Introduction to HTML5 video

BY BRUCE LAWSON, PATRICK H. LAUKE · 11 FEB, 2010

Published in: CSS, CANVAS, ACCESSIBILITY, HTML5, MULTIMEDIA, VIDEO, OPERA 10.50

Introduction

A long time ago, in a galaxy that feels a very long way away, multimedia on the Web was limited to tinkling MIDI tunes and animated GIFs. As bandwidth got faster and compression technologies improved, MP3 music supplanted MIDI and real video began to gain ground. All sorts of proprietary players battled it out — Real Player, Windows Media Player, etc. — until one emerged as the victor in 2005: Adobe Flash. This was largely because of the ubiquity of its plugin and the fact that it was the delivery mechanism of choice for YouTube; Flash has become the de-facto standard for delivering video on the web.

One of the most exciting new features of HTML5 is the inclusion of the `<video>` element, which allows developers to include video directly in their pages without the need for any plugin-based solution. Opera proposed the standard `<video>` element (http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2007-February/009702.html) as part of HTML5 back in 2007, and it made its official Opera debut in the Opera 10.50 pre-alpha release.

This article gives you an introduction to `<video>` and some of its associated APIs. We look at why native video support in browsers is important, give an overview of the element's markup, and outline the most important ways in which video can be controlled via JavaScript.

Why do we need a <video> element?

Until now, if you wanted to include video in a web page, you had to wrangle some fairly cryptic markup. Here's an example, taken directly from YouTube:

```
<object width="425" height="344">
  <param name="movie"
    value="http://www.youtube.com/v/9sEI1AUFJKw&hl=en_GB&fs=1&"></param>
  <param name="allowFullScreen"
    value="true"></param>
  <param name="allowscriptaccess"
    value="always"></param>
  <embed src="http://www.youtube.com/v/9sEI1AUFJKw&hl=en_GB&fs=1&"
    type="application/x-shockwave-flash" allowscriptaccess="always"
    allowfullscreen="true" width="425"
```

```
    height="344"></embed>
</object>
```


First of all, we have an `<object>` element — a generic container for "foreign objects" — to include the Flash movie in. To work around browser inconsistencies, we also include an `<embed>` element as fallback content and duplicate most of the `<object>`'s parameters. The resulting code is ungainly and not very readable.

Anatomy of the `<video>` element

Compared to the convoluted construct necessary to include Flash in your page, the basic markup necessary for `<video>` in HTML5 is refreshingly straightforward:

```
<video src=turkish.ogv
  width=320
  height=240
  controls
  poster=turkish.jpg>
</video>
```

> Note that in our example we're taking advantage of HTML5's more relaxed syntax — you don't have to put quotes around attribute values, and you can use simple boolean attributes such as `autoplay` as single words. If you prefer, you can of course also stick to the XHTML syntax `controls="controls"` and quote all your attribute values. It obviously makes sense to choose the coding style that suits you best and stick with it, for consistency and maintainability. In XHTML5, you **must** use XHTML syntax (and you must also serve your pages as XML with the correct MIME type, which currently won't work in Internet Explorer).

The `<video>` element attributes we've used in our sample code are quite self-explanatory:

`src`

> The source of the element, providing the URL of your video file.

`width` and `height`

> As with `img` elements, you can explicitly set the dimensions of your video — otherwise, the element will simply default to the intrinsic width and height of the video file itself. If you specify one but not the other, the browser will adjust the size of the unspecified dimension to preserve the video's aspect ratio.

`controls`

> If this boolean attribute is present, the browser will display its native video controls for playback and volume. If you omit this, the user will only see the first frame (or the specified `poster` image) and won't be able to play the video, unless you trigger the movie from somewhere in your JavaScript or create your own custom controls, as we'll demonstrate later in this article.

`poster`

> The `poster` attribute points to an image that the browser will use while the video is downloading, or until the user tells the video to play. If this is not included, the first frame of the video will be used instead.

For Web browsers that do not currently support `<video>`, it's possible to include alternative content — at the very least, this could include some text and a link to the video file itself, so that users can download it and play it in a media player application:

```
<video src=turkish.ogv
  width=320
  height=240
```

```
  controls
  poster=turkish.jpg>
  Download the <a href=video.ogg>Turkish dancing masterclass video</a>
</video>
```

So, without further ado, check out this video of a Turkish dancing masterclass (http://people.opera.com/patrickl/articles/introduction-html5-video/basic/), implemented using nothing but the power of the native `<video>` element.

There are more attributes we're not covering in our examples. They are:

`autoplay`

> This instructs the browser to start playback of the video automatically. Use this attribute with care, as this can be highly annoying, if not downright problematic, particularly for users with assistive technologies such as screen readers or those on low-bandwidth connections (such as on a mobile device).

