

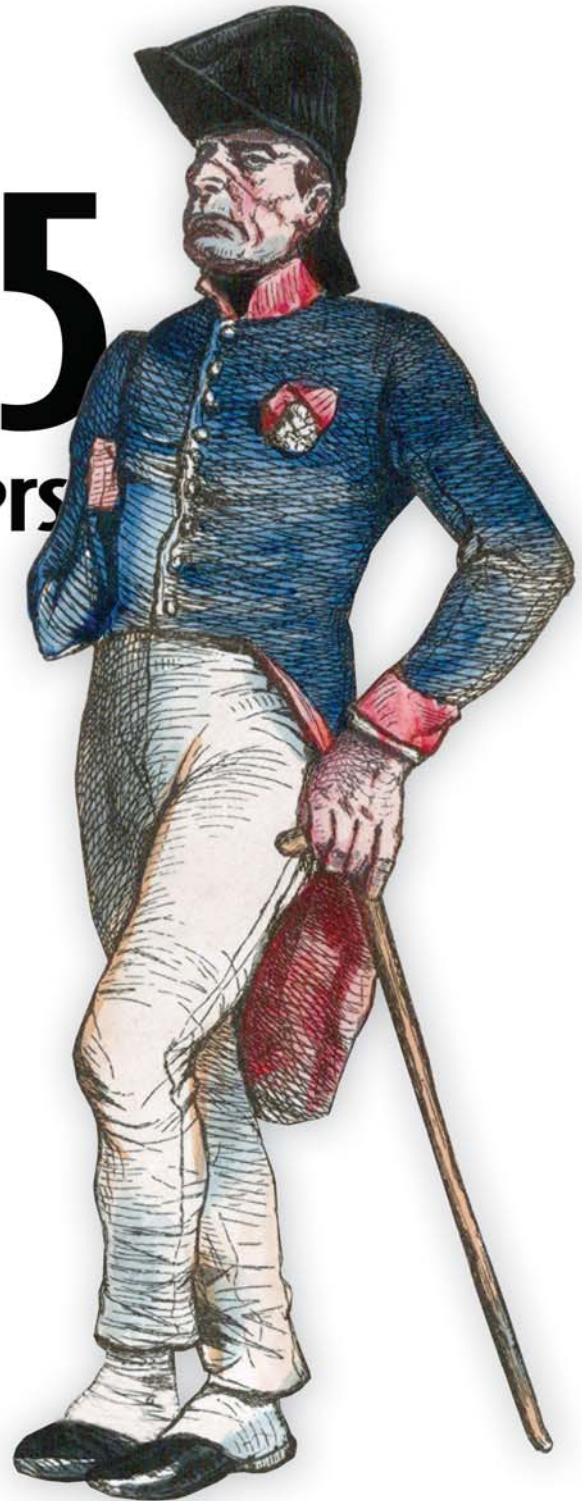
Single page web apps, JavaScript, and semantic markup

# HTML5

## for .NET Developers

Jim Jackson II  
Ian Gilman

FOREWORD BY  
Scott Hanselman





# ***HTML5 for .NET Developers***

by Jim Jackson II  
Ian Gilman

## **Chapter 3**

Copyright 2013 Manning Publications

## *brief contents*

---

- 1 ■ HTML5 and .NET 1
- 2 ■ A markup primer: classic HTML, semantic HTML, and CSS 33
- 3 ■ Audio and video controls 66
- 4 ■ Canvas 90
- 5 ■ The History API: Changing the game for MVC sites 118
- 6 ■ Geolocation and web mapping 147
- 7 ■ Web workers and drag and drop 185
- 8 ■ Websockets 214
- 9 ■ Local storage and state management 248
- 10 ■ Offline web applications 273

# Audio and video controls

---

## ***This chapter covers***

- Using audio and video controls with no code
- Integrating JavaScript controls with audio and video
- Simple binding techniques for controlling audio and video
- Understanding audio and video formats

It wasn't so long ago that the only way to play video content was to embed a QuickTime, Flash, Silverlight, or other custom-installed program inside your HTML page with `<object>` tags. These elements had very little interactivity with the surrounding page and were, for all intents, islands of media on the page. Audio content was only a little better and, in some cases, worse. When was the last time you visited a page that played a song in the background? From the user's perspective, it's the height of annoyance that you can't do anything with that page other than turn down your computer's volume or navigate away. This chapter will show you how to fix all those problems with HTML5.

HTML5 brings two new tags to the table: `<audio>` and `<video>`. Both of these tags implement the same API interface, so while the internal implementations are

## Browser support

Audio & video tag/API  
(Desktop)



Audio & video tag/API  
(Mobile)




Audio

Video

## Chapter 3 map

Audio and video tags allow the browser to load and play both audio and video content without the need for plugin frameworks. Each browser vendor controls which media formats it will support, and the page developer can specify multiple levels of fallback content to play if a particular format isn't supported.

