HTML5 Multimedia

DEVELOP AND DESIGN

Ian Devlin

HTML5 Multimedia

DEVELOP AND DESIGN

Ian Devlin

HTML5 Multimedia: Develop and Design
Ian Devlin

Peachpit Press
1249 Eighth Street
Berkeley, CA 94710
510/524-2178
510/524-2221 (fax)

Find us on the Web at: www.peachpit.com
To report errors, please send a note to: errata@peachpit.com
Peachpit Press is a division of Pearson Education.

Editor: Rebecca Gulick
Development and Copy Editor: Anne Marie Walker
Technical Reviewer: Chris Mills
Production Coordinator: Myrna Vladic
Compositor: David Van Ness
Proofreader: Patricia Pane
Indexer: Valerie Haynes-Perry
Cover Design: Aren Howell Straiger
Cover Production: Jaime Brenner
Interior Design: Mimi Heft

ISBN-13: 978-0-321-79393-5
ISBN-10: 0-321-79393-5

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

Dedicated to the memory of Paul Fallon

Tá daoine a shiúlann inár saolta agus shiúlann amach astu go luath
Tá daoine a fhanann ar feadh tamaill
Agus fágann said rianta a gcos ar ár gcroíthe
Agus casann ár n-anamacha port nua go deo deo

ACKNOWLEDGMENTS

Writing a book is a time-consuming and difficult process, and one I knew nothing about before embarking on this project. A number of people have helped me through the book-writing process, and others have helped me through the HTML5 process, whether they know it or not. All deserve my thanks.

To Rebecca Gulick for giving me the opportunity to actually write this book and for clearly explaining to me the steps involved.

To Anne Marie Walker for ensuring that my words are clear and understandable.

To Chris Mills for his editing and technical reviewing skills, and providing me with many good suggestions and alterations throughout the text.

To Rich Clark for giving me the opportunity to curate for HTML5 Gallery (www.html5gallery.com), which not only increased my interest in and knowledge of HTML5, but it also led to me writing this book.

To Remy Sharp for first drawing my attention to HTML5 in an article in *.net* magazine back in October 2009.

To you, the reader, for deciding to purchase this book with the intention of learning. I hope you find it enjoyable and educational.

CONTENTS

INTRODUCTION

As a web developer or web designer, or those of you who just maintain your own website, you know that the web is constantly changing, and the tools and methods that are used to build websites are in constant development. Like sand dunes in the Sahara, they shift constantly, but fortunately, usually in a forward direction.

The shift in web technologies has currently arrived at HTML5, the latest version of the language used to define and build web pages. With it comes an easier method of adding multimedia to your web pages.

The goal of this book is to provide you with an introduction to adding audio and video to your website, and to give you a glimpse of what else you can do with HTML5 multimedia.

Throughout the book you'll find in-depth details of the various HTML5 multimedia elements, as well as full code examples on how you can use them to add audio and video to your website. You'll also learn about the accompanying JavaScript API that allows you to create your own media controls.

In addition, you'll find explanations and examples of how you can style the HTML5 media elements with CSS, including some of the new features that CSS3 has to offer. You'll also learn about multimedia and accessibility, and how you can add subtitles to your website video.

WHO THIS BOOK IS FOR

This book is aimed at those who are starting out with HTML5 and adding HTML5 audio and video to their websites, and those who are already familiar with HTML5 multimedia but want to learn more.

Some basic knowledge of HTML and CSS is assumed, and the later chapters require at least a rudimentary knowledge of JavaScript. That said, all the examples on the book's accompanying website at www.html5multimedia.com are complete.

SCREEN SHOTS AND BROWSER VERSIONS

During the course of writing this book, some browser vendors released newer versions of their products. Firefox is now on version 7, Chrome is on version 14, and Safari has moved to 5.1. The screen shots in the book usually indicate which browser and version it was taken from at the time the chapter was written. This, of course, means that some of the screen shots are from older versions of the browser. But rest assured that they still work just as well in the latest versions, and if they don't, it is clearly marked.

THE WEBSITE FOR THIS BOOK

All the code used in the examples in this book is on the accompanying website at www.html5multimedia.com. You can either download the files in their entirety or navigate to the various files via the website and see them working online.

CONTACT

If you would like to contact me, you can do so at info@html5multimedia.com.

BEFORE YOU BEGIN

In the later chapters of this book, some of what you'll read is experimental due to specifications that were still in development at the time of this writing and poor or nonexistent support in browsers. This of course may have changed by the time you have this book in your hands. The book's website will indicate improved support where applicable.

