CHAPTER 29

HTML5 Audio and Video

One of the biggest driving forces behind the growth of the internet has been the insatiable demand from users for ever more multimedia in the form of audio and video. Initially, bandwidth was so precious that there was no such thing as live streaming, and it could take minutes or even hours to download an audio track, let alone a video.

The high cost of bandwidth and limited availability of fast modems drove the development of faster and more efficient compression algorithms, such as MP3 audio and MPEG video, but even then the only way to download files in any reasonable length of time was to drastically reduce their quality.

One of my earlier internet projects, back in 1997, was the UK's first online radio station licensed by the music authorities. Actually, it was more of a podcast (before the term was coined) because we made a daily half-hour show and then compressed it down to 8-bit, 11 KHz mono using an algorithm originally developed for telephony, and it sounded like phone quality, or worse. Still, we quickly gained thousands of listeners who would download the show and then listen to it as they surfed to the sites discussed in it by means of a pop-up browser window containing a plug-in.

Thankfully for us, and everyone publishing multimedia, it soon became possible to offer greater audio and video quality, but still only by asking the user to download and install a plug-in player. Flash became the most popular of these players, after beating rivals such as RealAudio, but it gained a bad reputation as the cause of many a browser crash and constantly required upgrading when new versions were released.

So, it was generally agreed that the way ahead was to come up with some web standards for supporting multimedia directly within the browser. Of course, browser developers such as Microsoft and Google had differing visions of what these standards should look like, but when the dust settled, they had agreed on a subset of file types that all browsers should play natively, and these were introduced into the HTML5 specification.

Finally, it is possible (as long as you encode your audio and video in a few different formats) to upload multimedia to a web server, place a couple of HTML tags in a web page, and play the media on any major desktop browser, smartphone, or tablet device, without the user having to download a plug-in or make any other changes.

About Codecs

The term *codec* stands for en*co*der/*dec*oder. It describes the functionality provided by software that encodes and decodes media such as audio and video. In HTML5 there are a number of different sets of codecs available, depending on the browser used.

One complication around audio and video, which rarely applies to graphics and other traditional web content, is the licensing carried by the formats and codecs. Many formats and codecs are provided for a fee, because they were developed by a single company or consortium of companies that chose a proprietary license. Some free and open source browsers don't support the most popular formats and codecs because it is unfeasible to pay for them, or because the developers oppose proprietary licenses in principle. Because copyright laws vary from country to country and because licenses are hard to enforce, the codecs can usually be found on the web for no cost, but they might technically be illegal to use where you live.

Following are the codecs supported by the HTML5 `<audio>` tag (and also when audio is attached to HTML5 video):

AAC
: This audio codec, which stands for Advanced Audio Encoding, is a proprietary patented technology that generally uses the *.aac* file extension. Its MIME type is `audio/aac`.

FLAC
: This audio codec, which stands for Free Lossless Audio Codec, was developed by the Xiph.Org Foundation. It uses the *.flac* extension, and its Mime type is `audio/flac`.

MP3
: This audio codec, which stands for MPEG Audio Layer 3, has been available for many years. Although the term is often (incorrectly) used to refer to any type of digital audio, it is a proprietary patented technology that uses the *.mp3* extension. Its Mime type is `audio/mpeg`.

PCM
: This audio codec, which stands for Pulse Coded Modulation, stores the full data as encoded by an analog-to-digital converter and is the format used for storing data on audio CDs. Because it does not use compression, it is called a *lossless* codec, and its files are generally many times larger than AAC or MP3 files. It usually has the extension *.wav*. Its MIME type is `audio/wav`, but you may also see `audio/wave`.

Vorbis
: Sometimes referred to as Ogg Vorbis—because it generally uses the *.ogg* file extension—this audio codec is unencumbered by patents and free of royalty payments. Its MIME type is `audio/ogg`, or the WebM container uses `audio/webm`.

As of mid 2021, all of AAC, MP3, PCM, and Vorbis are generally supported by most operating systems and browsers (not including Microsoft's discontinued Internet Explorer), with the following Safari-related exceptions:

Vorbis `audio/ogg`
: MacOS 10.11 and earlier Safari requires Xiph Quicktime

Vorbis `audio/webm`
: Not supported on Safari

FLAC `audio/ogg`
: Not supported on Safari (although `audio/flac` is)

Consequently, unless you really have a reason to use Vorbis, it is nowadays safe to just stick with either AAC or MP3 for compressed lossy audio, FLAC for compressed lossless, or PCM for uncompressed audio.

The <audio> Element

To cater for various platforms, you can record or convert your content using multiple codecs and then list them all within `<audio>` and `</audio>` tags, as in Example 29-1. The nested `<source>` tags then contain the various media you wish to offer to a browser. Because the `controls` attribute is supplied, the result looks like Figure 29-1.