Identifying <code>&lt;audio&gt;</code> and <code>&lt;video&gt;</code> tags	page 72
Using the <code>controls</code> attribute	page 72
Using the <code>autoplay</code> attribute	page 72
Repeating content with the <code>loop</code> attribute	page 72
Queueing content with the <code>preload</code> attribute	page 72
Using <code>&lt;video&gt;</code> and <code>&lt;audio&gt;</code> tags without code	page 74
Learning to use <code>HTMLMediaElement</code> in JavaScript	page 76
Playing content with JavaScript	page 79
Pausing content	page 79
Handling volume changes	page 80

Look for this icon  in this chapter and throughout the book to quickly identify discussions of key HTML5 functionality.

different, the external interfaces are identical. Furthermore, neither tag requires any additional supporting downloads to work in supported browsers. You just add the tag, supply the source content, and your users can see and hear your media. It really is that simple!

You'll also be able to go even farther, as you'll see in this chapter's sample application. This example starts with basic operational features that help you understand how the `<audio>` and `<video>` tags and their JavaScript APIs work, and evolves over the course of the chapter. By the end, it will be able to do the following:

- Download and play audio and video content
- Use HTML objects to control content playback

- Play and pause content using JavaScript
- Change volume and mute media content

When the example is complete, you'll have a working knowledge of the new HTML5 audio and video controls and will be able to start building such content into any existing web application.

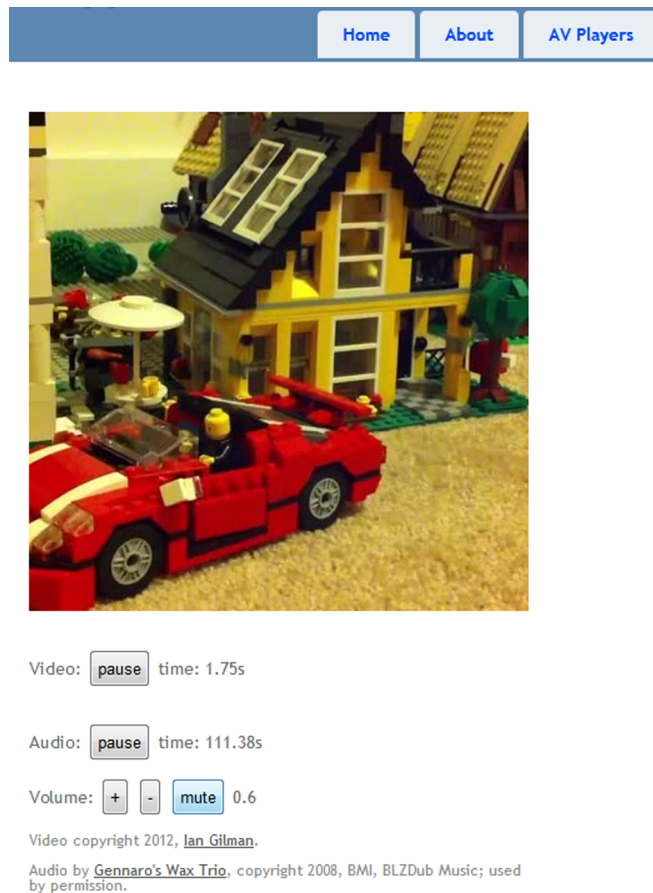
As you work through the project, you'll learn how to do several things:

- Use the audio and video tags without JavaScript, as HTML elements
- Control audio and video playback with JavaScript
- Update media types for open source content

Before we get to those topics, let's look at what you'll be building and walk through the steps involved in getting the application started.

### 3.1 Building a site to play audio and video

Figure 3.1 shows exactly what the site should look like in any compatible browser.



**Figure 3.1** The sample site will play audio and video content without plugins or extra downloads.

To get started building the application skeleton, perform the following steps:

- 1 Open Visual Studio and create a new ASP.NET MVC application.
- 2 Select Internet Application, Razor View Engine, and HTML5 semantic markup.
- 3 Leave the Create Unit Test check box unchecked.
- 4 Name the application AudioVideo.
- 5 When Visual Studio starts, navigate to Tools > Library Package Manager > Manage NuGet Packages for Solution.
- 6 Select the Installed Packages tab on the left, and then click Manage on the following packages:
  - Entity Framework
  - jQuery UI
  - jQuery Validation
- 7 When the pop-up window appears, deselect the project and click OK to remove the package from your solution.
- 8 Select the Updates tab on the left and click the Update button for each package remaining in the center of the window.
- 9 Open the Razor View file located at `\Views\Shared\_Layout.cshtml` in the solution, and update the script tags at the top to match the newly updated scripts in your solution's Scripts folder.
- 10 In the menu area, add a new list item:

```
<li>@Html.ActionLink("AV Players", "Players", "Home")</li>
```

This creates a link that points to `/Home/Players` when the application runs. It won't work yet because that endpoint and its associated view don't exist. That's the next setup step.