It's time to begin! Let's start by taking a quick look at HTML5, what it is, and where it comes from.

This page intentionally left blank

4

USING **VIDEO**

The popularity of video-sharing sites such as YouTube and Vimeo, combined with bandwidth speeds that makes online video feasible, have led to a huge demand for embedding video in web documents. Yet until recently the only way was by using third-party plugins such as Flash and QuickTime. HTML5 provides that missing standard method for embedding videos in web documents. Major browsers have begun to support it in their latest releases, so you can be confident that modern browsers can handle your video content.

This chapter covers the file formats and codecs supported by HTML5 video, how to convert between formats, and solutions to issues you might encounter. You'll also learn how to deliver video to browsers that don't support HTML5 video.

VIDEO CODECS AND FILE FORMATS

As with audio, HTML5 video has a number of different formats that you can use to encode video content in due to browser vendors being unable to agree on a standard. The video file formats available include Theora Ogg, MP4 (H.264), and WebM. Let's look at each in detail.

CODECS AND CONTAINERS

As mentioned in Chapter 3, a *codec* is a computer program that uses a compression algorithm to encode and/or decode a digital stream of data, making it more suitable for playback.

A *container* is a wrapper format whose specification describes how the different data elements within the container exist and interact together within a computer file.

THEORA OGG

Theora Ogg, as you've probably guessed, is also from the Xiph.Org Foundation (www.xiph.org). Like its audio counterpart, Theora Ogg is free and open, and has no licensing or royalty issues.

As with audio, Ogg is the name of the container format, and in this case Theora refers to the video-compression format that it uses. Earlier versions of the Theora codec showed it to be inferior to other similar codecs at the time, but it has improved a great deal and is now considered comparable to YouTube's H.264 output (before YouTube started encoding high-definition video).

Theora Ogg uses the `application/ogg` MIME type and the `video/ogg` video attribute type.

MP4 (H.264)

MPEG-4 Part 10, or MP4, is a compressed video format, which like the MP3 audio format (see Chapter 3) was defined by the Moving Picture Experts Group (MPEG; www.mpeg.org). It was developed to deliver DVD-quality video and audio in a small package. This small file size makes MP4 files highly suitable for portable players, and naturally, the web.

H.264 has been split into 17 different "profiles"; each of which provides additional features that usually increase the file size. Some are suitable for use with HTML5 video, whereas others are not. The Baseline and Main profiles are usually used for HTML5 video. For a full list of these profiles, see en.wikipedia.org/wiki/H.264#Profiles.

MP4 uses the `video/mpeg` MIME type and the `video/mp4` video attribute type.

WEBM

WebM is a project (www.webmproject.org) that is supported by web-industry giants, such as Mozilla, Opera, Adobe, and Google. The aim of the project is to produce a high-quality, royalty-free, open video format.

The video content is compressed with the VP8 codec, which was developed by On2 Technologies (the company was acquired by Google in February 2010). The codec tends to be used within the WebM container.

WebM uses the `video/webm` MIME type and the `video/webm` video attribute type.

BROWSER SUPPORT FOR VIDEO FORMATS

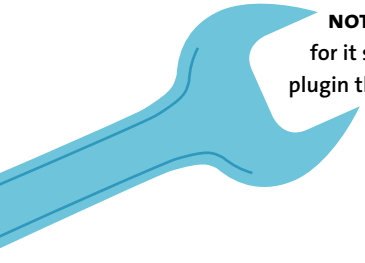
When the first draft of the HTML5 specification was released, it recommended that browsers should support the Theora Ogg video format. Knowing that all browsers that supported HTML5 video would support a standard video format would have allowed you to guarantee availability of your video content to users when you served up a Theora Ogg video file. However, note the use of the phrase "would have."

Unfortunately, like the audio specification, both Nokia and Apple objected to the requirement to support Theora Ogg, which they regarded as not being widely supported and not as open and free as the Xiph.Org Foundation claimed. So the requirement was removed. As a result, you're back to having to supply your video content in more than one format to guarantee coverage of all browsers that support HTML5 video.

Table 4.1 contains a list of which vendors support certain video formats in their browsers.

