JW Player for Flash and HTML5

Release 5.3

Embedding Guide

September 29, 2010

CONTENTS

CHAPTER

ONE

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 three methods for embedding the JW Player:

- Using a generic JavaScript embedder (like SWFObject).
- Using a HTML tag (like <*object*> or <*embed*>).
- Using the JW Player's own JavaScript embedder (**jwplayer.js**).

We recommend using the included JW Player Embedder. It can detect if a browser supports Flash, it ensures that the player's *Player API* works and it avoids browser compatibility issues. Also, as of version 5.3, the JW Player Embedder allows you to use the player's HTML5 feautures. Detailed instructions can be found below.

1.1 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 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. Finally, to use the JW Player Embedder and HTML5 player, upload **jwplayer.js**.

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*.

Note: We recommend putting everything in a folder called "jwplayer" at the root of your site. This enables the *Quick Embed* method of setting up the player.

1.2 SWFObject

There's a wide array of good, open source libraries available for embedding Flash. **SWFObject** is 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 *Player 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**.

1.3 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: The Flash version reference is not in the embed tag: this is one of the drawbacks of this method: it's not possible to detect Flash and selectively hide it, e.g. if the flash version is not sufficient or if the device (iPad ...) doesn't support Flash.

1.4 JW Embedder

New in version 5.3, the JW Player features its own embedding method. While the previous embed methods can still be used, the built-in embed method has a couple of useful features:

- Seamless failover between the Flash and HTML5 players.
- Automatic integration with the *JavaScript API*.

1.4.1 Getting started

Embedding the JW Player in your website is a simple, 3-step process:

1. Upload the *jwplayer.js*, *player.swf* and *yt.swf* files from the download ZIP to your server. All other files in the download (documentation, source code, etc) are optional.
2. Include the *jwplayer.js* somewhere in the head of your website:

   ```
   <script type="text/javascript" src="/jwplayer/jwplayer.js"></script>
   ```

3. Call the player setup somewhere in the body of your website. Here's a basic example:

   ```
   <div id="container">Loading the player ...</div>

   <script type="text/javascript">
       jwplayer("container").setup({
   ```

```
            flashplayer: "/jwplayer/player.swf",
            file: "/uploads/video.mp4",
            height: 270,
            width: 480
        });
    </script>
```

When the page is loading, the JW Player is automatically instantiated on top of the <*div*>. By default, the player is rendered in Flash. If Flash is not supported (e.g. on an iPad), the player is rendered in HTML5.

The *flashplayer* option (to tell the javascript where the SWF resides) is just one of many configuration options available for configuring the JW Player.

Here's another setup example, this time using a <*video*> tag instead of a generic div:

```
<video
    file="/uploads/video.mp4"
    height="270"
    id="container"
    poster="/uploads/image.jpg"
    width="480">
</video>

<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf"
    });
</script>
```

In this case, the JW Player is actually inspecting <video> tag and loading its attributes as configuration options. It's a useful shortcut for setting up a basic player.

Quick Embed

If you've uploaded your *player.swf* and *jwplayer.js* files to a folder called "jwplayer" in the root of your website, you can embed the player by using two simple lines of HTML:

```
<script type="text/javascript" src="/jwplayer/jwplayer.js"></script>
<video class="jwplayer" src="/uploads/video.mp4" poster="/uploads/image.jpg"></video>
```

That's it! As long as you have everything in the right place, all <video> tags on your page whose class is **jwplayer** will be replaced on your page by the JW Player.

1.4.2 Setup Syntax

Let's take a closer look at the syntax of the *setup()* call. It has the following structure:

```
jwplayer(container).setup({options});
```

In this block, the *container* is the DOM element(<*video*> or <*div*>, <*p*>, etc.) you want to load the JW Player into. If the element is a <*video*> tag, the attributes of that tag (e.g. the *width* and *src*) are loaded into the player.

