

# Introduction to HTML5 Video

**Instructor: Frank Stepanski**

## History of Multimedia on the Web

When browsers and the idea of the web first appeared back in the early 1990s, there wasn't any web multimedia. Soon thereafter, images began to be incorporated but were at best a poor man's multimedia. Although they could be animated with the advent of animated GIFs, they were of course completely non-interactive.

Even with the existence of audio and video files, the ability of internet technology to deliver this multimedia across the web was limited. Internet connections were slow, audio and video files were large, and no one wanted to wait for large files to download. Once the file did arrive, an external player had to be used to view the contents, which was separate from the web browser. This was the norm, and few complained.

The phenomenal increase in internet connection speeds brought with it the ability to send multimedia across the web even faster, and web browser technology had to move just as quickly, which of course it did with the introduction of what's now referred to as **native multimedia**.

## Media Players

A [media player](#) is a standard term used to describe a piece of software that has the capability of playing back multimedia files, such as audio and video, usually via a graphical user interface.

In the mid-1990s, the MIDI (Musical Instrument Digital Interface) file format was used to play background music on web documents, and the music usually played automatically. Although highly annoying, this was the beginning of multimedia becoming available through the browser.

A host of other players from different vendors were then developed to infest your computer. RealNetworks released its audio player RealAudio back in 1995, which first introduced the idea of playing audio through the web using proprietary .ra and .ram audio files.

Further developments of RealAudio led to the release of RealVideo in 1997, which allowed video streaming and was based on the H.263 video format.

These two players eventually were bundled together under the RealPlayer name and were included in Windows 98 as a selectable tool. [RealPlayer](#) is still around today and is available across many platforms; it is capable of playing multiple audio and video file formats.

Macromedia released its first edition of Shockwave in 1995, which was originally developed for the Netscape web browser. At the time, the company allowed users of its popular Director multimedia software to create interactive applications and animations, and insert them into a web document. Macromedia also had another media player in development; it released the Shockwave Flash player in 2002, which supported video as well as audio.

Shockwave Flash player used the SWF file format developed by FutureWave software. The main intent of the SWF file format was to create small files for displaying animations, as well as to use it to exchange audio, video, and data.

Macromedia was bought by Adobe in 2005, and the Shockwave Flash player was renamed Adobe Flash. It is this particular player that became the player to use when delivering multimedia through the browser.

Microsoft improved on its previous offerings and released DirectShow in 1996, which later became Media Player 6.1 and was released as part of Windows 98. With the release of Windows 2000, Microsoft also released version 7.1 of Windows Media Player. This had a much improved graphical interface and overall offered a better experience.

With this release came [codecs](#), a special file protocols that allow for creating and playing back media content.

With subsequent releases, Microsoft enhanced its [Media Player](#), continuously improving the offering along the way.

Microsoft also worked on an application framework to rival Adobe Flash in that it could run browser plugins (sets of small software components that add specific abilities to other, larger software applications—in this case a web browser) and other rich internet applications.

Initially released in 2007, Microsoft Silverlight supported many different types of audio and video file formats, as well as animation and interactivity. Originally, it had a slow adoption rate, but as of June 2011, it had roughly 73 percent market penetration, with Adobe Flash holding at 97 percent market share.

Apple was also in the multimedia player game from the early days. Its first version of QuickTime was released in 1991 for the System Software 6 operating system.

[QuickTime](#) continued to be a Mac-only piece of software until the release of QuickTime 4.0 in 1999, which also ran on Windows and supported MP3 audio playback. QuickTime 5 and 6 delivered video and Flash playback, with version 6 also supporting MP4 playback.

Release 7 came out in 2005 and had improved MP4 playback but dropped support for Flash content. The latest version of QuickTime has versions for Mac OS and the latest versions of Microsoft Windows.

Every vendor had its own way of delivering multimedia, but of course none were perfect. There was no standardized method of embedding video into a web document, and it is for this reason, to standardize embedding of multimedia that the audio and video elements became part of the [HTML5 specification](#).

In addition, different plugins from different vendors could conflict with each other, causing browser instability.

Security was also an issue, because plugins were and still are one of the main target areas for malware. An example of this was identified previously when it was determined that a critical vulnerability in Flash Player 10.2 could allow remote attackers to execute arbitrary code or cause a denial of service attack on the infected machine.