- 11 Navigate to the Controllers folder and open the Home Controller. Add the following snippet of code to create the new endpoint:

```
public ActionResult Players()
{
    return View();
}
```

- 12 Right-click on the word View, and from the pop-up menu select Add View. This will create a new file called `Players.cshtml` in the `\Views\Home` folder of your solution.
- 13 If you have not done so already, get the audio and video content from the GitHub account and add these four files to your solution's Content folder:
  - gwt.ogg
  - gwt.mp3
  - lego.ogv
  - lego.mp4

The `.ogg` and `.ogv` files are open source audio and video formats used for high quality digital media. The format is maintained by the Xiph.org Foundation

(<http://www.xiph.org/>). The .mp3 and .mp4 formats are proprietary but very common.

### Finding the audio/video content for the sample application

You can download the audio and video content we use in this chapter from <https://github.com/axshon/HTML-5-Ellipse-Tours/tree/master/demos/av/Media>.

This GitHub project was originally designed as a monolithic sample application for this book, but we decided that the application-per-chapter paradigm would be more suited to introducing and explaining each HTML5 API in isolation. All the code in this book and some additional content are available in the GitHub repository.

### Converting audio and video file formats

There are lots of options for converting your audio and video to Ogg or WebM, but right now the easiest is the free Miro Video Converter (<http://www.mirovideoconverter.com/>). Once you've installed it, just drag your audio or video file into it and pick a format.

For Ogg video (.ogv), select Theora, an open video format ([theora.org](http://theora.org)). The Theora setting also converts audio files into Ogg audio, but you'll want to change the resulting file extension to .ogg.

Table 3.1 shows the current browser compatibility levels for the .ogg and .ogv formats.

**Table 3.1** Open source media format compatibility (.ogg and .ogv)

Browser	Starting version support
Chrome	4
Internet Explorer	Not supported
Firefox	3.5
Opera (desktop)	10.5
Safari (all versions)	Not supported
Opera Mobile	11
Android (all versions)	Not supported
Windows Mobile	Not supported

Now that the application skeleton is in place, we'll look next at the <audio> and <video> tags and their basic similarities, and at three different ways that you can use the tags in your projects. You'll use one of those ways to build this chapter's sample application.

## 3.2 Audio and video tags

`<audio>` and `<video>` tags in HTML5 are similar in that they both implement the `HTMLMediaElement` interface. This interface describes all the major functions, properties, and events necessary to play and control multimedia content on the web. You'll see a few of these events and properties in action later when you build out the JavaScript side of the player application, but you can get the full story on the W3C site at <http://dev.w3.org/html5/spec/media-elements.html#htmlmediaelement>.

The simplest tag that you can implement for rendering audio and video content in HTML5 plays whatever audio content is in the audio file, as long as it exists in the relative URL specified in the `src` attribute. It looks like this:

```
<audio src="myaudio.mp3"></audio>
```

The tags become much more powerful when you implement the `<source>` child tags. These let you provide multiple formats for your content so it's more likely to be playable across different browser platforms. Here's an example:

```
<audio>
  <source src="myaudio.ogg" type="audio/ogg" >
  <source src="myaudio.mp3" type="audio/mp3" >
</audio>
```

The `<source>` tag is the same for both the `<audio>` and `<video>` tags. You just need to add as many elements as you have available for the specific piece you want the user to see or hear, and the browser will start from the top and work its way down until it finds a format that it supports. No checking is done to detect bandwidth, video size, or any other information about the source media file. If the format is supported, the browser will use it. End of story.

There is also no concept of “more supported” when it comes to playing source files. A particular format either is or isn't supported. As you might infer from table 3.1, there is no format for either audio or video that's universally supported—not even mp3—so for now you must put on your format-converter hat and translate whatever you have into formats supported by your target user's browser.

**NOTE** We decided not to give the full story around which formats are supported by which browsers because the format “wars” are still ongoing, with no clear winner so far. What is supported on which browser will likely continue to change rapidly. Expect to have at least two or three formats for each media file you present to your users.

### ASSIGNING HTML ATTRIBUTES TO TAGS

Using an audio or video tag as in the previous snippets isn't enough to play your media content, because you haven't yet assigned the appropriate HTML attributes to the tags to run the audio content. Audio and video tags can operate in one of three ways:

- Strictly as HTML elements
- Strictly as JavaScript controls
- As a hybrid using both HTML attributes and JavaScript to control playback

In the next few pages, we'll describe the ins and outs of presenting media content using the first method, and then you'll continue building the sample application using the second method.

Because the third (hybrid) method just combines the first two, everything you learn as we proceed should equip you to build out using that method as well. A typical situation where you might use the hybrid method is on a video site that automatically starts playing as soon as content is ready but that allows a user to stop and restart using JavaScript code.

### 3.2.1 Using audio and video tags without JavaScript

Core API



Let's start with the first method: using the tags strictly as HTML elements. You might want to do this when you're building a static site displaying a tutorial or a web page that shows an introductory audio or video clip for a site.