The *options* are the list of configuration options for the player. The configuration options guide contains the full overview. Here's an example with a bunch of options:

```
<div id="container">Loading the player ...</video>

<script type="text/javascript">
    jwplayer("container").setup({
        autostart: true,
        controlbar: "none",
        file: "/uploads/video.mp4",
        duration: 57,
        flashplayer: "/jwplayer/player.swf",
        volume: 80,
        width: 720
    });
</script>
```

Though generally a flat list, there are a couple of options that can be inserted as structured blocks inside the setup method. Each of these blocks allow for quick but powerful setups:

- **playlist**: allows inline setup of a full playlist, including metadata.
- **levels**: allows inline setup of multiple quality levels of a video, for bitrate switching purposes.
- **plugins**: allows inline setup of JW Player plugins, including their configuration options.
- **events**: allows inline setup of javascripts for player events, e.g. when you want to do something when the player starts.
- **players**: allows inline setup of a custom player fallback, e.g. HTML5 first, fallback to Flash.

The sections below explain them in detail.

1.4.3 Skins

The JW Player has a wide variety of skins that can be used to modify the look and feel of the player. They can be downloaded from our AddOns Library.

To embed a JW Player 5 skin, simply place the ZIP file on your web server and add the *skin* property to your embed code:

```
<div id="container">Loading the player ...</div>

<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf",
        file: "/uploads/video.mp4",
        height: 270,
        width: 480,
        skin: "/skins/modieus/modieus.zip"
    });
</script>
```

Note: If you're configuring the Embedder to run primarily in HTML5 mode using the *Players* block, you'll need to take the additional step of unzipping the skin ZIP and uploading its contents to your web server in the same location as the ZIP file itself. Your skin's folder structure would look something like this:

```
/skins/modieus/modieus.zip
/skins/modieus/modieus.xml
/skins/modieus/controlbar/
```

```
/skins/modieus/playlist/
etc.
```

1.4.4 Playlist

Previously, loading a playlist in the JW Player was only available by using an XML playlist format like RSS or ATOM. With the JW Player embed method though, it is possible to load a full playlist into the player using the **playlist** object block.

Here is an example. In it, a playlist of three items is loaded into the player. Each item contains a **duration** hint, the **file** location and the location of a poster **image**.

```
<div id="container">Loading the player...</div>

<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf",
        playlist: [
            { duration: 32, file: "/uploads/video.mp4", image: "/uploads/video.jpg" },
            { duration: 124, file: "/uploads/bbb.mp4", image: "/uploads/bbb.jpg" },
            { duration: 542, file: "/uploads/ed.mp4", image: "/uploads/ed.jpg" }
        ],
        "playlist.position": "right",
        "playlist.size": 360,
        height: 270,
        width: 720
    });
</script>
```

Note: The *playlist.position* and *playlist.size* options control the visible playlist inside the Flash player. To date, the HTML5 player doesn't support a visible playlist yet (though it can manage a playlist of videos).

A playlist can contain 1 to many videos. For each entry, the following properties are supported:

- **file**: this one is required (unless you have *levels*, see below). Without a video to play, the playlist item is skipped.
- **image**: location of the poster image. Is displayed before the video starts, after it finishes, and as part of the graphical playlist.
- **duration**: duration of the video, in seconds. The player uses this to display the duration in the controlbar, and in the graphical playlist.
- **start**: starting point inside the video. When a user plays this entry, the video won't start at the beginning, but at the offset you present here.
- **title**: title of the video, is displayed in the graphical playlist.
- **description**: description of the video, is displayed in the graphical playlist.
- **streamer**: streaming application to use for the video. This is only used for RTMP or HTTP streaming.
- **provider**: specific media playback API (e.g. *http*, *rtmp* or *youtube*) to use for playback of this playlist entry.
- **levels**: a nested object block, with multiple quality levels of the video. See the *levels* section for more info.

1.4.5 Levels

The **levels** object block allows you to load multiple quality levels of a video into the player. The multiple levels are used by the Flash player (HTML5 not yet) for RTMP or HTTP bitrate switching. Bitrate switching is a mechanism where the player automatically shows the best possible video quality to each viewer.