Of course as soon as these vulnerabilities are detected, the vendors move quickly to patch them and push out a release. But again this means that the user must constantly upgrade to the latest versions when they become available.

One major issue with the most popular plugin of them all, Flash, is [Apple's April 2010 decision](#) not to support Flash on iPhones and iPads. This of course makes content served in Flash completely unavailable on these devices.

## **Native Multimedia**

With native multimedia, the object is no longer embedded into the web document but is treated as a native object by the browser and therefore built in.

This provides immediate benefits, such as:

- Plugins are no longer required.
- Speed - anything native to a browser will be faster than any third-party add-on.
- Native controls are provided by the browser.
- Keyboard accessibility is built in automatically

## **Native Multimedia: The <video> Element**

The HTML5 video element is a way of embedding playable media files directly into a web page without having to use Flash or a plug-in.

The elements can be styled with CSS, integrated with SVG and Canvas, and controlled with JavaScript (covered in a later lesson).

Browsers and other user agents that implement the HTML5 media elements also provide default controls and behavior for each. Support for the media elements is relatively broad, though not all features of the media elements are supported in all browsers.

The video element allows you to embed video content into your web document, and once again you can specify a number of attributes to control this video content.

```
<video poster = "mov_bbb.jpg" src=" ../videos/mov_bbb.mp4"
width="320" height="240" controls autoplay preload></video>
```

The **controls** attribute is a boolean attribute. It's either there or it's not. If it is there, then the browser will add its built-in controls to the video display. This varies by browser, so check out each browser to see what they look like.

The **src** attribute is just like the `<img>` element's `src`—it is a URL that tells the video element where to find the source file. In this case, the source is `../videos/mov_bbb.mp4`.

The **autoplay** boolean attribute tells the browser to start playing the video as soon as it has enough data.

The boolean attribute **preload** is typically used for fine-grained control over how video loads for optimization purposes. Most of the time, the browser chooses how much video to load, based on things like whether autoplay is set and the user's bandwidth. You can override this by setting `preload` to `none` (none of the video is downloaded until the user "plays"), `metadata` (the video metadata is downloaded, but no video content), and `auto` to let the browser make the decision.

The browser will typically display one frame of the video as a **poster** image to represent the video. If you remove the `autoplay` attribute you'll see this image displayed before you click play. It's up to the browser to pick which frame to show; often, the browser will just show the first frame of the video which is often black. If you want to show a specific image, then it's up to you to create an image to display, and specify it by using the `poster` attribute.

The **width** and **height** attributes set the width and height of the video display area (also known as the "viewport"). If you specify a poster, the poster image will be scaled to the width and height you specify. The video will also be scaled, but will maintain its aspect ratio (e.g., 4:3 or 16:9) so if there's extra room on the sides, or the top and bottom, the video will be letter-boxed or pillar-boxed to fit into the display area size. You should try to match the native dimensions of the video if you want the best performance (so the browser doesn't have to scale in real time).

The look and feel of your controls is dictated by those who implement the browsers. They do tend to look different in different browsers and operating systems (Figure 1). In some cases, for instance, on a tablet, they have to look and behave differently because the device just works differently.

**Reference:**

[http://html5video.javascriptworkshop.com/Lesson1/basic\\_player.html](http://html5video.javascriptworkshop.com/Lesson1/basic_player.html)



**Figure 1 - Default player controls (Internet Explorer, Chrome, and Firefox)**

If you don't want the default control UI (each browser has its own), leave the attribute off.

Web developers wanting to provide custom controls remove the controls attribute so that the default control doesn't conflict with the custom control. The developer typically adds the controls attribute to the video element, and then removes it using script as soon as the media element is loaded.

**Note:** This form of [progressive enhancement](#) ensures that if scripting is disabled, the user can still play the media resource, which will be covered in a later lesson.