To do this, you can just add the `controls` attribute to an `<audio>` tag and refresh the page:

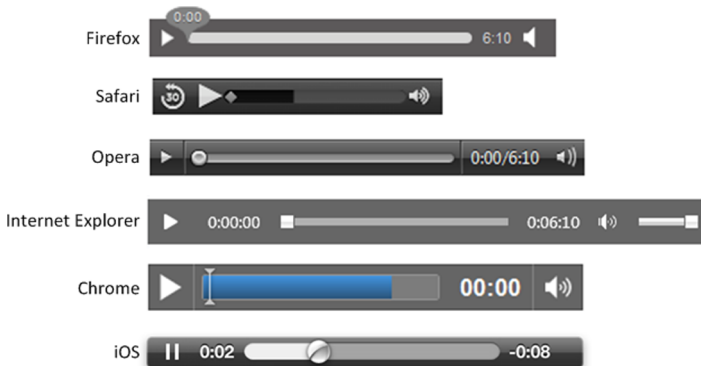
```
<audio id="audio" controls>
  <source src="myaudio.ogg" type="audio/ogg" >
  <source src="myaudio.mp3" type="audio/mp3" >
</audio>
```

You should see one of the various audio player formats built into whatever browser you run. The selection of screen shots in figure 3.2 may change over time as browsers are upgraded, but we expect the changes to be minimal.

The other common `<audio>` attributes are listed here; values for each are in the discussion that follows:

```
<audio id="audio"
  controls
  autoplay
  loop
  preload="metadata or none or auto">
```

Let's look at these attributes in a little more detail.



**Figure 3.2** The default audio players for browsers vary in height and width. If layout is a concern, consider creating your own player interface. Section 3.3 shows you how to do that.

### THE CONTROLS ATTRIBUTE



The controls attribute just needs to be present and doesn't need a value assigned. You may also see it listed as `controls="controls"`, which is the same thing.

The attribute's function, as you saw earlier, is to show the user audio controls. The rendering and function of these are entirely dependent upon what the browser vendor wants to give you. They can be always visible or only visible on hover, depending upon the browser, but this facet of their operation isn't something you as the developer can affect.

### THE AUTOPLAY ATTRIBUTE



The autoplay attribute doesn't need a value set and will start playing the audio content as soon as the element is loaded. This harkens back to the early days of audio content on the internet, when sites would play background music for you while you browsed the site. This is practically never a good idea because it annoys users and uses bandwidth unnecessarily. It does have its place in site design and content presentation, but remember: less is more.

### THE LOOP ATTRIBUTE



The loop attribute will start the audio again after it has completed playing. This can be handy in game development for soundtracks, but again, it can easily be overused. `autoplay` will continue to play the audio track until the user unloads the page or pauses it. If controls aren't displayed on a track of audio content, and you have `loop` turned on, you can guarantee that visitors will leave your site as quickly as they can. `loop` is also an attribute that doesn't need a value set.

### THE PRELOAD ATTRIBUTE



The preload attribute has three possible values: `metadata`, `auto`, and `none`:

- `metadata`—Requests that the player download enough information about the audio track to show the total length of the track and possibly other information, depending upon the browser vendor's implementation
- `none`—Tells the browser to download nothing until the user presses the play button
- `auto`—Attempts to start loading the track as soon as the element is rendered on the page

Keep in mind that these preload settings are *suggestions* for the browser. The browser may choose, for whatever reason, to ignore these settings when downloading audio or video content for the element.

Now that you have a grasp of how a generic HTML5 media tag works, let's spend a moment focusing on the `<audio>` tag.

## 3.2.2 Using the audio tag as an HTML element

If you worked through the markup in the `Players.cshtml` page you created in section 3.1, you should have seen the players appear on the page when you ran the application in a browser and, depending upon the attributes you assigned, it may have started playing or been queued up ready to be started by a user.

### Forcing audio and video elements to show their controls

There may be times when you land on a page with audio or video controls that has controls turned off, and you would like to see them or you want to test your own pages at runtime. If these pages are using the HTML5 tags and jQuery, the easiest way to find the elements and turn controls on is to use the `attr` function:

```
$("#audio").attr("controls", "controls");
$("#video").attr("controls", "controls");
```

This code assumes an audio and video element with IDs of `audio` and `video` respectively, and it will immediately cause the `controls` feature of these elements to be turned on.

The code in the following snippet is a little more detailed and uses the ASP.NET MVC `Url.Content` helper method to build URLs that are relative without parsing HTTP request address strings. Note the order of the `<source>` elements. As mentioned earlier, the browser will always try to play these by starting at the top and working down until it finds a compatible format. It stops there:

```
<audio id="audio">
```

```
  <source src="@Url.Content("~/Content/gwt.ogg")"
    type="audio/ogg" >
```

```
  <source src="@Url.Content("~/Content/gwt.mp3")"
    type="audio/mp3" >
```

```
</audio>
```

Basic audio tag will give code a way to play sounds but won't, by default, display.

Source elements inside media tags will attempt to play in order.