Example 29-1. Embedding three different types of audio files

```
<audio controls>
  <source src='audio.m4a' type='audio/aac'>
  <source src='audio.mp3' type='audio/mp3'>
  <source src='audio.ogg' type='audio/ogg'>
</audio>
```

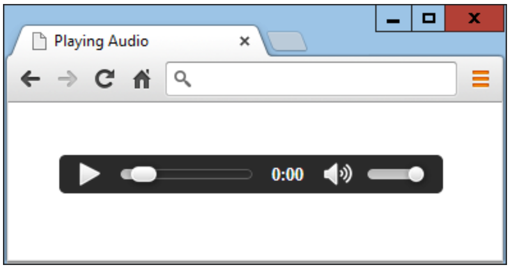

Figure 29-1. Playing an audio file

In this example I included three different audio types, because that's perfectly acceptable and can be a good idea if you wish to ensure that each browser can locate its preferred format rather than just one it knows how to handle. However, you can drop either (but not both) of the MP3 or the AAC files and still be confident that the example will play on all platforms.

The `<audio>` element and its partner `<source>` tag support several attributes:

`autoplay`
: Causes the audio to start playing as soon as it is ready

`controls`
: Causes the control panel to be displayed

`loop`
: Sets the audio to play over and over

`preload`
: Causes the audio to begin loading even before the user selects Play

`src`
: Specifies the source location of an audio file

`type`
: Specifies the codec used in creating the audio

If you don't supply the `controls` attribute to the `<audio>` tag, and don't use the `autoplay` attribute either, the sound will not play, and there won't be a Play button for the user to click to start playback. This would leave you no option other than to offer this functionality in JavaScript, as in Example 29-2 (with the additional code required highlighted in bold), which provides the ability to play and pause the audio, as shown in Figure 29-2.

Example 29-2. Playing audio using JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <title>Playing Audio with JavaScript</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <audio id='myaudio'>
      <source src='audio.m4a' type='audio/aac'>
      <source src='audio.mp3' type='audio/mp3'>
      <source src='audio.ogg' type='audio/ogg'>
    </audio>

    <button onclick='playaudio()'>Play Audio</button>
    <button onclick='pauseaudio()'>Pause Audio</button>

    <script>
      function playaudio()
      {
        O('myaudio').play()
      }
      function pauseaudio()
      {
        O('myaudio').pause()
      }
    </script>
  </body>
</html>
```

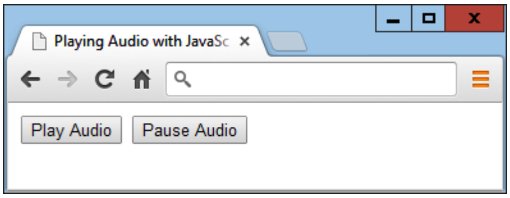

Figure 29-2. HTML5 audio can be controlled with JavaScript

This works by calling the `play` or `pause` methods of the `myaudio` element when the buttons are clicked.

The <video> Element

Playing video in HTML5 is quite similar to audio; you just use the `<video>` tag and provide `<source>` elements for the media you are offering. Example 29-3 shows how to do this with three different video codec types, as displayed in Figure 29-3.

Example 29-3. Playing HTML5 video