Attribute	Description
src	Provides the address of the media resource, in this case a video file or stream. This attribute must contain a valid URL to the video source.
preload	Is used to hint to the browser how to preload the video source, if at all. Again, it's only a hint to the browser, which may choose to override it with a possible user setting. It has three possible settings: <ul style="list-style-type: none"><li>• <b>none:</b> Informs the browser that the user is not expected to want to view the video, so there's no need to perform any preloading.</li><li>• <b>metadata</b> Informs the browser that the user may want to view the video, and therefore its metadata (e.g., dimensions, duration) should be preloaded.</li><li>• <b>auto</b> Tells the browser to decide whether to download the metadata, the entire source, or not to download anything. This is the default if preload is not specified.</li></ul>

autoplay	Is a boolean attribute that informs the browser to automatically start playing the video source as soon as it can. It's not considered good practice to do this!
controls	Is also a boolean attribute, which if present, tells the browser to provide a default set of controls for the video.
loop	Another boolean attribute, which indicates that the video is to be continuously played in a loop.
poster	Allows an image file to be displayed when there's no video data available. If poster is not specified, the browser will display the first frame from the video, which may of course not be the image that you want shown. It must contain a valid URL to the image in question.
width	Specifies the width of the video element in pixels.
height	Specifies the height of the video element in pixels.

**Reference:**

<http://www.whatwg.org/specs/web-apps/current-work/multipage/the-video-element.html#video>

**Native Multimedia: video formats (.mp4, .webm and .ogv)**

While the browser makers have agreed on what the <video> element and API look like in HTML5, not everyone can agree on the actual format of the video files themselves. For instance if you are on Safari, [H.264](#) format is favored, if you're on Chrome, [WebM](#) is favored, and so on.

For Google Chrome, use the .webm format by replacing your src attribute with:

```
src=" ../videos/mov_bbb.webm"
```

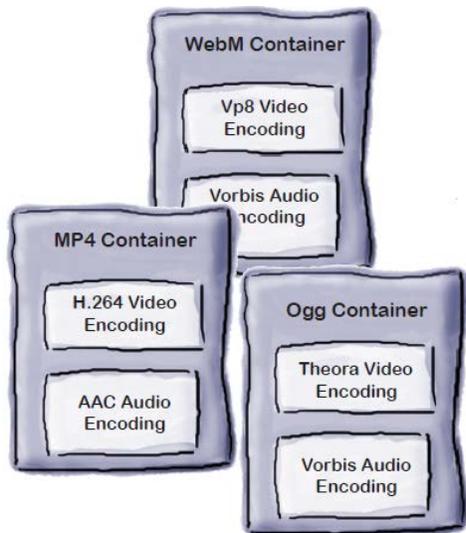
If you're using Firefox or Opera, then replace your src attribute with:

```
src=" ../videos/mov_bbb.ogv"
```

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:** If you're using IE8 or earlier (though HTML5 video support on IE9 can be buggy), you will need a fallback which usually entails using Adobe Flash player, which we will discuss later in this lesson.

## Native Multimedia: Video Formats



We wish everything was as neat and tidy as the video element and its attributes, but as it turns out, video formats are a bit of a mess on the Web.

What's a video format?

a video file contains two parts, a video part and an audio part, and each part is encoded (to reduce size and to allow it to be played back more efficiently) using a specific encoding type.

That encoding, for the most part, is what no one can agree on—some browser makers are enamored with H.264 encodings, others really like VP8, and yet others like the open source alternative, Theora.

## THEORA OGG

[Theora Ogg](#), is also from the Xiph.Org Foundation ([www.xiph.org](http://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.

**Note:** Theora OGG is only used for older versions of Firefox, Chrome and Opera so its importance is decreasing as newer versions of these browsers are released.

### Ogg container with Theora Video and Vorbis Audio

Theora is an open source codec.

Video encoded with Theora is usually contained in an Ogg file, with the .ogv file extension.

Ogg/Theora is supported by Firefox, Chrome and Opera.

## 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](http://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.

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

### MP4 container with H.264 Video and AAC Audio

H.264 is licensed by the MPEG-LA group.

There is more than one kind of H.264; each is known as a “profile.”

MP4/H.264 is supported by Safari and IE9+. You may find support in some versions of Chrome.

## WEBM

WebM is a project ([www.webmproject.org](http://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.

### WebM container with VP8 Video and Vorbis Audio

WebM was designed by Google to work with VP8 encoded videos.

WebM/VP8 is supported by Firefox, Chrome and Opera.

You'll find WebM formatted videos with the .webm extension.

**Note:** Android phones also support MP4 (and WebM from version 2.3), Windows Phone 7 and 8 has native support for both MP4 and WebM, and Blackberry supports MP4 from version 7 of its browser.