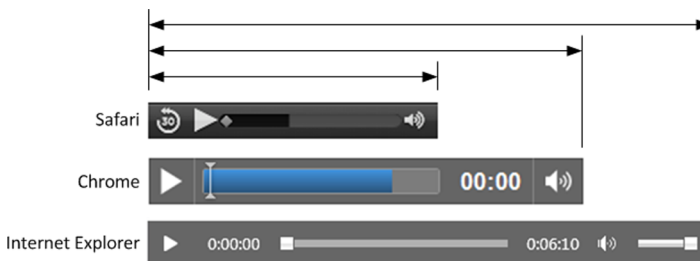
If first source tag is unsupported by current browser, next is tried, and so on.

Be careful when using the browser default players for `<audio>` content, because they can differ greatly in the amount of space they take up on the page. Take another look at the rendered audio players in figure 3.3 and notice how the Safari player is much narrower and the Internet Explorer player is much wider than Chrome. Any of these could change your page layout if not handled properly.

### 3.2.3 Using the video tag as an HTML element



The `<video>` tag in HTML5 is similar to `<audio>` in most of its implementation respects. In fact, anything you can control on an audio track, you can also control on a



**Figure 3.3** The differences between default rendered audio players in various browsers. The Chrome version is the same as the size in most other browsers; Safari and IE fall outside the normal boundaries.

video track. Features, events, and properties are all the same, except that video contains a few more that are inappropriate for audio. With the `<video>` tag, you can assign some additional properties to set the size of the video as well as the image that appears before the video starts to play.

A basic `<video>` tag, like the one in the following snippet, will have source tags nested inside it and use the same format fallback mechanism described earlier. The difference between `<audio>` and `<video>` when rendered is that video will always display unless told not to either in code or by some CSS rule. `<audio>`, on the other hand, won't display if controls isn't turned on:

```
<video id="video">
  <source src="@Url.Content("~/Content/lego.ogv")" type="video/ogg" >
  <source src="@Url.Content("~/Content/lego.mp4")" type="video/mp4" >
</video>
```

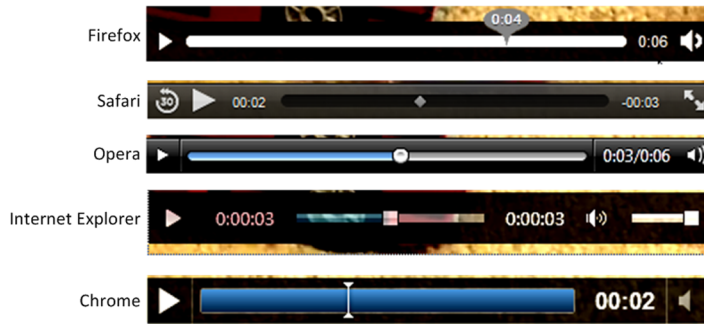
This code will display the opening frame of the content video but no controls to control the play. One exception to this rule is that some browsers, notably Internet Explorer 9 and above, will allow you to right-click on the video and get a context menu with player controls. These can't be turned on or off, but it's possible in code to override the right-click.

Just like the `<audio>` tag, you can add the controls, autoplay, loop, and metadata attributes, and their function is identical. The additional features only available with video are the height and width attributes, which will constrain the video to a specific rectangle, and the poster attribute. poster is a URL value that points to a valid image file. The image will be scaled to fit inside the assigned height and width, but the aspect ratio isn't guaranteed to be retained, unlike video content, which will shrink until the correct aspect fits inside the assigned height and width rectangle:

```
<video id="video"
  controls
  autoplay
  loop
  preload="metadata or none or auto"
  poster="@Url.Content("~/Content/VideoPoster.png")">
```

Figure 3.4 shows the default video player controls implemented by the major browsers at the time of this writing. The controls appear inside the boundaries of the video—some push the video content up, some appear over it with a slight transparency effect, but all are rendered at the bottom of the player's rendered area.

Note that the width of the controls will be the same as the width of the player, but the browser vendor can choose any height for the rendered control surfaces. While this is fine for tangential content that isn't necessarily the core purpose of a particular page, the fact is that if you're building the page specifically to present audio or video content, the default players simply won't do. They're visually inconsistent and offer no customization capabilities. Enter the JavaScript APIs.



**Figure 3.4** The current default video player controls will appear inside the defined height and width properties of the video element.

### 3.3 Controlling audio and video playback with JavaScript



Using the JavaScript API available for `<audio>` and `<video>` elements, you can control nearly every feature of playback in the client browser. You can also wire up events and properties to any other HTML controls so the presentation is entirely up to the site designer. As mentioned earlier, you can also use the JavaScript APIs to control features of playback while leaving the existing browser-provided controls in place.

Here are some things you can do with these two media elements in code:

- Assign source values
- Monitor the state of play
- Get the total duration and current time of the track being played
- Detect and modify the rate of playback
- Assign attributes for loop, autoplay, and controls
- Detect when a track has finished playing
- Turn the volume up or down
- Mute or unmute the volume

As you can see, there are a lot of control features available to you when playing audio and video content, and you're going to use the most common ones in your sample application, such as play/pause, mute/unmute, and volume control. In this section, you'll learn how to use the audio and video APIs as you do the following:

- Build a custom audio and video control surface
- Build the main.js library structure
- Create a JavaScript media player object
- Attach JavaScript to audio and video event models to complete the user's media experience

We'll start with building custom controls.

