longtail VIDEO

JW Player

Release 5.3

www.longtailvideo.com

September 28, 2010

CONTENTS

CHAPTER

ONE

EMBEDDING THE PLAYER

As of version 5.3, the JW Player features its own embedding method. While external embed methods like SWFObject can still be used, the built-in embed method has a couple of awesome features:

- Complete integration of Flash and HTML5 player.
- Super easy to use *javascript API*.

1.1 Getting started

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

1. Upload the *jwplayer.js* and *jwplayer.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/jwplayer.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/jwplayer.swf"
    });
</script>
```

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

1.2 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. For a full overview of the <*video*> tag, see our *HTML5 Video Tag Reference*.

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/jwplayer.swf",
        skin: "/uploads/modieus.zip",
        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.

1.3 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/jwplayer.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 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/jwplayer.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/jwplayer.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>
```

1.5 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/jwplayer.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/jwplayer.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>
```

1.6 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/jwplayer.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/jwplayer.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.

1.7 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.
3. A plain **download** link.

Using the **players** block, it is possible to e.g. try to used the HTML5 player first:

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

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

Here is another variation. This time, the player source is also included in the *players* block, and the video download fallback is discarded:

```
<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/jwplayer.swf" }
        ]
    });
</script>
```

1.8 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.

2.1 Getting started

To get a sense of the possibilities, 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/jwplayer.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/jwplayer.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.

2.2 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/jwplayer.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();
```

API CHART

Here is a compact reference chart of the JW Player API. It includes all variables, functions and events the player supports:

Subject .	**Variable(s)**	**Function(s)**	**Event(s)**
Percentage loaded	getBuffer		onBufferChange onBufferFull
Media playback errors			onError
Fullscreen playback	getFullscreen	setFullscreen	onFullscreen
Media file metadata	getMeta		onMeta
Muting of the audio	getMute	setMute	onMute
The playlist	getPlaylist	load	onPlaylist
Current item	getPlaylistItem	playlistItem playlistPrev playlistNext	onPlaylistItem
Loading of the player			onReady
Resizing of the player	getHeight getWidth	resize	onResize
The playback state	getState	play pause stop	onPlay onPause onIdle onBuffer onComplete
The playback position	getPosition getDuration	seek	onTime
The audio volume	getVolume	setVolume	onVolume

CHAPTER

FOUR

HTML5 VIDEO TAG REFERENCE

This guide provides an overview of the functionalities and options provided by the HTML5 <video> tag. It includes a few examples, plus a list of known browser-specific quirks.

4.1 Introduction

Our HTML5 player is designed to gracefully enhance videos embedded with the HTML5 <*video*> tag. For a complete reference, visit the W3C HTML5 video element draft.

4.2 Example

Here is a basic example of a video embedded with the HTML5 video tag:

```
<video height="270" src="/static/bunny.mp4" width="480" controls>
  <a href="/static/bunny.mp4">Download this video</a>
</video>
```

This example will display the video in 480 by 270 pixels with a small controlbar over it. Browsers that do not support the video tag (nor the MP4 video) will render the enclosed download link instead.

4.3 Attributes

The HTML5 <*video*> tag has a short list of useful attributes:

`autoplay ()`
: Include this attribute (no value) to let the video start as the page gets loaded. Use sparsely, since this might get annoying.

`controls ()`
: Include this attribute (no value) to display a simple controlbar, provided by the user agent. The looks of this controlbar vary per browser.

`height (number)`
: Height of the video on the page, in pixels is required.&nbsp;</li>

`loop ():`
: Include this attribute to let the browser continously repeat playback of the video. At present, no browser honors this attribute. More info below, under *browser quirks*

poster (string)
The url of a poster frame that is shown before the video starts. Can be any PNG, JPG or GIF image. Is *undefined* by default.

preload (*auto*, *metadata*, *none*)
This attribute defines whether the video is entire loaded on page load, whether only the metadata is loaded on page load, or whether the video isn't loaded at all. At present, no browser honors this attribute. More info below, under *browser quirks*,

src (string)
The URL of the video to play. This is not required, since source files can also be listed using <*source*> tags (see below). It is *undefined* by default.

width (*number*)
Width of the video on the page, in pixels. Is required.

Here is another example of an HTML5 video tag, this time using more attributes:

```
<video
  controls
  height="270"
  loop
  poster="/static/bunny.jpg"
  src="/static/bunny.mp4"
  width="480">
  <a href="/static/bunny.mp4">Download the video</a>
</video>
```

This example will display the video with a poster frame and controls. After starting the video, it will repeat playback.

4.4 Multiple Sources

It is possible to provide a video tag with multiple source videos. The browser can then pick whichever file it can playback. This functionality works through the HTML5 <*source*> tag, which can be nested inside the <*video*> tag. A common use case is the provision of two videos, one in OGG for Firefox/Opera and one in MP4 for Chrome/Safari/iPhone/iPad:

```
<video height="270" width="480" controls>
  <source src="/static/bunny.mp4" type="video/mp4">
  <source src="/static/bunny.ogg" type="video/ogg">
  <a href="/static/bunny.mp4">Download the video</a> for local playback.
</video>
```

This example will play the MP4 video in Chrome/Safari and the OGG video in Firefox/Opera. Note that Chrome will play the OGG file if you reverse the order of the two <*source*> tags (Chrome supports both formats). We advise against this though: MP4 files render smoother and seek faster in Chrome than OGG files.

4.5 Browser Quirks

The W3C HTML5 video format is still in draft. Therefore, browser might have implemented older versions of this draft or filled the gaps with custom implementations. Especially around subjects like event broadcasting and (pseudo)streaming, things are still very buggy.

The biggest issue to be aware of - it's actually hard to have missed this in the press - is that there is no single video format supported by all browsers. Firefox and Opera support the *OGG* format, Internet Explorer (9), Safari and the iPhone/iPad support the *MP4* (H264) format. Chrome does both.