`autobuffer`

> If you're pretty sure that the user will want to activate the video (for example they've navigated specifically to it and it's the only reason to be on the page) but you don't want to use `autoplay`, you can set the `autobuffer` boolean. This asks the browser to begin downloading the media immediately, anticipating that the user will play the video. (This part of the specification is currently in flux and subject to change; it is therefore not implemented in Opera 10.5 beta)

`loop`

> The `loop` attribute is another boolean attribute. As you would imagine, it loops the video. (Currently this is not implemented in Opera 10.50 beta)

Codecs — the fly in the ointment

Opera supports the Ogg Theora video codec (http://theora.org/):

> Theora is a free and open video compression format from the Xiph.org Foundation...it can be used to distribute film and video online and on disc without the licensing and royalty fees or vendor lock-in associated with other formats.

Firefox and Chrome also support Theora. However, Safari doesn't, preferring instead to provide native support for the H.264 codec (http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC) (which Chrome also supports). Therefore, we need to encode our videos twice, once as Theora and once as H.264, add alternative `<source>` elements with appropriate `type` attributes to the video and let the browser download the format that it can display. Note that in this case we do not provide a `src` attribute in the `<video>` element itself:

```
<video width=320 height=240 controls poster=turkish.jpg>
  <source src=turkish.ogv type=video/ogg>
  <source src=turkish.mp4 type=video/mp4>
  Download the <a href=video.ogg>Turkish dancing masterclass video</a>
</video>
```

Here's our Turkish dancing masterclass with both `.ogv` and `.mp4` (http://people.opera.com/patrickl/articles/introduction-html5-video/multi-source/) sources, which should now work in browsers from either side of the current codec war.

At the time of writing (February 2010), Microsoft representatives have given no public announcement of which video codec they will support (if any). However, the MP4 filetype can also be played by the Flash player plugin, so these can be used in combination as a fallback for Internet Explorer and older versions of other browsers. See Kroc

Camen's nifty implementation of this technique in his article *Video for Everybody!* (http://camendesign.com/code/video_for_everybody), in which he "frankensteins" the `object` and `embed` of old into the alternative content part of the `<video>` element.

> Of course, if the browsers that don't support the `<video>` element fall back to using Quicktime or Flash plugins, we're really back where we started, and we won't be able to take advantage of any of the new features and improvements we're about to outline below. *"What's the point then?"*, you may ask. We would say that this is a transitional solution, until native video support hits all major browsers. It's a case of graceful degradation – users may receive a slightly cut-down version of your page, but at least they're able to see your movies.

No longer a second-class citizen on the Web

So, we've seen that the markup for the new HTML5 `<video>` element is an order of magnitude more readable and understandable compared to what we currently have to do in order to get Flash movies into our markup. But regardless of how horrid the old way of coding is, in most cases it works, doesn't it? So why would we want to move away from this approach of handing over the display of video to a third-party plugin such as Flash?

The major advantage of the HTML5 `<video>` element is that finally video is a fully-fledged citizen on the Web, rather than being shunted off to the hinterland of `object` or the non-validating `embed` element (although that now validates in HTML5).

Keyboard accessibility out-of-the-box

One of the great unresolved problems of using Flash is keyboard accessibility. With the exception of Internet Explorer on Windows, there is usually no way for a keyboard user to `Tab` or otherwise move their focus into a Flash movie. And even if these users somehow manage to get their focus into the movie (using additional assistive technologies), there is no easy way for them to `Tab` back out of it (unless additional ActionScript code is added to the movie to programmatically set the browser focus back out of the plugin and onto the page). In contrast, as the browser is directly responsible for the `<video>` element, it can handle the movie's controls as if they were regular buttons on a web page and include them in its normal tabbing order.

> Keyboard support for native video has not currently been implemented across all browsers...however, it already works fine in Opera 10.50 beta.

`<video>` plays well with the rest of the page

In simple terms, whenever you include a plugin in your pages, you're reserving a certain drawing area that the browser delegates to the plugin. As far as the browser is concerned, the plugin's area remains a black box — the browser does not process or interpret anything that is happening there.

Normally, this is not a problem, but issues can arise when your layout overlaps the plugin's drawing area. Imagine for instance a site that contains a Flash movie, but also has JavaScript or CSS-based dropdown menus that need to unfold over the movie. By default, the plugin's drawing area sits on top of the webpage, meaning that these menus will strangely appear behind the movie. A similar unsightly effect happens in cases where your page uses lightboxes — any Flash movie would still appear to float on top of the dimmed page overlay. This is why most ready-made lightbox scripts often hack around the issue by first removing any plugin objects from the page before triggering the overlay itself, and reintroducing them once the overlay is closed.

> In the specific case of Flash plugins, developers can fix this display issue by adding the `wmode='opaque'` attribute to their `<object>` element and the equivalent `<param name='wmode' value='opaque'>` to their `<embed>` element. However, this also causes the plugin to become completely inaccessible to users with screen readers, and is therefore best avoided.

Problems and quirks can also arise if your page has dynamic layout changes. If the dimensions of the plugin's drawing area are resized, this can sometimes have unforeseen effects — a movie playing in the plugin may not resize, but instead simply be cropped or display extra white space.

With the native <video> element, it's the browser itself that is taking care of the rendering. As such, <video> behaves no differently from any other element in your page layout. It can be positioned, floated, overlapped or dynamically resized, with no additional hacks required. It is even possible to achieve interesting effects such as semi-transparent video simply by setting the opacity for the element via CSS.