#### 3.3.1 Building custom controls for audio and video

Before you begin, if you've been following along in your solution and fiddling with `<audio>` and `<video>` tag attributes, clear them all out from the `Players.cshtml` page.

You'll also need to add `id` properties to the tags so that you can easily identify them in code. They should look like this:

```
<audio id="audio">
  <source src="@Url.Content("~/Content/gwt.ogg")" type="audio/ogg" >
  <source src="@Url.Content("~/Content/gwt.mp3")" type="audio/mp3" >
</audio>
<video id="video">
  <source src="@Url.Content("~/Content/lego.ogv")" type="video/ogg" >
  <source src="@Url.Content("~/Content/lego.mp4")" type="video/mp4" >
</video>
```

Next, below the video element add a few standard `<button>`, `<div>`, and `<span>` tags to bind to the control code you'll write shortly. The following listing shows the layout of this markup. These will act as the controlling user interface elements in the final page.

### Listing 3.1 Controls for audio and video elements

```
<div id="video-controls">
  Video:
  <button class="play">play</button>
  <span class="time"></span>
</div>
<div id="audio-controls">
  Audio:
  <button class="play">play</button>
  <span class="time"></span>
  <div class="secondary-controls">
    Volume:
    <button id="volume-up">+</button>
    <button id="volume-down">-</button>
    <button id="mute">mute</button>
    <span id="volume"></span>
  </div>
</div>
```

Annotations for Listing 3.1:

- Play/pause button for video content**: Points to the `<button class="play">play</button>` line in the `video-controls` div.
- Note to show current time of video**: Points to the `<span class="time"></span>` line in the `video-controls` div.
- Play/pause button for audio content**: Points to the `<button class="play">play</button>` line in the `audio-controls` div.
- Note to show current time of audio**: Points to the `<span class="time"></span>` line in the `audio-controls` div.
- Volume and mute controls for audio**: Points to the `<div class="secondary-controls">` block in the `audio-controls` div.
- Display for current audio volume**: Points to the `<span id="volume"></span>` line in the `audio-controls` div.

### Where are all the HTML5 semantic tags?

You may have noticed that in this chapter we're using regular `<div>` and `<span>` tags to organize the structure of our page. We do this in various places throughout the book for a number of reasons.

First, the audio/video content in this chapter is contained in a single page, otherwise known as a single page app (SPA). One of the primary functions of semantic markup is to allow a web crawler, search engine, or accessibility tool to "read" the content of a page, but SPAs generally load content dynamically using JavaScript based on user interaction or other conditions. A web crawler won't execute JavaScript, so it won't be able to load and parse the dynamic content. This makes the semantic tags somewhat useless.

Second, we want to make it abundantly clear throughout the book that while you can use semantic HTML tags right away in all of your web pages and HTML applications, it's optional. The previous (classic HTML) tags are still valid and common throughout the web.

### 3.3.2 Building the main.js library structure

With the controls in place, you can start building the controlling code structures.

Create a new JavaScript library file called `main.js` in the `Scripts` folder of your solution and open it. You'll have only three functions in your `main.js` library, as you can see in the next listing. These will initialize the page, initialize an audio or video object, and update the volume value on the screen.

**Listing 3.2** The basic JavaScript structure of the `main.js` library

```
$(document).ready(function () {
    Main.init();
});

window.Main = {
    init: function () {
    },
    initMedia: function (name) {
    },
    showVolume: function () {
    }
};
```

**Checks for features, creates two objects, and binds volume controls to HTML controls**

**Takes either “audio” or “video” as parameter and builds player object that’s bound to appropriate media element and a few of element’s event handlers**

**Updates volume display on screen**

The basic structure here initializes the `Main` object and creates custom objects via `initMedia` to control playback of either audio or video content. Inside the `init` function, you’ll test for browser compatibility using `Modernizr` and then execute the function to create your objects. That code is shown in the following listing.

**Listing 3.3** `init` function checks and initializes video and audio elements

```
var self = this;

if (!Modernizr.audio) {
    alert("Audio tag not supported.");
    return;
}

if (!Modernizr.video) {
    alert("Video tag not supported.");
}

this.video = this.initMedia("video");
this.audio = this.initMedia("audio");
```

**Checks for audio and video support**

**Creates audio and video objects and attaches them to Main object**

`Modernizr` checks for audio and video compatibility. Then this code adds two properties to the `Main` object created earlier using `window.Main = { ... }`. The properties (`video` and `audio`) are created with the `initMedia` function. Read on to see how and why.

### 3.3.3 Creating a JavaScript media player object

This `initMedia` function is a great example of how you can reduce the volume of code you write and improve maintainability. In this function, you'll find various elements in the interface, the most important of which is either the rendered `<audio>` or `<video>` element. You then treat that element not as a piece of audio or video content, but as a piece of generic content. You can do this because both tags implement the `HTMLMediaElement` interface.