**Reference:** <http://www.longtailvideo.com/html5>

## Native Multimedia: Playing video with different sources

You can't assume your web page readers have plug-ins installed to play Ogg or WebM videos in Safari or IE. In order to ensure that a video is accessible by all of the target browsers, you'll need to provide, at minimum, two different (preferably three) source elements for your video element.

The video elements provide a src attribute in which to place a URL to the audio or video source.

But that attribute only allows for one source. So how can you add multiple sources?

This is where the source element comes in. Any number of source elements can be contained within a video element declaration, and it is through this element that you can specify multiple media sources.

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

## Native Multimedia: Being More Specific with Video Formats

Telling the browser the location of your source files gives it a selection of different versions to choose from, however the browser has to do some detective work before it can truly determine if a file is playable. You can help your browser even more by giving it more information about the MIME type and (optionally) codecs of your video files.

The browsers will parse the @type attribute and use it as a hint to determine if they can play the file. MIME types do not always provide a full description of the media resource. Thus, based on the value of the @type attribute, the browser will guess whether it may be able to play the media resource.

```
<video src="../videos/mov_bbb.mp4" type='video/mp4;
codecs="avc1.42E01E, mp4a.40"'></video>
```

```
<video src="../videos/mov_bbb.webm" type='video/webm;
codecs="vp8, vorbis"'></video>
```

```
<video src="../../videos/mov_bbb.ogv" type='video/ogg; codecs="theora, vorbis"'></video>
```

**Note:** When you need to frame multiple parameters with double quotes and thus you have to put the @type value in single quotes or otherwise escape the double quotes around the @type attribute value.

**Reference:** [http://wiki.whatwg.org/wiki/Video\\_type\\_parameters](http://wiki.whatwg.org/wiki/Video_type_parameters)

## Native Multimedia: MIME type

A [MIME type](#) (sometimes called a content type) is a piece of information that identifies the type of content in a web resource. For example, the MIME type of a web page is *text/html*.

Before a web server sends a resource to a browser, it sends the MIME type. For example, if a browser asks for the page *SuperVideoPlayerPage.html*, the web server sends the *text/html* MIME type, a few other pieces of information, and the actual file content. When the browser receives the MIME type, it knows what to do with the content that comes next.

It doesn't need to try to make a guess based on a file name extension or some other sort of hackery.

## Native Multimedia: Codec

A video codec is software that compresses the video for your desired needs, and as you've just learned with all of the previous video properties, all of them are configurable using video certain codecs.

When talking about media file types, we're really talking about two separate components:

- the software used to encode and decode the audio or video stream: *codec*, which is short for *compressor-decompressor* or *coder-decoder*
- and the *container*, which is a wrapper format that contains the media streams and information about how the data and metadata co-exist.

Video files typically wrap two different media streams: the video and the audio data streams.

## Native Multimedia: Flash Fallback

For extra compatibility with less recent browsers, let's put the old-style Flash embedding code inside the <video>element as fallback, so that if the browser doesn't know anything about the beauty of HTML5 native video, it defaults to the Flash fallback content inside the tags.

```
<video poster = "mov_bbb.jpg" width="320" height="240" controls>
  <source src="../videos/mov_bbb.mp4" type="video/mp4">
  <source src="../videos/mov_bbb.webm" type="video/webm">
  <source src="../videos/mov_bbb.ogv" type="video/ogg">

  <object type="application/x-shockwave-flash" data="player.swf"
    width="320" height="240">
    <param name="allowscriptaccess" value="always">
    <param name="flashvars" value="file=="mov_bbb.mp4">
    <!--[if IE]><param name="movie" value="player.swf"><![endif]-->
    
    <p>Your browser can't play HTML5 video.
    <a href="mov_bbb.mp4">Download it</a></p>

</object>
</video>
```

### Reference:

[http://html5video.javascriptworkshop.com/Lesson1/basic\\_player\\_flash\\_fallback.html](http://html5video.javascriptworkshop.com/Lesson1/basic_player_flash_fallback.html)



Figure 2 – Flash fallback player (Internet Explorer 8 or older)

Anything stated inside the <video> element that is not inside one of the specific child elements of the <video> element is regarded as “fallback content”. It’s “fallback” in so far as web browsers that do not support the HTML5 <video> element will ignore these elements, but still display their contents and thus is a means to be backwards compatible.

