HOUR 12

# Audio and Video in HTML5

### *What You'll Learn in This Hour:*

- Why the <video> and <audio> tags are important
- What support is available for the new multimedia elements
- How to add video and audio into your HTML that works
- How to force older browsers to display your multimedia
- How to create custom controls for your video and audio files

One of the most well-known of the new HTML5 features is the <video> element, but there is a new <audio> element as well.

In this hour you will learn how to use both elements to place video and audio in your web pages in a similar way to how you put images in. You will also start to understand the reason for these tags and how they relate to the other ways already available for embedding audio and video in web pages.

You will also learn how to add video and audio that is backward compatible for browsers that don't support HTML5, how to improve your multimedia playback, and create custom controls.

## Why Use HTML5 for Audio and Video vs. Flash

The HTML5 standard created the <audio> and <video> elements to allow web designers a standardized way to place video and audio files directly in their web pages. One way to think of it is as though audio and video files are the same as image files. You can embed an image in your HTML with the <img> element, and now you can embed sound and video with the <audio> and <video> elements. In previous versions of HTML, you had to embed your video and audio files and hope

that the browsers could open them correctly, but these HTML5 elements make embedding video and audio much easier.

A lot of discussion is going on about whether HTML5 video will "beat" Flash, and people are offering many reasons why it should (or should not). Some of the pros of using HTML5 (instead of Flash) for media elements include:

- Adding media with these tags is more semantic. A user agent knows immediately that an `<audio>` file is sound. It has to do more analysis to determine what a sound file inside an `<embed>` or `<object>` element is.
- You are not limited to one vendor to play your movies and sound.
- Your users do not have to have a plug-in to play your movies and sound.
- Because these elements are part of the DOM, you have more control over them through scripts and CSS.
- iOS devices do not have Flash.
- Adding sound and video to your web pages is a lot easier.

But using HTML5 also has some drawbacks, mostly around browser support:

- As with most other HTML5 features, Internet Explorer 8 and lower do not support the HTML5 `<video>` and `<audio>` elements.
- You have to encode video in at least two or three different formats.

Many people who already do all their video in Flash see no point in switching, but Flash has some serious drawbacks of its own, not least of which is that it causes the majority of crashes on Linux and Macintosh machines.

The problem with HTML5 video isn't really HTML5 at all; it's the codecs—the way videos are encoded for viewing. User agents currently support three formats:

- MP4 or H.264
- ogg/Theora
- WebM

Chrome and Android support all three. Firefox and Opera support ogg/Theora and WebM. IE 9 supports MP4 and WebM with an add-on. Safari and iOS support MP4.

Still, if you are serious about creating web applications with video and audio for mobile devices, then you can't use Flash only. Flash is fine as a fallback option for desktop users, but for mobile you should encode in H.264 and WebM, at a minimum.

# Choosing Video Formats for the Best Compatibility

As mentioned in the previous section, you should use the three primary video codecs (plus Flash for fallback purposes): MP4 or H.264, ogg/Theora, and WebM. Table 12.1 shows the current state of browser support for the video codecs.

**TABLE 12.1** Browser Support for Video Codecs

| Browser | MP4/H.264 | ogg/Theora | WebM |
|---|---|---|---|
| Android | version 3.0 | version 2.3 | version 2.3 |
| Chrome | version 9 (Windows)<br>version 7 (Macintosh) | version 9 (Windows)<br>version 7 (Macintosh) | version 9 (Windows)<br>version 7 (Macintosh) |
| Firefox | | version 3.6 | version 3.6 |
| Internet Explorer | version 9 | | version 9 (with components) |
| iOS | version 3 | | |
| Opera | | version 10.63 | version 10.63 |
| Safari | version 5 | | |

**Watch Out!**

**Check Your Server MIME Types**

If you are having a problem getting your web server to display videos, the MIME types might not be set up correctly. If you have access to the Apache httpd.conf file, add the following three lines and then reboot your server:

```
AddType video/mp4 .mp4
AddType video/ogg .ogv
AddType video/webm .webm
```

Contact your hosting provider if you can't add those lines yourself.

## MP4 or H.264

H.264 is the codec that is supported primarily by Microsoft and Apple. H.264 files have the `.mp4` extension. With it, you can create good quality videos that are small in size. Plus a lot of programs are available that will encode H.264 video. In fact, Flash uses H.264 video.