Here's an example setup, using RTMP bitrate switching (also called *dynamic streaming*). Note the additional *streamer* option, which tells the player the location of the RTMP server:

```
<div id="container">Loading the player...</div>

<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf",
            height: 270,
            width: 480,
            image: "/uploads/video.jpg",
            levels: [
                { bitrate: 300, file: "assets/bbb_300k.mp4", width: 320 },
                { bitrate: 600, file: "assets/bbb_600k.mp4", width: 480 },
                { bitrate: 900, file: "assets/bbb_900k.mp4", width: 720 }
            ],
            provider: "rtmp",
            streamer: "rtmp://mycdn.com/application/"
    });
</script>
```

Here is another example setup, this time using HTTP bitrate switching. The HTTP switching is enabled by setting the *provider* option to *http*:

```
<div id="container">Loading the player...</div>

<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf",
        height: 270,
        width: 480,
        image: "/uploads/video.jpg",
        levels: [
            { bitrate: 300, file: "http://mycdn.com/assets/bbb_300k.mp4", width: 320 },
            { bitrate: 600, file: "http://mycdn.com/assets/bbb_600k.mp4", width: 480 },
            { bitrate: 900, file: "http://mycdn.com/assets/bbb_900k.mp4", width: 720 }
        ],
        provider: "http",
        "http.startparam":"starttime"
    });
</script>
```

1.4.6 Plugins

Plugins can be used to stack functionality on top of the JW Player. A wide array of plugins is available in our library, for example for viral sharing, analytics or advertisements.

Here is an example setup using both the HD plugin and the Google Analytics Pro plugin:

```
<div id="container">Loading the player...</div>

<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf",
        file: "/uploads/video.mp4",
        height: 270,
        plugins: {
            hd: { file: "/uploads/video_high.mp4", fullscreen: true },
            gapro: { accountid: "UKsi93X940-24" }
        },
        image: "/uploads/video.jpg",
        width: 480
    });
    </script>
```

Here is another example, using the sharing plugin. In this example, plugin parameters are also included in the playlist block:

```
<div id="container">Loading the player...</div>

<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf",
        playlist: [
            { file: "/uploads/bbb.mp4", "sharing.link": "http://bigbuckbunny.org" },
            { file: "/uploads/ed.mp4", "sharing.link": "http://orange.blender.org" }
        ],
        plugins: {
            sharing: { link: true }
        },
        height: 270,
        width: 720
    });
</script>
```

1.4.7 Events

The **events** block allows you to respond on player events in javascript. It's a short, powerful way to add player - pager interactivity. Here is a swift example:

```
<div id="container">Loading the player ...</div>

<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf",
        file: "/uploads/video.mp4",
        height: 270,
        width: 480,
        events: {
            onComplete: function() { alert("the video is finished!"); }
        }
    });
</script>
```

Here is another example, with two event handlers. Note the *onReady()* handler autostarts the player using the *this* statement and the *onVolume()* handler is processing an event property:

```
<div id="container">Loading the player ...</div>

<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf",
        file: "/uploads/video.mp4",
        height: 270,
        width: 480,
        events: {
            onReady: function() { this.play(); },
            onVolume: function(event) { alert("the new volume is "+event.volume); }
        }
    });
</script>
```

See the *API reference* for a full overview of all events and their properties.

1.4.8 Players

The **players** option block can be used to customize the order in which the JW Player uses the different browser technologies for rendering the player. By default, the JW Player uses this order:

1. The **Flash** plugin.
2. The **HTML5** <video> tag.

Using the **players** block, it is possible to specify that the Embedder try the HTML5 player first:

```
<div id="container">Loading the player ...</div>

<script type="text/javascript">
    jwplayer("container").setup({
        file: "/uploads/video.mp4",
        height: 270,
        width: 480,
        players: [
            { type: "html5" },
            { type: "flash", src: "/jwplayer/player.swf" }
        ]
    });
</script>
```

1.4.9 Player Removal

In addition to setting up a player, the JW Player embed script contains a function to unload a player. It's very simple:

```
jwplayer("container").remove();
```

This formal **remove()** function will make sure the player stops its streams, the DOM is re-set to its original state and all event listeners are cleaned up.

CHAPTER

TWO

PLAYER API

The 5.3 player introduces a new, shorthand javascript API for interacting with your website. This API abstracts any differences between Flash and HTML5; any code you write will work with both technologies.