TABLE 4.1 Video Formats and Browser Support

FORMAT	BROWSER				
Theora Ogg	Firefox 3.5+	Chrome 5+	Opera 10.5+		
MP4/H.264	Safari 3+	Chrome 5–?	IE9+	iOS	Android 2+
WebM	Firefox 4+	Chrome 6+	Opera 11+	IE9+	Android 2.3+



NOTE: Chrome currently supports MP4/H.264 but will drop support for it soon. Internet Explorer 9 will support WebM as long as a third-party plugin that can play it is installed.

In fact, you need to serve up at least two different formats, MP4 and WebM, in order to support the latest versions of the major browsers. This isn't too much of a chore if you're only serving up a few videos, because it can be easily done using the `video` and `source` elements mentioned in Chapter 2. For video-intensive sites, it can unfortunately be a burden because many files need to exist in at least two different formats, thus doubling the storage space required.

MOBILE **DEVICES**

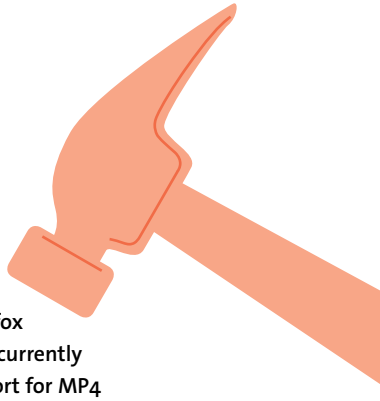
Another good reason to use HTML5 video is Apple's decision not to support Flash in the iPhone and iPad. iOS, the operating system that runs on these phones, has support for MP4 video.

Android phones also support MP4 (and WebM from version 2.3), Windows Phone 7 has native support for both MP4 and WebM, and Blackberry supports MP4 from version 7 of its browser.

Because you'll have your video file in one format, the easiest way of providing the two formats is by converting from one format to the other.

Let's take a quick look at how you'd convert your video content between the different formats before diving into the code examples.

ENCODING YOUR VIDEO FILES



You might already have the video content that you want to display on your web document, or you might still need to record it. Let's assume that you already have the content, and that it is in one of the aforementioned formats, although it could just as easily not be. Either way it's not a problem because there are plenty of encoders on the market that you can use to convert your video content from one format to another.

TIP: It's wise to support at least MP4 and WebM to cover the latest versions of the major browsers. Safari supports MP4; Firefox and Opera support WebM; and Chrome and Internet Explorer 9 currently support both MP4 and WebM, although Chrome will drop support for MP4 in the near future and IE9 needs a plugin for WebM. However, you might also choose to support Theora Ogg, because WebM wasn't supported by Firefox until version 4, and Firefox doesn't support MP4 at all.

Here are three of my favourite encoders:

- **Miro Video Converter** (http://www.mirovideoconverter.com). As well as converting between audio formats, this converter also supports conversion of video files to Theora Ogg, MP4, and WebM. It really is all you need and runs on both Windows and Mac.
- **Handbrake** (http://handbrake.fr). This open source converter allows you to convert video files to MP4 and the Theora Ogg format. It runs on Windows, Mac, and Linux.
- **Media Converter** (http://www.mediaconverter.org). This online conversion application allows you to upload a file for conversion or provide the URL of an existing file. It allows you to convert files to Theora Ogg, MP4, and Flash FLV, among others.

Although you can choose from many other encoders, these are three solid encoders that you can use to get started.

MP4 ENCODING AND DELAYED PLAYBACK

Sometimes the way an MP4 file is coded can cause problems with its playback. Namely, the file doesn't start playing until it has downloaded completely. This is due to the encoding process placing the file index—with all the metadata on file length and so on—at the end of the file rather than the beginning.

If you find this is happening to your MP4 files, you can fix the problem by running the files through the QTIndexSwapper (http://renaun.com/blog/code/qtindexswapper) by Renaun Erickson of Adobe. QTIndexSwapper simply moves the index to the beginning of the file.

Now that you've converted your files, you're ready to start using them within your documents!

USING THE VIDEO ELEMENTS

Let's begin with some basic examples of embedding video files within a web document. You'll have previously encountered all the elements and attributes used in the examples in Chapter 2, so nothing should be new to you.

PLAYING A VIDEO FILE

The easiest example of them all is to play a simple video file of one format with default media controls for the user.

To play a WebM file, you use:

```
<video src="snowy-tree.webm" controls></video>
```

The `control` attribute informs the browser that it should display a set of basic video controls on top of the video player.