The problem is that the H.264 codec is not free. It is a licensed product owned by MPEG LA. They have stated that they won't charge royalties for internet video that uses their codec, as long as that video is free to end users. But the Mozilla group

(makers of Firefox) have stated that Firefox will never support H.264 video because of patent issues and their focus on open source software.

However, if you want your videos to play on iPhones and iPads, you need to use this format, because iOS doesn't support any other format.

The benefits (beyond iOS support) to this format are that it creates small, high-quality videos, and a lot of programs are available that will create videos in this format.

### ogg/Theora

Ogg/Theora is an open standard for video encoding that Firefox does support. Ogg/Theora files use the `.ogg` file extension. It generates high-quality videos, but they are often much larger than the same video in H.264 format. ogg/Theora videos can be 40%–50% larger than the same video in H.264.

In addition to large file sizes, the other problem with ogg/Theora is that finding a good program to encode in this format can be very difficult. Professional-level video programs such as Premiere and Final Cut Pro don't support it without a plug-in.

The benefit to ogg/Theora is that it is open source and free to use.

### WebM

WebM uses the VP8 codec owned by Google. WebM files use the `.webm` extension. It has very good compression and creates high-quality videos. Google claims that the quality is better than H.264, but independent tests have shown it to be about the same or only slightly better (according to Streaming Media magazine www.streamingmedia.com/Articles/Editorial/Featured-Articles/WebM-vs.-H.264-A-Closer-Look-68594.aspx).

Like ogg/Theora, WebM can be hard to encode because programs such as Premiere and Final Cut Pro don't support it without a plug-in.

However, also like ogg/Theora, WebM is open source and free to use.

## Choosing Audio Codecs for the Widest Support

As with video, you can use several codecs for creating audio for web pages: MP3, Vorbis, and WAV. Table 12.2 shows the current state of browser support for these audio codecs.

**TABLE 12.2** Browser Support for Audio Codecs

| Browsers | MP3 | Vorbis | WAV |
|---|---|---|---|
| Android | version 2.3 | version 2.3 | |
| Chrome | version 6 | version 6 | |
| Firefox | | version 3.6 | version 3.6 |
| Internet Explorer | version 9 | | version 9 |
| iOS | version 3 | | |
| Opera | | version 10.5 | version 10.5 |
| Safari | version 5 | | version 5 |

## MP3

MP3 is a popular sound format because of the high compression rates and high quality. These files have the `.mp3` extension. A lot of encoders are also available to create this format for your audio files.

However, because patents are associated with this format, it will not be supported by Mozilla browsers such as Firefox until the patents run out in 2017 or later.

## Vorbis

Vorbis, also known as Ogg Vorbis, is an open source audio codec. These files have the `.ogg` extension. It generates high-quality sound files with much smaller size than other audio codecs.

Finding commercial programs to convert files to Vorbis can be difficult, but many online converters are available that will do it for you.

## WAV

WAV is a codec that was designed by Microsoft and IBM. WAV files have the `.wav` extension. It is widely supported, and often users with a browser that won't play a WAV file inline can play it if they download it to their computer.

WAV files can be compressed, but typically are not, and so tend to be a lot larger (many times the size) than the MP3 and Vorbis files of the same audio. It does, however, create high-quality recordings. Most developers only use WAV files for sound effects.

# The New HTML5 Media Elements

HTML5 has several new elements you can use to add audio and video to your web pages:

- **<video>**—Use this for a video stream.
- **<audio>**—Use this for an audio stream.
- **<source>**—This is the media source(s) for <audio> and <video> elements.
- **<track>**—Use this to create supplementary media tracks, such as subtitles and captions for <audio> and <video> elements.

Here is how to add an audio stream to your HTML:

```
<audio>
    <source src="sound.mp3">
</audio>
```

You add a video stream in the same way:

```
<video>
    <source src="movie.webm">
</video>
```

The nice thing about the <audio> and <video> elements is that you are not limited to one <source>. So, even though Firefox doesn't support MP3 and iOS doesn't support WebM, you can write your HTML so that it doesn't matter, by adding a second <source> tag to your media referencing a different source file in another format. You can add as many <source> elements as you have media files to reference:

```
<video>
    <source src="movie.mp4">
    <source src="movie.ogv">
    <source src="movie.webm">
</video>
```

***Watch Out!***

**iOS Issues with HTML5 Video**

iOS 3.2 won't recognize video if you use a `poster` attribute. The `poster` attribute points to a URL of an image that represents the video when it isn't playing. If you have a lot of iOS 3.2 users, you should avoid this attribute. iPads running iOS 3.2 won't notice anything but the first video source listed, so if you don't list MP4 first, the video won't play. These bugs were fixed in version 4.0, but anyone who hasn't upgraded will have these problems.