For versions 5.2 and below, the player used the 4.x JavaScript API. A reference for that API can be found on the player support site.

2.1 Getting started

First, you'll need to upload the API library (*jwplayer.js*) to your web server. We recommend putting it, along with *player.swf*, in a folder called **jwplayer** in the root of your site. Once it's on your web server, add this bit of code to your HTML pages, in the <*head*> of your page:

```
<head>
        <script type="text/javascript" src="/jwplayer/jwplayer.js"></script>
</head>
```

To get a sense of the possibilities of what you can do with the API, here's a quick example that showcases how to control the player from the page:

```
<div id="container">Loading the player ...</div>

<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf",
        file: "/uploads/video.mp4",
        height: 270,
        width: 480
    });
</script>

<ul>
    <li onclick="jwplayer().play()">Start the player</li>
    <li onclick="alert(jwplayer().getPosition())">Get current position</li>
</ul>
```

Of course it's also possible to have the player manipulate the page. Here's a second example, using the *event block* of the JW Player embedder:

```
<div id="container">Loading the player ...</div>
```

```
<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf",
        file: "/uploads/video.mp4",
        height: 270,
        width: 480,
        events: {
            onComplete: function() {
                document.getElementById("status").innerHTML("That's all folks!");
            }
        }
    });
</script>

<p id="status"></p>
```

The following sections give a detailed description of the JW Player API, describing how to:

- Select a player.
- Get variables from a player.
- Call functions on a player.
- Listen to events from a player.

2.1.1 Embedding with SWFObject

If you embed the player using SWFObject, rather than the built-in *setup()* function, you can still use the JavaScript API, although you'll need to wait for Flash to be loaded on the page before interacting with the API. SWFObject 2.2 includes a callback function (in this example, named **flashLoaded**) which is executed when SWFObject has finished embedding Flash into the page. Make sure you wait until this function is called before making any calls to the API.

Here's a simple example of using the SWFObject callback:

```
var flashvars = { file:"/videos/video.mp4" };
var params = { allowfullscreen:"true", allowscriptaccess:"always" };
var attributes = { id:"player", name:"player" };

swfobject.embedSWF("/jwplayer/player.swf", "container", 320, 240, "9.0.115", "false",
    flashvars, params, attributes, flashLoaded);

function flashLoaded(e) {
        // e.ref is a reference to the Flash object.  We'll pass it to jwplayer() so the API knows wh

    // Add event listeners
    jwplayer(e.ref).onReady(function() { alert("Player is ready"); });
    jwplayer(e.ref).onPlay(function() { alert("Player is playing"); });

    // Interact with the player
    jwplayer(e.ref).play();
}
```

2.1.2 Embedding with an <object> or <embed> tag

If you embed the player directly using an *<object>* or *<embed>* tag, simply pass your tag's id to the API when referencing the player:

```
<embed
  id="player"
  name="player"
  src="/jwplayer/player.swf"
  width="320"
  height="240"
  allowscriptaccess="always"
  allowfullscreen="true"
  flashvars="file=/videos/video.mp4"
/>

<script type="text/javascript">
    jwplayer("player").onReady(function() { alert("Player is ready"); });
    jwplayer("player").onPlay(function() { alert("Player is playing"); });
    jwplayer("player").play();
</script>
```

2.2 Selecting

The first thing you need to do when attempting to interact with a JW Player, is to get a reference to it. The easiest way, probably sufficient for 95% of all use cases is this:

```
// Start the player on this page
jwplayer().play();
```

Only when you have multiple players on a page, you need to be more specific on which player you want to interact with. In that case, there are three ways to select a player:

- With the *id* of the element you *instantiated* the player over:

  ```
  jwplayer("container").play();
  ```

- With the actual DOM element itself:

  ```
  var element = document.getElementById("container");
  jwplayer(element).play();
  ```

- With the index in the list of players on the page (in order of loading):

  ```
  jwplayer(2).play();
  ```

  Note: The selector *jwplayer(0)* is actually the same as *jwplayer()*.