If you wanted the video to start playing as soon as the page loads, you could simply add the `autoplay` attribute:

```
<video src="snowy-tree.webm" controls autoplay></video>
```

You might also want the video to start playing immediately and then keep playing in a loop via the `loop` attribute:

```
<video src="snowy-tree.webm" controls autoplay loop></video>
```

It is, however, strongly advised not to do this: Not only is it annoying, but it can be an accessibility nightmare because a looping video file might play over added audio that's inserted for accessibility reasons.

You could also mute the video file on startup by using the `muted` attribute. Of course, if your video has no sound, this has no effect:

```
<video src="snowy-tree.webm" controls autoplay muted></video>
```

NOTES: None of the major browsers currently support the `muted` attribute. However, you can set it via the Media JavaScript API, which is discussed in Chapter 5.

All of the examples in this section and more can be found on the accompanying website at www.html5multimedia.com.

FIGURE 4.1 Restoring the browser's default media controls via the browser menu in Firefox 5.

By default, the browser will start loading the entire video file when the page loads. If you would prefer that the browser not do this (perhaps you think it's unlikely that users will want to view the video and don't want to waste bandwidth because they might be viewing your site over a mobile network), you can use the `preload` attribute and set it to `none`:

```
<video src="snowy-tree.webm" controls preload="none"></video>
```

You can also request that the browser load the video's metadata (e.g., file length) by setting the `preload` attribute to `metadata`:

```
<video src="snowy-tree.webm" controls preload="metadata"></video>
```

Any setting of the `preload` attribute merely suggests to the browser what your intentions are, but ultimately, the browser will decide what to do. The browser may, for example, ignore your suggestion due to a user setting in the browser.

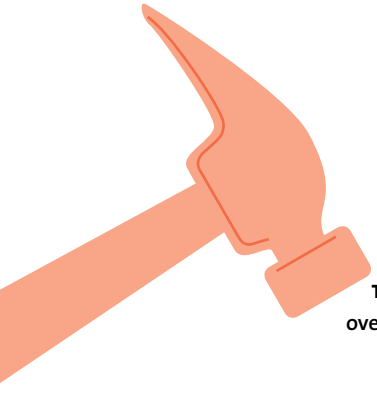
If you want to hardcode the width and height of the video rather than letting the browser automatically decide for you, you can do so via the `width` and `height` attributes:

```
<video src="snowy-tree.webm" controls width="300" height="210">
→ </video>
```

You can also remove the controls entirely by omitting the `controls` attribute:

```
<video src="snowy-tree.webm"></video>
```

Note that the user can restore the default controls in most browsers by right-clicking on the video and selecting the controls from the displayed drop-down menu **Figure 4.1**.



TIP: If you also specify `autoplay`, the `preload` setting will be overridden because the video must be downloaded for it to play!

All of the preceding examples use just one video file format. But because you'll need to serve up more than one video file format to cover all major browsers, let's take a look at how to do that next.

PLAYING A VIDEO FILE WITH DIFFERENT SOURCES

Presenting different video file formats to the browser is quite easy using the `source` element, which you also used in the audio examples in Chapter 3.

Here is the code you need to provide two different sources for the video to play:

```
<video controls>
    <source src="snowy-tree.mp4" type="video/mp4">
    <source src="snowy-tree.webm" type="video/webm">
</video>
```

But let's also support Theora Ogg, just in case a Firefox 3.5 or Opera 10.5 user wants to view your video:

```
<video controls>
    <source src="snowy-tree.ogv" type="video/ogg">
    <source src="snowy-tree.mp4" type="video/mp4">
    <source src="snowy-tree.webm" type="video/webm">
</video>
```

NOTE: You probably won't need to add support for Theora Ogg, and you should only really bother if you know that you need to support specific older versions of Firefox.

The examples in the previous section, "Playing a Video File," where different attributes were applied to show you how they work and what they do, also apply to any `video` element that contains multiple `source` elements.

THE `type` ATTRIBUTE

When you use the `source` element within the `video` element, you'll notice that the `type` attribute moves from the `video` element to the `source` element. The reason is that the whole idea of serving up different sources is because they use different formats, and each source element needs to specify the format the source is in via its own `type` attribute.

The `type` attribute can also contain the actual codec that the video file is encoded in. For example:

```
<source src="snowy-tree.mp4" type='video/ogg; codecs="theora, vorbis"'>
<source src="snowy-tree.mp4" type='video/webm; codecs="vp8, vorbis"'>
<source src="snowy-tree.mp4" type='video/mp4; codecs="mp4a.40.2"'>
```

Including the codec in the `type` attribute can be beneficial because it helps the browser decide if it can play the file or not. It's best to only include the codec if you know for certain which codec was used to encode your video content.