You can then include the <track> element if you have captions or subtitles for your media:

```
<audio>
    <source src="sound.mp3">
    <source src="sound.ogg">
    <source src="sound.wav">
    <track kind="captions" src="captions.srt" srclang="en">
</audio>
```

**There Is No Standard for Timed Track Formats**

*Did you Know?*

Captions, subtitles, and video descriptions are all information that can be stored in the `<track>` element. But as of this writing, no standard format for time-based data exists. Some of the formats under consideration are SRT, WebVTT, and many others. Until this format is finalized (possibly by browser implementation), adding captions or other timed track media to your media files will be difficult.

Finally, you can include fallback text and links for browsers that don't support the media element. Here is an example of using fallback content for browsers that don't support the `<video>` element:

```
<video>
    <source src="movie.mp4">
    <source src="movie.ogv">
    <source src="movie.webm">
    <track kind="subtitles" src="subtitles.srt"></track>
    <p>If your browser doesn't support video playback, download the video: <a
href="movie.mp4">MP4</a>, <a href="movie.ogv">ogg/Theora</a>, <a
href="movie.webm">WebM</a>
</video>
```

One attribute you should consider using every time you use a media tag is `controls`. This attribute provides your users with controls that they can use to start, stop, fast-forward, and rewind through your video and audio files, as well as turn off the volume:

```
<audio controls>
```

**Android Issues with HTML5 Video**

*Watch Out!*

Versions of Android before 2.3 don't like seeing a `type` attribute on the `<source>` element. To solve this issue, always make sure your H.264 videos end with a `.mp4` extension. Also, versions 2.2 and lower of Android do not support the `controls` attribute. You need to include your own interface controls if you have a lot of users who use 2.2 and lower. These bugs have been fixed in 2.3.

## Try It Yourself

### Adding a Music File to Your HTML

Adding music to your web pages is easy with the `<audio>` element. In this exercise you will save your music as MP3, WAV, and Vorbis files so that it has good browser support.

1. Record your audio file and save it as an MP3 file.
2. Convert it to WAV and Vorbis.
3. Upload all three audio files to your web server.
4. Add an `<audio>` element to your HTML:

```
<audio controls>
```

5. Include three `<audio>` elements pointing to each file:

```
<source src="Slap Happy.mp3">
<source src="Slap Happy.ogg">
<source src="Slap Happy.wav">
```

6. Add fallback text for browsers that don't support `<audio>`:

```
<p>Your browser does not support audio playback, download the file:
<a href="Slap Happy.mp3">MP3</a>,
<a href="Slap Happy.ogg">Vorbis</a>,
<a href="Slap Happy.wav">WAV</a>
```

7. Add a Modernizr script to only show that text to non-compliant browsers:

```
<script>
if (!Modernizr.audio) {
    document.write('<p>Your browser does not support audio playback,
download the file <a href="Slap Happy.mp3">MP3</a>,
<a href="Slap Happy.ogg">Vorbis</a>,
<a href="Slap Happy.wav">WAV</a>');
}
</script>
```

8. Close the `<audio>` element:

```
</audio>
```

Don't forget to test in as many browsers as you can.

# Useful Attributes to Extend Your Media

So far, the only attributes you've used are the `controls` attribute on both `<video>` and `<audio>` and the `src` attribute on the `<source>` element. However, you can use a number of other attributes to control even more about your videos.

## Audio and Video Attributes

Several attributes can be used on both `<audio>` and `<video>` elements:

- **`autoplay`**—This tells the browser to start playing the song or video as soon as it has streamed enough to play without stopping. This is an attribute without any values, but if you're writing XHTML, you would write `autoplay="autoplay"`.

**Watch Out!**

**Autoplay Will Drive Away Your Users**

One of the fastest ways to get your users to leave your page without reading or buying anything is to have music or video automatically start when they come to your page. Remember that they may have music already on or they may be in a workplace environment where sounds are inappropriate. Use the `autoplay` attribute with extreme caution.