2.3 Variables

Here is a list of all the variables that can be retrieved from the player:

getBuffer()
Returns the current PlaylistItem's filled buffer, as a **percentage** (0 to 100) of the total video's length.

getFullscreen()
Returns the player's current **fullscreen** state, as a boolean (*true* when fullscreen).

getMetadata()
Returns the current PlaylistItem's **metadata**, as a javascript object. This object contains arbitrary key:value parameters, depending upon the type of player, media file and streaming provider that is used. Common metadata keys are *width*, *duration* or *videoframerate*.

getMute()
Returns the player's current audio muting state, as a boolean (*true* when there's no sound).

getPlaylist()
Returns the player's entire **playlist**, as an array of PlaylistItem objects. Here's an example playlist, with three items:

```
[
    { duration: 32, file: "/uploads/video.mp4", image: "/uploads/video.jpg" },
    { title: "cool video", file: "/uploads/bbb.mp4" },
    { duration: 542, file: "/uploads/ed.mp4", start: 129 }
]
```

getPlaylistItem(*index*):
Returns the playlist **item** at the specified *index*. If the *index* is not specified, the currently playing playlistItem is returned. The **item** that is returned is an object with key:value properties (e.g. *file*, *duration* and *title*). Example:

```
{ duration: 32, file: "/uploads/video.mp4", image: "/uploads/video.jpg" }
```

getWidth()
Returns the player's current **width**, in pixels.

getHeight()
Returns the player's current **height**, in pixels.

getState()
Returns the player's current playback state. It can have the following values:

- **BUFFERING**: user pressed *play*, but sufficient data has to be loaded first (no movement).
- **PLAYING**: the video is playing (movement).
- **PAUSED**: user paused the video (no movement).
- **IDLE**: either the user stopped the video or the video has ended (no movement).

getPosition()
Returns the current playback **position** in seconds, as a number.

getDuration()
Returns the currently playing PlaylistItem's duration in seconds, as a number.

getVolume()
Returns the current playback volume percentage, as a number (0 to 100).

2.4 Functions

Here is a list of all functions that can be called on the player:

setFullscreen(state)
Change the player's fullscreen mode. Parameters:

- **state**:Boolean (*undefined*): If state is undefined, perform a fullscreen toggle. Otherwise, set the player's fullscreen mode to fullscreen if true, and return to normal screen mode if false.

setMute(state)
Change the player's mute state (no sound). Parameters:

- **state**:Boolean (undefined): If *state* is undefined, perform a muting toggle. Otherwise, mute the player if true, and unmute if false.

load(playlist)
Loads a new playlist into the player. The **playlist** parameter is required and can take a number of forms:

- *Array*: If an array of PlaylistItem objects is passed, load an entire playlist into the player. Example:

```
[
    { duration: 32, file: "/uploads/video.mp4", image: "/uploads/video.jpg" },
    { title: "cool video", file: "/uploads/bbb.mp4" },
    { duration: 542, file: "/uploads/ed.mp4", start: 129 }
]
```

- *Object*: If a PlaylistItem is passed, load it as a single item into the player. Example:

```
{ duration: 32, file: "/uploads/video.mp4", image: "/uploads/video.jpg" },
```

- *String*: Can be an XML playlist, or the link to a single media item (e.g. an MP4 video).

playlistItem(index)
Jumps to the playlist item at the specified index. Parameters:

- **index**:Number: zero-based index into the playlist array (i.e. playlistItem(0) jumps to the first item in the playlist).

playlistNext()
Jumps to the next playlist item. If the current playlist item is the last one, the player jumps to the first.

playlistPrev()
Jumps to the previous playlist item. If the current playlist item is the first one, the player jumps to the last.

resize(width, height)
Resizes the player to the specified dimensions. Parameters:

- **width**:Number: the new overall width of the player.
- **height**:Number: the new overall height of the player.

Note: If a controlbar or playlist is displayed next to the video, the actual video is of course smaller than the overall player.

play(state)
Toggles playback of the player. Parameters:

- **state**:Boolean (undefined): if set *true* the player will start playing. If set *false* the player will pause. If not set, the player will toggle playback.

pause(state)
Toggles playback of the player. Parameters:

- **state**:Boolean (undefined): if set *true* the player will pause playback. If set *false* the player will play. If not set, the player will toggle playback.

stop()
Stops the player and unloads the currently playing media file from memory.

seek(position)
Jump to the specified position within the currently playing item. Parameters:

- **position**:Number: Requested position in seconds.

setVolume(volume)
Sets the player's audio volume. Parameters:

- **volume**:Number: The new volume percentage; *0* and *100*.