To start, look at the next listing very carefully. It's the first part of `initMedia`. There's a lot of locating of elements and assigning of variables going on here.

**Listing 3.4 The `initMedia` function assigning properties for a new object**

```
var result = {};
result.$media = $("#" + name);
result.media = result.$media[0];
result.$controls = $("#" + name + "-controls");
result.$play = result.$controls.find(".play");
result.$time = result.$controls.find(".time");
```

**Find media element with jQuery and pull actual media element from wrapped set.**

**Find controls <div> based on concatenated naming convention.**

**With controls <div> find play button and time <span> element.**

Core API



Notice that you have `$media`, `$controls`, `$play`, and `$time` all as wrapped sets from jQuery selectors, plus the `media` element that corresponds to either the `<audio>` or `<video>` element based on the input parameter (`name`) value. Why go through all these gyrations? Because a wrapped set will give you all the normal jQuery functionality you need to bind to events, change assigned CSS attributes, and update text values, but it won't give you the ability to call functions on individual API objects. For that, you must have a single object, not a wrapped set.

The next bit of code in the `initMedia` function shows the process of getting a wrapped set and then using the object (not the wrapped set) to execute functions:

```
result.$play.click(function () {
    if (result.media.paused)
        result.media.play();
    else
        result.media.pause();
});
```

**You must have single object to check properties like paused or execute functions like play.**

**Use wrapped set to bind to click event.**

Core API



You've just implemented the `click` handler for the `$play` button, so the `media` (audio or video) will play, but you still have to bind to the various other player events so that you can pause the media and track what's happening while it plays. You can use the `$media` wrapped set for this because you aren't executing specific functions.

Listing 3.5 shows how to bind to the playing, pause, ended, and `timeupdate` events. Again, each time you have to call into a specific function or property of the `HTMLMediaElement` API, either audio or video, you make the call against the `media` local property. This is the last part of the `initMedia` function.

**Listing 3.5** Binding events to the media object in the `initMedia` function

```

result.$media
    .bind("playing", function () {
        result.$play.text("pause");
    })
    .bind("pause", function () {
        result.$play.text("play");
    })
    .bind("ended", function () {
        result.media.play();
    })
    .bind("timeupdate", function () {
        var prettyTime =
            Math.round(result.media.currentTime * 100) / 100;
        result.$time.text("time: " + prettyTime + "s");
    });
result.media.play();
return result;

```

There's a lot of interplay here with the audio and video elements, the controls on the page, and the Main object's various properties. To reiterate, the secret sauce that makes this event binding with jQuery wrapped sets work is the fact that audio and video elements implement the `HTMLMediaElement` interface, making them generally function the same way as each other but with different output to the browser. Figure 3.5 shows the various wrapped sets and properties that you established in your code and how they all play together in the `window.Main` object.

Coming full circle to the object you created in `initMedia`, you first instance an object variable called `result` and then bind a bunch of jQuery wrapped sets and a media object to it. Then you bind the various media events to create a responsive interface. Finally, you start the media content playing and then return the object to the caller, which in this case is the `init` function. It seems complicated, but you're really just setting up the interface and playing the media.