- **`preload`**—The `preload` attribute lets you give the user agent hints as to what type of preloading it should attempt to do. Values are `none`, `metadata`, and `auto`. The `none` attribute tells the browser that it shouldn't attempt to preload; `metadata` tells it to get the duration, track list, dimensions, and other meta information; and `auto` suggests that it should try to preload the whole stream. Leaving off the value is the same as writing `preload=auto`.
- **`controls`**—Provides a user interface for controlling the stream. This attribute has no values.
- **`loop`**—The `loop` attribute tells the browser to restart the stream when it gets to the end. This attribute has no values.
- **`mediagroup`**—This tells the user agent to link several streams (audio or video or both) together. For example, a video might have a separate audio track and a sign-language interpretation. These could all be linked by giving them the same `mediagroup` identifier.
- **`src`**—This is an alternative to the `<source>` element if your stream has only one source.

## Video

The `<video>` element also has a few other attributes that you can use to improve your videos:

- **`poster`**—This is a URL of an image to be shown while no video is available. This option is a great way to provide a preview of your videos to entice people to watch them. Otherwise, they will see just a blank or black screen.

- **height** and **width**—Set the height and width of the video in CSS pixels. You can resize your videos using these controls, but you won't be able to change the aspect ratio.
- **muted**—This allows you to set the audio to mute as the default state. This attribute was only recently added to the specification, and many browsers don't support it.

## Source Attributes

The `<source>` element has three attributes: `src`, `media`, and `type`. The `src` attribute is self-explanatory—it is a URL pointing to a source file. The `media` and `type` attributes are a little trickier.

The `media` attribute takes a media query list and uses it to help the user agent determine whether the media will be useful to the user. You use media queries (discussed in Hour 4, "Detecting Mobile Devices and HTML5 Support") to define a comma-separated list of media that the video can play on. For example:

```
media="screen, 3d-glasses, resolution > 900dpi"
```

This tells the user agent that this video is viewable on "screen," with "3d-glasses," and on devices with a "resolution greater than 900dpi." (Note that 3d-glasses is not a supported type, yet. If you use this, most user agents will ignore it.) Just like with CSS media queries, you can set up complex scenarios for when the source file should be used, such as:

```
media="screen and (aspect-ratio: 16/9) and (scan: progressive)"
```

*Did you Know?*

**A Specific Media Query for Retina Display**

If you want to send a higher quality source file to iPhones with Retina display (the higher resolution display found on newer model iPhones), you can use a special `-webkit` media query: `-webkit-min-device-pixel-ratio: 2`.

As usual, be sure to test your media queries as much as possible, because they are not widely supported in this context.

The `type` attribute provides the browser more information about the type of the video and is the trickiest for the `<video>` element. Although it's not required, using it is a good idea especially if you are using video types other than MP4, ogg/Theora, or WebM. But even for these, you should consider using the `type` attribute on your source files. If the browser doesn't think it can load a video file based on the type you list, it won't download the file, which will make your whole page faster.

The `type` attribute has the following format:

```
type='MIME type; codecs="video codec, audio codec"'
```

The MIME types for MP4, Ogg, and WebM are

- **H.264 or MP4**—`video/mp4`
- **ogg/Theora**—`application/ogg`, `video/ogg`
- **WebM**—`video/webm`

Here is how to write the `<source>` elements for a video in three formats:

```
<source src="video.mp4" type='video/mp4; codecs="vc1.42E01E, mp4a.40.2"'>
<source src="video.ogv" type='video/ogg; codecs="theora, vorbis"'>
<source src="video.webm" type='video/webm; codecs="vp8, vorbis"'>
```

Writing the MP4 HTML is the most challenging, because you need to know the correct video codec and the AVC level. For more information, check out the Video type parameters page on the WHATWG Wiki (http://wiki.whatwg.org/wiki/Video_type_parameters).

## Track Attributes

The `<track>` element uses attributes to determine the source of the track file, the type of track file it is, and more:

- **`src`**—The URL to the timed track file.
- **`srclang`**—The language of the timed track.
- **`label`**—A user-readable title for the track.
- **`default`**—An attribute that defines this track as the one that should be enabled, if user preferences do not specify a different one. This attribute has no values.
- **`kind`**—The five different types of timed track are as follow: `subtitles`, which provide a translation of the dialog; `captions`, which provide a transcription of the video sounds and dialog, including sound effects and musical cues; `descriptions`, which are a textual description of the visual components of a video; `chapters` for chapter titles; and `metadata` for meta content used at scripted points in the video.

