata*, sound waveform data or image bitmap data).

Generally, file loads (XML or SWF) will fail if there's no crossdomain access. Attempts to access or manipulate data (ID3, waveforms, bitmaps) will abort.

10.1 Crossdomain XML

Crossdomain security restrictions can be lifted by hosting a crossdomain.xml file on the server that contains the files. This crossdomain file must be placed in the root of your (sub)domain, for example:

```
http://www.myserver.com/crossdomain.xml
http://videos.myserver.com/crossdomain.xml
```

Before the Flash Player attempts to load XML files, SWF files or raw data from any domain other than the one hosting the *player.swf*, it checks the remote site for the existence of such a *crossdomain.xml* file. If Flash finds it, and if the configuration permits external access of its data, then the data is loaded. The file is not loaded or the data is not shown.

10.1.1 Allow All Example

Here's an example of a *crossdomain.xml* that allows access to the domain's data from SWF files on any site:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

Our *plugins.longtailvideo.com* domain includes such a crossdomain file, so players from any domain can load the plugins hosted there.

Note that this example sets your server wide open. Any SWF file can load any data from your site, which might lead to sercurity issues.

10.1.2 Restrict Access Example

Here is another example *crossdomain.xml*, this time permitting SWF file access from only a number of domains:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <allow-access-from domain="*.domain1.com"/>
  <allow-access-from domain="www.domain2.com"/>
</cross-domain-policy>
```

Note the use of the wildcard symbol: any subdomain from *domain1* can load data, whereas *domain2* is restricted to only the *www* subdomain.

Crossdomain policy files can even further finegrain access, e.g. to certain ports or HTTP headers. For a detailed overview, see Adobe's Crossdomain documentation.

CHAPTER

ELEVEN

RELEASE NOTES

11.1 Version 5.2

11.1.1 Build 1065

New Features

Version 5.2 introduces a number of new features to the XML/PNG skinning model.

- Support for customized font settings (face, weight, style, color) in controlbar and playlist text fields.
- Ability to set custom *backgroundcolor* for each element.
- Ability to set custom *overcolor* and *activecolor* for playlist items.

  These colorization settings replace the generic *backcolor*, *frontcolor*, *lightcolor* and *screencolor* *options*, allowing for more fine-grained control.
- Customized controlbar layout:
  - Allows placement of any button, text field or slider available in the controlbar
  - Adds the ability to insert arbitrary divider images
  - Adds the ability to insert arbitrary *spacer* elements
- New skinning elements:
  - Left and right end caps for time and volume sliders (*timeSliderCapLeft*, *timeSliderCapRight*, *volumeSliderCapLeft*, *volumeSliderCapRight*)
  - Active state for playlist item background (*itemActive* element)
  - Image placeholder for playlist item images (*itemImage* element)
  - Top and bottom end caps for playlist slider (*sliderCapTop*, *sliderCapBottom*)
  - Background images for text fields (*elapsedBackground*, *durationBackground*)
  - Over states for display icons (*playIconOver*, *muteIconOver*, *bufferIconOver*)
- Ability to control rate and amount of display *bufferIcon* rotation.
- Ability to use SWF assets in addition to JPGs and PNGs in XML skinning

An in-depth walkthrough of all new skinning features can be found in the *XML/PNG Skinning Guide*.

Bug Fixes

- Allows YouTube videos to be embedded in HTTPS pages
- Makes the YouTube logo clickable
- Fixes an issue where some dynamic streams only switch on resize events
- Fixes an issue which would cause dynamically switched RTMP livestreams to close early
- No longer hides the the display image when playing AAC or M4A audio files
- Allows querystring parameters for .flv files streamed over RTMP. This fixes a problem some Amazon CloudFront users were having with private content.

11.2 Version 5.1

11.2.1 Build 897

Bug Fixes

- Fixed an issue where load-balanced RTMP streams with bitrate switching could cause an error
- Fixed buffer icon centering and rotation for v5 skins

11.2.2 Build 854

New Features

- Since 5.0 branched off from 4.5, version 5.1 re-integrates changes from 4.6+ into the 5.x branch, including:
- Bitrate Switching
- Bandwidth detection
- DVR functionality for [wiki:FlashMediaServerDVR RTMP live streams].

Major Bug Fixes

- Allows loading images from across domains without *security restrictions*.
- Fixes some RTMP live/recorded streaming issues
- Fixes an issue where the *volume* flashvar is not respected when using RTMP
- Fixes issue where adjusting volume for YouTube videos doesn't work in Internet Explorer
- Various JavaScript API fixes
- Various visual tweaks
- Brings back icons=false functionality
- Brings back *id* flashvar, for Linux compatibility
- Better support of loadbalancing using the SMIL format

A full changelog can be found here

11.3 Version 5.0

11.3.1 Build 753

Features new to 5.0

- Bitmap Skinning (PNG, JPG, GIF)
- API Update for V5 plugins
- Player resizes plugins when needed
- Player sets X/Y coordinates of plugins
- Plugins can request that the player block (stop playback) or lock (disable player controls).
- MXMLC can be used to [browser:/trunk/fl5/README.txt compile the player].
- Backwards compatibility
- SWF Skins
- Version 4.x plugins
- Version 4.x JavaScript

4.x features not available in 5.0

- Bitrate switching, introduced in 4.6
- *displayclick* flashvar
- *logo* flashvar (for non-commercial players)

A full changelog can be found [/query?group=status&milestone=Flash+5.0&order=type here]