Flash supports playback of H.264 video, so all we need is our ready-made Flash player (inside the same folder), and we’ll pass it the URL for the H.264 version of our video as a parameter.

It’s important to note that the path to the MP4 needs to be either absolute or relative to the location of the SWF file – for simplicity, we’ve placed the player and the videos in the same directory

## Native Multimedia: Encoding

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

- **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.
- **VLC Player** (<http://www.videolan.org/vlc/>). This open source video player can play all video formats as well as encode them.
- **Freemake** (<http://www.freemake.com/>). Open source converter for all video formats for Windows computers.

## 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.

## **Native Multimedia: Aspect Ratio**

Much like images, videos have a width property and a height property. In video, much like images, it's important to maintain the proportion of the original dimensions. In video, *aspect ratios* are the proportional relationship between its width and its height.

Most common aspect ratios in web video are 16:9 for wide-screen format and 4:3 for standard format. This proportional relationship is directly tied to the size of the pixels within the video, because they can be square or rectangle. You will often hear this being referenced as “sixteen by nine” or “four by three.”

It's important to preserve the aspect ratio of the video you are transcoding because this keeps a one-to-one relationship between the source video asset and the transcoded file. However, in some cases, your creative will require you to serve your video asset into size not designed in 16:9 or 4:3.

In this case, you will use a technique called letterboxing or pillarboxing, which adds black bars where there is empty space within the allotted video space.

## **Importance of Video in Online Marketing**

As the digital landscape evolves and mainstream media transforms, marketers are developing new strategies for engaging with consumers

Online video plays a central role in the new marketing ecosystem. Consumers love it, seek it out, and share it actively across their networks—everything from their favorite ads to longer-form entertainment, informational, and instructional content.

A uniquely vivid and compelling medium, video has an unmatched ability to tell stories, convey information, and evoke an emotional response.

The growth of online video and social media in [content marketing](#) is no surprise. At every stage of the customer lifecycle, online video and social media can make paid, owned, and earned media strategies more tightly integrated, more symbiotic, and more impactful.

Social media provides a powerful multiplier effect for the reach generated by online video. YouTube isn't only a social network around video; it's also the second-largest search engine on the web, attracting 157 million unique viewers per month.

Compelling video not only attracts visitors to a website; it can also help keep them there. Vivid storytelling and deep information focus viewers' attention and entice them to watch additional videos. As their time on site grows, they remain within your branded environment and available for further messaging and offers.

Half of interactive marketers expect online video to increase in effectiveness over the next three years which is more than those who expect an increase in the effectiveness of other interactive marketing tactics like email marketing, SEO, and display advertising.

Online video content can be used as a product brochure, demonstration, destination site, or game, among other applications, depending on a marketer's need. It also can enable visitor engagement in ways that a traditional ad can't. For instance, users can click on in-stream annotations, download or share videos, and post comments or video responses through YouTube or other video communities.



## Challenges of a Mobile Environment

There are known quirks about the use of the HTML5 media elements in mobile devices. For instance, Apple has been a big fan of HTML5 from the beginning, deciding against support for Flash on iOS devices in favor of HTML5 video.

However, some things that work on the desktop that may not in an Apple mobile environment.

With every new version and update to iOS, these “quirks” are usually resolved, but some linger and some are even replaced with others.

In addition, the iOS environment has its own native application for playback control, so it ignores the controls attribute.

Most importantly, the video capability itself is limited in mobile environments. There is the resolution/size issue, of course, but there are also issues with containers and codecs.

Mobile devices don’t have the processing power our computers have, which means that the file sizes are larger (because of simpler compression techniques). At the same time, mobile devices have data access limitations as well as issues with storage, so larger files aren’t mobile-friendly.

There’s also the challenge associated with the sheer number of mobile operating systems, mobile browsers, and devices—especially devices.

You’ll also need to provide appropriately sized video files, as well as include support for both WebM/ Ogg Theora and H.264.

### **Android: Reference:**

<http://developer.android.com/guide/appendix/media-formats.html>

### **iOS Reference:**

<https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/MediaLayer/MediaLayer.html>

Copyright 2014 © Frank Stepanski

Lessons, files and content of these classes cannot be reproduced and/or published without the express written consent of the author.