# Creating Fallback Options for Internet Explorer

The number one reason most developers are resistant to using the new <video> and <audio> elements is because IE 8 and lower don't support these elements. To get your sound and movies to play in these browsers you have to create some type of fallback.

By including an <object> element inside your <video> or <audio> element, you provide a fallback for browsers that don't support these elements, because they ignore the elements they don't recognize and instead just play the object files. Browsers that support the <video> and <audio> elements will find a source they can play, and ignore the <object>. If you still have problems, you can wrap the <object> in a Modernizr script to check for HTML5 support.

For video, you can use any Flash video player you like. Flowplayer (http://flowplayer.org/) is a good option because it is easy to use and only requires three additional files. For audio, you can use the WAV file you already created; just embed it in an <object> element like this (change FILE.wav to your filename):

```
<object>
    <param name="autostart" value="false">
    <param name="src" value="FILE.wav">
    <param name="autoplay" value="false">
    <param name="controller" value="true">
    <embed src="FILE.wav" controller="true" autoplay="false"
        autostart="false" type="audio/wav">
</object>
```

## ▼ Try It Yourself

### Adding a Video File with Flash Fallback

Adding video to your web pages is easy, and in this Try It Yourself, you will learn how to include a Flash fallback option for browsers that don't support HTML5 video.

1. Create your video and convert it to four formats: MP4, ogg/Theora, WebM, and FLV.
2. Save them to your web directory and upload them all to your web server.
3. Add the HTML5 video to your page:

   ```
   <video controls height="600" width="800">
   ```

4. Add source files for the MP4, Ogg, and WebM—make sure that MP4 is first:

```
<source src="Shasta.mp4">
<source src="Shasta.theora.ogv">
<source src="Shasta.webm">
```

5. Install Flowplayer on your website (go to http://flowplayer.org/download/index.html to download).

6. Point your Flowplayer text to the FLV file you created:

```
<!— fallback to Flowplayer: —>
<a
    href="Shasta.flv"
    style="display:block;width:800px;height:600px"
    id="player">
</a>
<script>
    flowplayer("player", "flowplayer/flowplayer-3.2.7.swf");
</script>
```

7. Don't forget to add the Flowplayer script to the head of your document:

```
<script src="flowplayer/flowplayer-3.2.6.min.js"></script>
```

8. Add text links to your video so that even older browsers can still download the file:

```
<script>
if (!Modernizr.video) {
    document.write('<p>Your browser does not support video playback,
download the video:
<a href="Shasta.mp4">MP4</a>,
➥<a href="Shasta.theora.ogv">ogg/Theora</a>,
➥<a href="Shasta.webm">WebM</a>');
}
</script>
```

9. Close the video element:

```
</video>
```

You can see this video in action at www.html5in24hours.com/examples/media-examples.html.

▲

# Creating Custom Controls with API Methods

After you start testing your videos you will notice that the default controls look different on every browser and device. Figure 12.1 shows you the controls for the same

video in four different browsers. As you can see, the controls look different in each and can be jarring if you are trying for a uniform look across browsers.

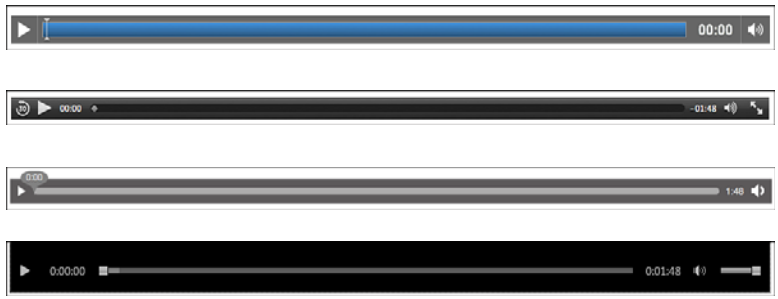


**FIGURE 12.1** Controls for the same video in Chrome, Safari, Firefox, and IE 9 (top to bottom).

But with the DOM API for video, you can write and style your own video controls. Some controls you can use are