```
<video width='560' height='320' controls>
  <source src='movie.mp4'  type='video/mp4'>
  <source src='movie.webm' type='video/webm'>
  <source src='movie.ogv'  type='video/ogg'>
</video>
```

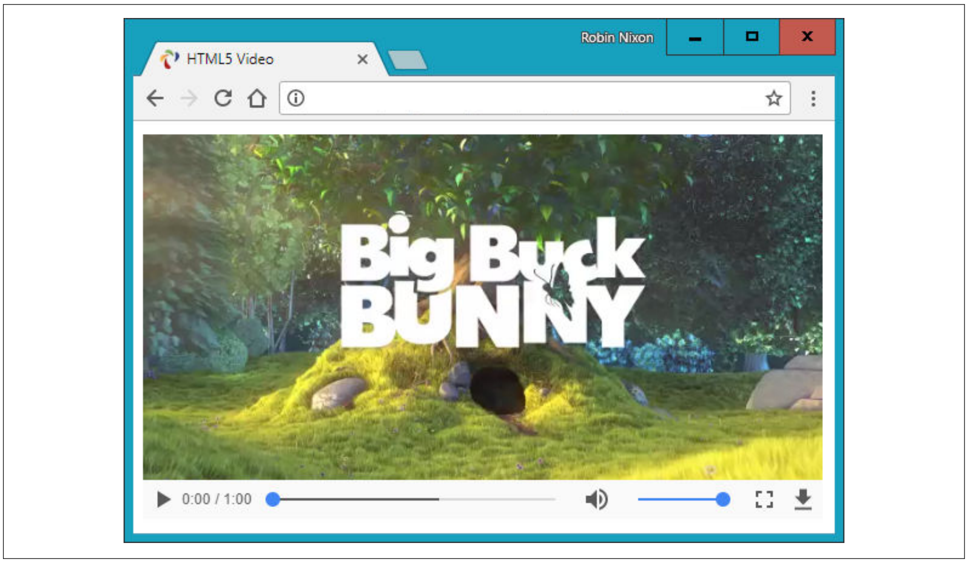

Figure 29-3. Playing HTML5 video

The Video Codecs

As with audio, there are a number of video codecs available, with differing support across multiple browsers. These codecs come in different containers, as follows:

MP4
: A license-encumbered, multimedia container format standard specified as a part of MPEG-4. Its MIME type is `video/mp4`.

Ogg
: A free, open container format maintained by the Xiph.Org Foundation. The creators of the Ogg format state that it is unrestricted by software patents. Its MIME type is `video/ogg`.

WebM
: An audio-video format designed to provide a royalty-free, open video compression format for use with HTML5 video. Its MIME type is `video/webm`.

These may then contain one of the following video codecs:

H.264 & H.265
: Patented, proprietary video codecs that can be played back for free by the end user but that may incur royalty fees for all parts of the encoding and transmission process. H.265 supports almost double the compression of H.264 for the same quality output.

Theora
: A video codec that is unencumbered by patents and free of royalty payments at all levels of encoding, transmission, and playback.

VP8
: A video codec that is similar to Theora but is owned by Google, which has published it as open source, making it royalty-free.

VP9
: The same as VP8 but more powerful, using half the bitrate.

Nowadays you can be sure that pretty much all modern browsers support all of these, with the exceptions that Theora `video/ogg` is not supported on iOS, and macOS earlier than 10.11 requires Xiph QuickTime.

Therefore you may wish to steer clear of Ogg if iOS is one of your target platforms (which will usually be the case), as you can safely rely on either MP4 or WebM on all platforms and forget about other formats for the time being. However, in Example 29-3 I have shown how you can add all the three main video types if you like, as the browser will then choose the format it prefers.

The `<video>` element and accompanying `<source>` tag support the following attributes:

`autoplay`
: Causes the video to start playing as soon as it is ready

`controls`
: Causes the control panel to be displayed

`height`
: Specifies the height at which to display the video

`loop`
: Sets the video to play over and over

`muted`
: Mutes the audio output

`poster`
: Lets you choose an image to display where the video will play

`preload`
: Causes the video to begin loading before the user selects Play

`src`
: Specifies the source location of a video file

`type`
: Specifies the codec used in creating the video

`width`
: Specifies the width at which to display the video

If you wish to control video playback from JavaScript, you can do so using code such as that in Example 29-4 (with the additional code required highlighted in bold), with the results shown in Figure 29-4.

Example 29-4. Controlling video playback from JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <title>Playing Video with JavaScript</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <video id='myvideo' width='560' height='320'>
      <source src='movie.mp4'  type='video/mp4'>
      <source src='movie.webm' type='video/webm'>
      <source src='movie.ogv'  type='video/ogg'>
    </video><br>

    <button onclick='playvideo()'>Play Video</button>
    <button onclick='pausevideo()'>Pause Video</button>

    <script>
      function playvideo()
      {
        O('myvideo').play()
      }
      function pausevideo()
      {
        O('myvideo').pause()
      }
    </script>
  </body>
</html>
```

This code is just like that for controlling audio from JavaScript. Simply call the `play` and/or `pause` methods of the `myvideo` object to play and pause the video.

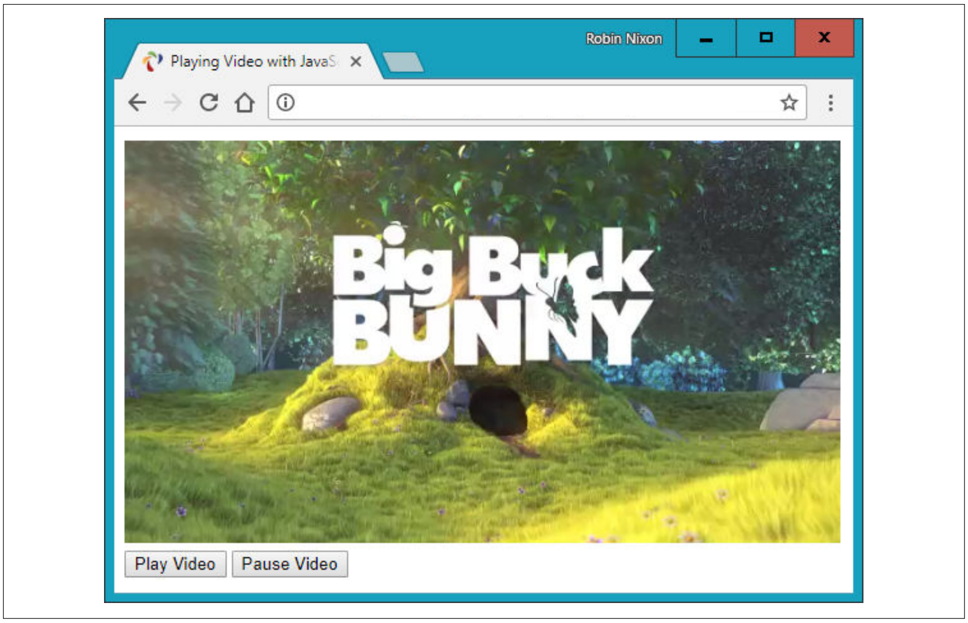

Figure 29-4. JavaScript is being used to control the video

Using the information in this chapter, you will be able to embed any audio and video you like on almost all browsers and platforms without worrying about whether users may or may not be able to play it.