Should you decide to include the codec, be *very careful* and ensure that you format the string correctly, paying particular attention to the quotes used; otherwise, the browser won't recognise the source.

In the preceding example, note how the entire string is enclosed within single quotes, the `type` and `codecs` attributes are separated by a semicolon, and the `codecs` values are contained within a double quote.

If you want to autoplay and loop your video, you would add the `autoplay` and `loop` attributes to the `video` element like this:

```
<video controls autoplay loop>
    <source src="snowy-tree.mp4" type="video/mp4">
    <source src="snowy-tree.webm" type="video/webm">
</video>
```

You can of course also autoplay and remove the controls like this:

```
<video autoplay>
    <source src="snowy-tree.mp4" type="video/mp4">
    <source src="snowy-tree.webm" type="video/webm">
</video>
```

FIGURE 4.2 The image on the left displays the first still from the video that the browser shows by default; on the right, the same video is shown but with a defined poster image displayed instead.

Notice that when autoplay is off, the first still from the video is displayed in the browser as an image. You might want to use a different image if the first still from the video isn't what you want to display; it might be blank or just not the still you want to show first.

If you want to use a different image, you can use the `poster` attribute to point at an image file to use instead:

```
<video controls poster="snowy-tree-poster.gif" width="300"
→ height="210">
    <source src="snowy-tree.mp4" type="video/mp4">
    <source src="snowy-tree.webm" type="video/webm">
</video>
```

You can get an idea of how the `poster` attribute works in **Figure 4.2**.

Now that you know how to play video files, you might want to think about the legacy browser fallback. How do you show video in legacy browsers? Let's take a look.

PLAYING A VIDEO FILE WITH DIFFERENT SOURCES AND LEGACY FALLBACK

Throughout this chapter I've recommended providing a fallback for legacy browsers, such as Internet Explorer 6 to 8, that don't support HTML5 and native multimedia. This of course means reverting to an old third-party plugin that these browsers understand, such as Flash.

Because browsers ignore what they don't understand, legacy browsers will ignore the `video` and `source` elements, and act as if they don't exist. This of course allows you to provide a simple link to the video file so it can be downloaded:

```
<video controls autoplay>
    <source src="snowy-tree.mp4" type="video/mp4">
    <source src="snowy-tree.webm" type="video/webm">
    <a href="snowy-tree.mp4">Download the video: snowy-tree.mp4</a>
</video>
```

You might prefer to actually provide an image link rather than a simple text link:

```
<video controls autoplay>
    <source src="snowy-tree.mp4" type="video/mp4">
    <source src="snowy-tree.webm" type="video/webm">
    <figure>
        <a href="snowy-tree.mp4">
            <img src="snowy-tree.gif" alt="Branches of a fern tree
            → covered in snow" height="210" width="300" />
            <figcaption>Download the video: snowy-tree.mp4
            → </figcaption>
        </a>
    </figure>
</video>
```

FIGURE 4.3 This is how the image and link to download the MP4 file might be displayed in Internet Explorer 8.

Figure 4.3 shows how this might look in Internet Explorer.

If you decide to support Flash and allow non-HTML5 browsers to play your video via Flash, you can of course do so using either the `embed` or `object` elements. You can then play the video using a downloaded Flash player (which you have uploaded to your server) and the `object` element.

You can also take advantage of the fact that Flash can play MP4 files, so there's no need to create another file in a different format. The following code shows how a Flash fallback can be achieved:

```
<video controls autoplay>
    <source src="snowy-tree.mp4" type="video/mp4">
    <source src="snowy-tree.webm" type="video/webm">
    <object type="application/x-shockwave-flash"
        data="player.swf?videoUrl=snowy-tree.mp4&autoPlay=true"
        height="210" width="300">
        <param name="movie"
        value="player.swf?videoUrl=snowy-tree.mp4&autoPlay=true">
    </object>
</video>
```

A non-HTML5 browser will ignore the two `source` elements because it doesn't know what to do with them. It will then recognise the `object` element, and provided Flash is installed, will play the video through the Flash player.

The same code using the embed element looks like this:

```
<video controls autoplay>
    <source src="snowy-tree.mp4" type="video/mp4">
    <source src="snowy-tree.webm" type="video/webm">
    <embed type="application/x-shockwave-flash" wmode="transparent"
        src="player.swf?videoUrl=snowy-tree.mp4&autoPlay=true"
        height="210" width="300">
</video>
```

It's better practice to use object instead of embed, because any content in the object start and end tags will be rendered even if the browser doesn't support the plugin that the object element specifies in its type attribute. This allows you to specify yet another fallback should you need to. As you can see in the previous object example, the param element will be read by browsers that don't understand the value specified by the type attribute in the object element.