- **`play()`**—To play the media
- **`pause()`**—To pause the media

These are very easy to use; all you need to do is leave off the `controls` attribute on your media element, and then use some buttons and JavaScript to get your media to play:

```
<video height="600" width="800">
    <source src="Shasta.mp4">
    <source src="Shasta.theora.ogv">
    <source src="Shasta.webm">
</video>
<script>
var video = document.getElementsByTagName('video')[0];
</script>
<p>
<button value="Play" class="play" onclick="video.play()">
<img src="images/Play.png" width="32" height="32" alt="Play"></button>
<button value="Pause" class="pause" onclick="video.pause()">
<img src="images/Pause.png" width="32" height="32" alt="Pause"></button>
```

Then, to use just one button to play and pause, you need to listen for the `play()` and `pause()` events. Remember that the user can play or pause by right-clicking on the video, so you shouldn't just have it alternate on click:

```
<script>
var video = document.getElementsByTagName('video')[0];
video.onpause = video.onplay = function(e) {
```

```
  playPause.value = video.paused ? 'Play' : 'Pause';
}
function playPause() {
  if (video.paused || video.ended) {
    video.play();
  } else {
    video.pause();
  }
}
</script>
<button value="Play/Pause" id="playPause" onclick="playPause()">
<img src="images/PlayPause.png" width="26" height="28"
➥alt="Play/Pause"></button>
```

You can do a lot more with the video functions. You can add volume and mute buttons, fast forward, scanning, and so on. However, the easiest way to get a custom video player on your web pages is to use a pre-made video player. Many free ones are available to choose from. Philip Bräunlich has a list of players (http://praegnanz.de/html5video/index.php) that gives you details such as the license, what JavaScript library it uses, if it has a Flash fallback, if it supports iOS and full screen, and more. He also helpfully evaluates how easy each tool is to integrate and theme.

# Summary

This hour took you through the basics of adding HTML5 video and audio to your web pages. This topic is challenging, with many pitfalls, but you should have a good sense of what you need to do to add these multimedia elements to your pages and applications.

You learned the benefits and drawbacks of HTML5 video and audio as well as why you should consider moving to HTML5 for your video and using Flash as the fallback. You also learned that it's not HTML5 itself that causes the difficulties, but the codecs and lack of across-the-board browser support for them.

You learned the three best codecs for both video and audio and which browsers support which codecs. Plus, you learned some of the pitfalls that can happen with Android and iOS browsers and how to avoid them.

You also learned how to build a fully functional page with video and audio that works in IE8 and other older browsers as well as all the modern HTML5 browsers.

# Q&A

**Q.** ***How is the*** `<track>` ***element intended to be used?***

**A.** This element is added to the `<video>` and `<audio>` elements to define information that is synchronized to the media. It can be a sign-language translation video, a text description, or captions and subtitles, which people are already familiar with.

**Q.** ***Can I use other codecs than the ones mentioned in this hour?***

**A.** If you know the MIME type for the codec, you can include it as a source file for your media. You should be aware, though, that there is no guarantee that a browser or user agent will know what to do with it.

**Q.** ***Is it really necessary to save my video in so many formats?***

**A.** This is one of the most common objections to using the `<video>` and `<audio>` elements in HTML5. But the reality is that if you want your video to be seen as professional, you have to create multiple versions. For example, when you go to the Apple movie trailers site, each trailer is offered in several different sizes. Site visitors choose the one they want to view. But with HTML5 and media queries, you can help users get the best quality video for their browser without them doing anything other than clicking "Play." They don't have to guess; you can just deliver the right one to them. But you can't do that if you only have one option.

# Workshop

The workshop contains quiz questions to help you process what you've learned in this chapter. Try to answer all the questions before you read the answers. Refer to Appendix A, "Answers to Quizzes," for answers.

## Quiz

1. What are the three commonly used audio codecs?
2. What video codecs does Internet Explorer 9 support? What about Android?
3. Name two ways you can include the source of a video in your HTML.
4. True or False. The `loop` attribute is only valid on the `<video>` element.

5. What is the format for the `type` attribute on the `<source>` element?
6. How does the fallback option work on an `<audio>` element?

## Exercises

1. Add an audio file to a web page. Make sure that it plays in as many browsers as possible.
2. Choose a video player (perhaps from http://praegnanz.de/html5video/index.php) and add some HTML5 video to your application. Test to make sure that it plays on both iOS and Android.