2.5 Events

Here is a list of all events the player supports. In javascript, you can listen to events by assigning a function to it. Your function should take one argument (the event that is fired). Here is a code example, with some javascript that listens to changes in the volume:

```
jwplayer("container").onVolume(
    function(event) {
        alert("the new volume is: "+event.volume);
    }
);
```

Note that our *official embed method* contains a shortcut for assigning event listeners, directly in the embed code:

```
<div id="container">Loading the player ...</div>

<script type="text/javascript">
    jwplayer("container").setup({
        flashplayer: "/jwplayer/player.swf",
        file: "/uploads/video.mp4",
        height: 270,
        width: 480,
        events: {
            onVolume: function(event) {
                alert("the new volume is: "+event.volume);
            }
        }
    });
</script>
```

And here's the full event list:

onBufferChange(callback)
Fired when the currently playing item loads additional data into its buffer. Event attributes:

- **percent**: Number: Percentage (between 0 and 100); number of seconds buffered / duration in seconds.

onBufferFull(callback)
Fired when the player's buffer has exceeded the player's bufferlength property (default: 1 second). No attributes.

onError(callback)
Fired when an error has occurred in the player. Event attributes:

- **message**: String: The reason for the error.

onFullscreen(callback)
Fired when the player's fullscreen mode changes. Event attributes:

- fullscreen: boolean. New fullscreen state.

onMetadata(callback)
Fired when new metadata has been discovered in the player. Event attributes:

data: Object: dictionary object containing the new metadata.

onMute(callback)
Fired when the player has gone into or out of the mute state. Event attributes:

- **mute**: Boolean: New mute state.

onPlaylist(callback)
Fired when a new playlist has been loaded into the player. Event attributes:

- **playlist**: Array: The new playlist; an array of PlaylistItem objects.

onPlaylistItem(callback)
Fired when the player is playing a new media item. Event attributes:

- **index** Number: Zero-based index into the playlist array (e.g. 0 is the first item).

onReady(callback)
Fired when the player has initialized and is ready for playback. No attributes.

onResize(callback)
Fired when the player's dimensions have changed (the player is resizing or switching fullscreen). Event attributes:

- **width**: Number: The new width of the player.
- **height**: Number: The new height of the player.

onPlay(callback)
Fired when the player enters the *PLAYING* state. Event attributes:

- **oldstate**: String: the state the player moved from. Can be *PAUSED* or *BUFFERING*.

onPause(callback)
Fired when the player enters the PAUSED state. Event attributes:

- **oldstate**: String: the state the player moved from. Can be *PLAYING* or *BUFFERING*.

onBuffer(callback)
Fired when the player enters the BUFFERING state. Event attributes:

- **oldstate**: String: the state the player moved from. Can be *PLAYING*, *PAUSED* or *IDLE*.

onIdle(callback)
Fired when the player enters the IDLE state. Event attributes:

- **oldstate**: String: the state the player moved from. Can be *PLAYING*, *PAUSED* or *BUFFERING*.

onComplete(callback)
Fired when the player has finished playing the current media. No event attributes.

onPosition(callback)
When the player is playing, fired as the playback position gets updated. This happens with a resolution of 0.1 second, so there's a lot of events! Event attributes:

- **duration**: Number: Duration of the current item in seconds.
- **offset**: Number: When playing streaming media, this value contains the last unbuffered seek offset.

- **position**: Number: Playback position in seconds.

onVolume(callback)

Fired when the player's volume changes. Event attributes:

- **volume**: Number: The new volume percentage (0 to 100).

2.6 Chaining

Note that every API call to a JW Player in turn returns the player instance. This makes it possible to chain API calls (like with jQuery):

```
jwplayer().setVolume(50).onComplete(function(){ alert("done!"); }).play();
```