Of course, you still need to rely on the fact that users have the Flash player installed on their computers, but this may not always be the case. Therefore, you can also add the image download link mentioned earlier as a final fallback, just in case Flash isn't installed:

```
<video controls autoplay>
    <source src="snowy-tree.mp4" type="video/mp4">
    <source src="snowy-tree.webm" type="video/webm">
    <object type="application/x-shockwave-flash"
        data="player.swf?videoUrl=snowy-tree.mp4&autoPlay=true"
        height="210" width="300">
        <param name="movie"
        value="player.swf?videoUrl=snowy-tree.mp4&autoPlay=true">
    </object>
    <a href="snowy-tree.mp4">Download the video: snowy-tree.mp4</a>
</video>
```

The download link here isn't a true fallback in the sense that if the browser falls back to Flash, the download link will also be displayed, but that's not necessarily a bad thing because it just provides another way to access the file.

There's plenty to think about when deciding which browsers you want to support and which fallbacks you want to provide to do so. Whatever you decide on, HTML5 multimedia should enable you to get the job done.

VIDEO FOR EVERYBODY!

You can read an excellent article by Kroc Camen of Camen Design (http://camendesign.com) on how to make video available to all without using JavaScript or browser sniffing. Kroc's site is definitely worth checking out for future reading because he keeps it up to date with any new developments or discoveries that he or others make.

You'll find the article at http://camendesign.com/code/video_for_everybody.

Of course, these days it isn't just modern and legacy desktop browsers that you need to worry about supporting. You also need to make your content available to users of modern mobiles, tablets, and other alternative browsing devices with video-playing capabilities. To optimise your web content for such devices, you also need to learn about *media types* and *media queries*, which is what you'll look at next.

TARGETING DEVICES WITH DIFFERENT VIDEO FILES USING MEDIA TYPES AND QUERIES

Let's say you wanted to serve up a different video file depending on the browser's capabilities and size. For example, you might want to play a smaller video, in both dimensions and file size, to a mobile phone that will have a small screen and possibly be retrieving data over a 3G connection. Is this even possible? It is if you use a combination of *media types* and *media queries*, and the `media` attribute in the `source` element.

Media types were introduced in CSS2 (www.w3.org/TR/CSS2/media.html) to enable you to target different devices with specific styling and/or style sheets. **Table 4.2** (on the next page) lists the media types.

TABLE 4.2 Media Types

TYPE	DEFINITION
all	Suitable for all devices.
braille	Aimed at Braille tactile feedback devices.
embossed	Aimed at paged braille printers.
handheld	Intended for handheld devices, such as mobile phones.
print	Targets paged material and material for display in print preview mode.
projection	Suitable for projected presentations.
screen	Suitable for displaying on a colour computer screen.
speech	Intended for speech synthesisers.
tty	Aimed at devices with a fixed-pitch character grid, such as a terminal.
tv	Intended for a television-type device.

You may have come across some of the media types listed in Table 4.2 before, although most of them are probably alien to you. If you've ever created a style sheet for printing content, you'll be familiar with the `print` media type; if you've ever attempted to target a mobile phone, the `handheld` type will also be familiar to you.

But it is the `handheld` type that has caused particular issues as technology has moved on. Initially, phones didn't have browsers that were capable of rendering HTML sites, so developers largely ignored them. When phones became "smarter" and came with improved browsers, the `handheld` media type wasn't being used in websites. Vendors then chose to ignore it and default to the `screen` media type instead. But something was needed to help combat this because website configurations that were meant for full-screen browsers were now rendering on phones, causing many an annoying scroll bar. This is where media queries come in.

Media queries were created by the W3C and have a complete specification of their own (see www.w3.org/TR/css3-mediaqueries). They are an extension to CSS3 media types that allow you to check for conditions of particular media features, such as `width`, `height`, and `orientation`, to deliver either different content or styling. You can check for a number of device features, the list of which appears in **Table 4.3**.