A whole new world of cute kitten videos awaits us. Now, I don't have any kittens but I do have the next best thing — children — so I'll use a couple of videos of them for demonstration purposes.

Styling video with CSS

Now video is part of the Open Web set of technologies, we can use CSS to style the element reliably. As a simple demonstration of what can now be achieved, we'll apply CSS transitions to the Turkish dancing video (http://people.opera.com/patrickl/articles/introduction-html5-video/transitions/) to change its dimensions once we `:hover` or `:focus` on it. (Read our *CSS3 transitions and 2D transforms* tutorial (http://dev.opera.com/articles/view/css3-transitions-and-2d-transforms/).)

Combining video and canvas

As the browser is taking care of laying out and rendering video, we can easily overlap and combine other elements on top of it. In this example, a <canvas> is superimposed over the video (http://people.opera.com/patrickl/articles/introduction-html5-video/video-canvas/). (Warning: this video has potentially upsetting images of a handsome Opera employee and his children being menaced by a gigantic mouse pointer.)

Note that the <canvas> does not completely cover the video. We've made the canvas 40 pixels shorter than the video height, so that the area of the video where the controls appear is not covered. This ensures that if the user mouses over the bottom of the video, there is enough of the <video> element poking out behind the canvas to receive the `hover` event and cause it to expose the controls. Keyboard access to the controls should work regardless of covering elements, however keyboard support currently varies across browsers.

Scripting your own controls

<video> and <audio> (which we will cover in a future article) are instances of the new HTML5 DOM media elements (http://www.w3.org/TR/html5/video.html#media-elements), which exposes a powerful API giving developers control over movie playback through a whole host of new JavaScript methods and properties. Let's have a look at some of the most relevant ones to build ourselves a simple custom controller:

`play()` and `pause()`

Quite obviously, these methods start and pause video playback. `play()` will always start the video from the current playback position. When a movie is first loaded, this will be the first frame of the movie. Note that there is no `stop()` method — if you want to stop playback and "rewind" to the start of the movie, you will have to `pause()` and programmatically change the current playback position yourself.

`volume`

This attribute can be used to get or set the volume of the video's audio track as a `float`, ranging from *0.0* (silent) to *1.0* (loudest).

`muted`

Regardless of `volume`, a video can be muted.

`currentTime`

When read, this attribute returns the current playback position in seconds, again expressed as a `float`. Setting this attribute will — if possible — move the playback position to the specified time index. (Note that setting `currentTime`, or seeking, is not currently implemented in the Opera 10.50 beta)

In addition, media elements also fire a range of events as part of their processing model, which we can listen for and hook into. For our example, we will only look at a few of these:

`loadeddata`

The browser has loaded enough video data to begin playback at the current position.

`play` and `pause`

Playback was started or paused. If we're controlling the video from JavaScript, we want to listen out for these to ensure that calling the `play()` and `pause()` methods actually return succesfully.

`timeupdate`

The current playback position has changed because the movie is playing, a script changed it programmatically, or the user has decided to jump to a different point in the video.

`ended`

We've reached the end of the movie, and the `<video>` element is not set to `loop` or play backwards (not covered in this article).

Now we have all the basic building blocks needed to create a simple controller. Before we start though, a word of warning: if we're building our own JavaScript-based controller, we obviously want to suppress any native browser controls. However, we may wish to provide those controls as a fallback, in case users have turned off JavaScript in their browser. For this reason, we will retain the `controls` attribute in our markup, and programmatically remove it at runtime using our script. Alternatively, we could also set the value of the attribute to *false* — both approaches are valid. As our custom controller itself relies on scripting to function, we'll generate the markup of the controller itself in JavaScript.

See our example for scripting your own HTML5 video controls (http://people.opera.com/patrickl/articles/introduction-html5-video/scripted-controls/) in action. The script is verbose, and would benefit from a bit of a clean-up before being used in a production environment, but hopefully it helps to illustrate some of the new powerful possibilities that HTML5 video opens up. With a bit of JavaScript knowledge, it's now easy for web developers to create custom video controls that perfectly complement their site designs, without the need to create bespoke Flash video players.

Read more

- `<video>` specification (http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#video)
- How `<video>` is implemented in Opera (http://my.opera.com/core/blog/2009/12/31/re-introducing-video)
- Accessible HTML5 Video with JavaScripted captions (http://dev.opera.com/articles/view/accessible-html5-video-with-javascripted-captions/)

del.icio.us (http://del.icio.us/post?url=/articles/view/introduction-html5-video/?lookup=introduction-html5-video&title=)

digg (http://digg.com/submit?url=/articles/view/introduction-html5-video/?lookup=introduction-html5-video&title=&media=news&topic=people&thumbnails=0)

reddit (http://reddit.com/submit?url=/articles/view/introduction-html5-video/?lookup=introduction-html5-video&title=)

Technorati (http://www.technorati.com/faves?add=/articles/view/introduction-html5-video/?lookup=introduction-html5-video)

Discuss this article