TABLE 4.3 Media Query Device Features

FEATURE	MIN/MAX PREFIXES	DEFINITION
width	Yes	The width of the target display area.
height	Yes	The height of the target display area.
device-width	Yes	The width of the device's rendering area.
device-height	Yes	The height of the device's rendering area.
orientation	Yes	Orientation of the rendering device: `portait` or `landscape`.
aspect-ratio	Yes	Ratio of target `width` to the `height`.
device-aspect-ratio	Yes	Ratio of `device-width` to the `device-height`.
resolution	Yes	Density of pixels in the device.
color	Yes	Number of bits per colour component.
color-index	Yes	Number of entries in colour lookup table.
grid	No	Tests if the device is grid-based or not.
monochrome	Yes	Number of bits per pixel in monochrome device.
scan	No	For TV browsing: `progressive` or `scan`.

The great thing is that you can combine media types and media queries to target certain devices using the *and* keyword:

```
screen and (min-device-width:300px)
```

You can also target all devices that don't match particular settings by using the logical *not* operator keyword:

```
not screen and (max-width:800px)
```

The *only* keyword can also be used to hide the settings from older browsers:

```
only screen and (max-width:800px)
```

Of course, these settings don't work on their own and need to be assigned to the `media` attribute of the required `source` element within the `video` container:

```
<source src="myVideo.webm" media=" screen and
→ (min-device-width:300px)">
```

The following code example serves a different video to all media types that have a maximum width of 600 pixels. Both WebM and MP4 formats are provided. Anything that doesn't match these features will move on to the succeeding source definitions:

```
<video controls>
    <source src="snowy-tree-small.mp4" type="video/mp4"
    → media="all and (max-width:600px)">
    <source src="snowy-tree-small.webm" type="video/webm"
    → media="all and (max-width:600px)">
    <source src="snowy-tree-medium.webm" type="video/webm">
    <source src="snowy-tree-medium.mp4" type="video/mp4">
</video>
```

If you want to also provide a medium-sized video file based on a larger maximum display width of 800 pixels, you can do so like this:

```
<video controls>
    <source src="snowy-tree-small.mp4" type="video/mp4"
    → media="screen and (max-width:600px)">
    <source src="snowy-tree-small.webm" type="video/webm"
    → media="screen and (max-width:600px)">
    <source src="snowy-tree-medium.webm" type="video/webm"
    → media="screen and (max-width:800px)">
    <source src="snowy-tree-medium.mp4" type="video/mp4"
    → media="screen and (max-width:800px)">
    <source src="snowy-tree-large.webm" type="video/webm">
    <source src="snowy-tree-large.mp4" type="video/mp4">
</video>
```

TESTING WITH MEDIA TYPES AND QUERIES

You might be wondering how on earth you would test media types and queries if you don't have specific devices available to you.

With the examples that I've provided, simply changing the size of the browser window and then refreshing the page will usually result in the desired outcome.

You can also use the ProtoFluid application (http://app.protofluid.com), which allows you to load a website (even those running on your local server) and change the view to that of a handful of phones (such as Blackberry and iPhone) and monitors of various sizes, among other devices.

You can see how the code in the section "Targeting Devices with Different Video Files Using Media Types and Queries" works in ProtoFluid in **Figure 4.4** and **Figure 4.5**.

Of course, nothing beats testing your code on the real thing, but that isn't always a viable option given the sheer number of devices on the market.

NOTE: **Firefox completely ignores media queries, so changing the browser window size will have no effect. I suggest using Opera, which does exactly what it's supposed to.**

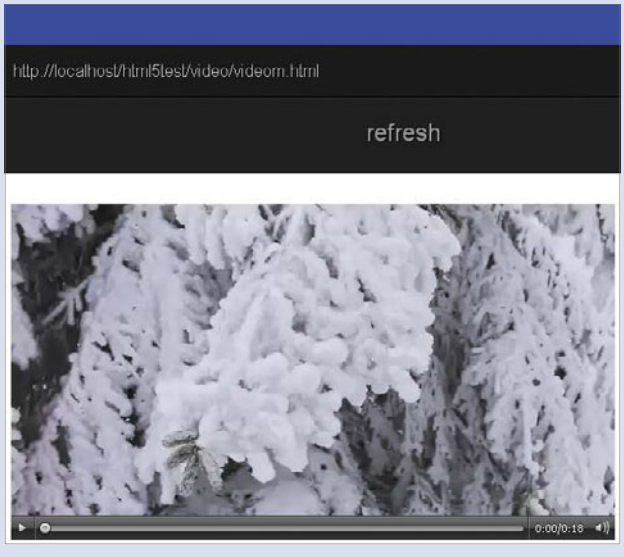


FIGURE 4.4 Selecting the iPhone size in ProtoFluid displays the smaller video file.

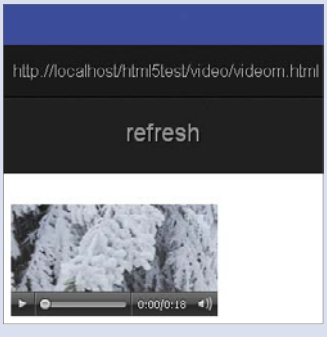


FIGURE 4.5 Selecting the desktop 800×600 size in ProtoFluid displays the medium video file.

Again, any device that has a maximum display size that is larger than 600 pixels or 800 pixels will ignore the smaller sizes and play whichever one of the WebM or MP4 larger videos that it is capable of playing.

These are just some simple examples of what you might want to achieve when targeting different devices. By combining media types and queries, you can target any device to meet your specific requirements. Unfortunately, none of the previous examples discussed will work with Android.

ANDROID AND VIDEO

The implementation of HTML5 video in Android is nothing short of shockingly poor. For this reason, it deserves its own small section to prevent you from tearing your hair out.

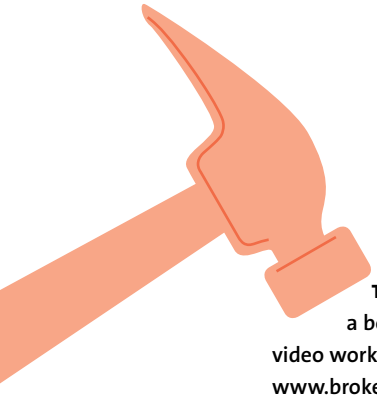
Android supports MP4 files from version 2.0 and WebM from version 2.3.

You shouldn't use the `type` attribute with the `video` or `source` element when defining the video file you want Android to use. For some reason, this confuses Android and it ignores the source.

In addition, Android will completely ignore the `controls` attribute, and you'll have to either implement your own controls via the JavaScript API (which is the subject of Chapter 5), or to achieve autoplay, play the video via the API on page load.

Android also won't show the first frame of the video as an image; it instead displays a video icon. It does however recognise and understand the `poster` attribute, so if you specify an image with that attribute, it will display that image correctly.

The code for specifying a video for Android follows, along with the JavaScript required to play the video on Android when the user presses the video icon. I'll defer the explanation of this JavaScript for now but will discuss it in detail in Chapter 5:



TIP: Peter Gasston, a web developer and author of many articles and a book on CSS3, provides an in-depth tutorial on how to make HTML5 video work on Android phones. Be sure to check out what he has to say at www.broken-links.com/2010/07/08/making-html5-video-work-on-android-phones if you have problems working with HTML5 video on Android phones.

```
<!DOCTYPE html>
<html>
<head>
    <title>Playing a Video File: Media Query Android</title>
    <script>
        function play() {
            var video = document.getElementById('video');
            video.addEventListener('click',function(){
                video.play();
            },false);
        }
    </script>
</head>
<body onload="play();">
    <video controls>
        <source src="snowy-tree-small.mp4" media="screen and
        → (max-width:800px)">
        <source src="snowy-tree-large.mp4" type="video/mp4">
    </video>
</body>
</html>
```

The preceding issues mentioned only affect the default browser that comes with Android. If the user uses another mobile browser, such as Opera Mobile, the preceding code isn't necessary. It's a good bet that Google will improve Android's implementation of HTML5 video in a future release. And by the time you read this, it may have already been updated.

WRAPPING **UP**

You should now be aware of just how easy it is to add video to your website using HTML5 native multimedia. You are probably also aware that there are still a number of details you need to consider before doing so. For example:

- Which browsers should you support?
- Will you support legacy browsers?
- Should you support mobile devices? If so, which ones?

These are just some of the questions you may need to ask before forging ahead.

Once you've made those decisions, however, harnessing the power of HTML5 multimedia to deliver video to your users is relatively simple. With the standardised method of delivery the multimedia part of the HTML5 specification brings, you know which viewers you'll reach and what kind of experience they will have.

So far you've been leaving it up to the browser to provide the video (and audio) controls. And these controls vary from one browser to the next.

In the next chapter you'll learn how to use the HTML5 multimedia JavaScript APIs included as part of the HTML5 specification to create your own custom controls for both audio and video. Let's go!

This page intentionally left blank

INDEX

NUMBERS

SYMBOLS

A

B

C

D

E

K

L

M

S

W

X