### 3.3.4 Completing the media experience by adding volume controls

Back in the `init` function, you have this code, which should make a lot more sense now:

```

this.video = this.initMedia("video");
this.audio = this.initMedia("audio");

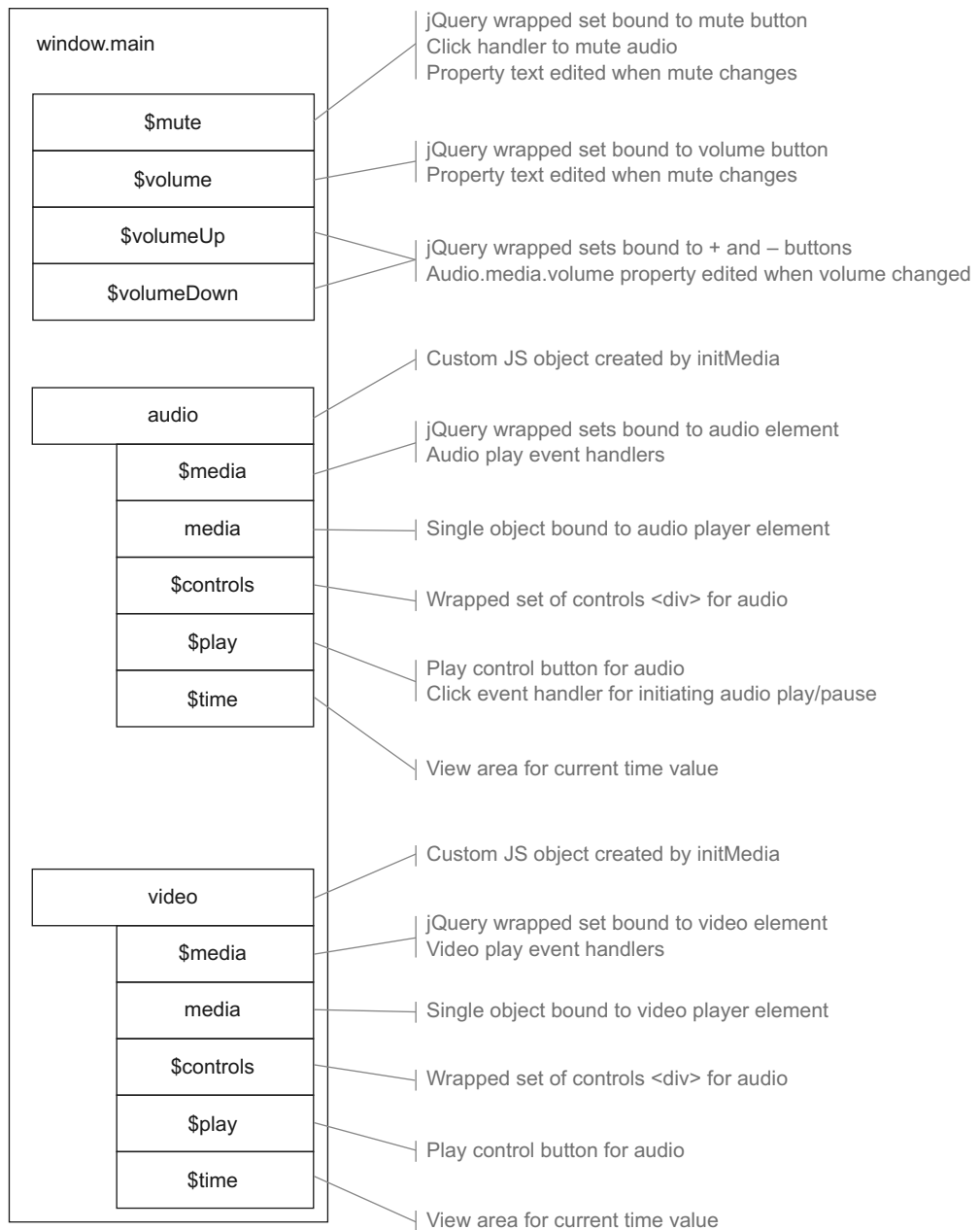
```

Core API



You're assigning a variable called `video` and one called `audio` to new objects created in the `initMedia` function. These objects take care of everything related to play, pause, loop, and content timer functionality. The only bits left to fill in are those pieces specific to the audio track.

Because the example video track you're using doesn't happen to have audio, your code must diverge from doing work inside the `initMedia` function. If you wanted to



**Figure 3.5** Due to the polymorphic<sup>1</sup> nature of `HTMLMediaElement`, you can create both audio and video objects in the same function. Volume control is separated in these objects because it's only being controlled when playing audio content in this example. Therefore, it doesn't need to be part of the polymorphic object.

<sup>1</sup> For more on polymorphism and encapsulation, see [http://en.wikipedia.org/wiki/Polymorphic\\_code](http://en.wikipedia.org/wiki/Polymorphic_code).

add volume controls to video as well as audio, you could have performed this work inside `initMedia`, but for this example we'll have you put it inside `init` so that the volume HTML elements only bind to the audio control. You can, however, still use the audio object you created.

The next listing shows the binding of controls for turning volume up and down and muting the audio, along with a simple binding statement to track the volume-change event. This code fills in the `init` function of the main object.

### Listing 3.6 `init` function binding UI events to the media object created in `initMedia`

```

this.$volume = $("#volume");

this.$volumeUp = $("#volume-up")
    .click(function () {
        self.audio.media.muted = false;
        self.audio.media.volume += 0.1;
    });

this.$volumeDown = $("#volume-down")
    .click(function () {
        self.audio.media.muted = false;
        self.audio.media.volume -= 0.1;
    });

this.$mute = $("#mute")
    .click(function () {
        self.audio.media.muted = !self.audio.media.muted;
    });

this.audio.$media
    .bind("volumechange", function () {
        self.showVolume();
    });

this.showVolume();

```

**Ensure mute is turned off and add 10% to volume.**

**Ensure mute is turned off and subtract 10% from current volume.**

**Toggle muted Boolean property.**

**Track audio control's volume change event to show current volume.**

The final step to getting your application to run is to fill in the `showVolume` function as shown in the next listing. This will simply round the volume off to the nearest tenth (0.1) value and display it on the page.

### Listing 3.7 `showVolume` function to update volume information on the page

```

var prettyVolume =
    Math.round(this.audio.media.volume * 10) / 10;
if (this.audio.media.muted) {
    prettyVolume = 0;
    this.$mute.text("unmute");
}
else {
    this.$mute.text("mute");
}
this.$volume.text(prettyVolume);

```

**Round off volume to nearest tenth**

**Check muted property of audio control**

**Assign text to `$mute` element based on current mute setting**

**Display volume on page**



**Figure 3.6** The completed application playing audio and video content, controlled by your own JavaScript and HTML elements

You should be able to run your application in Chrome, Internet Explorer, and Safari and see everything running. Load the Players page, and the music and video should immediately start playing, as shown in figure 3.6.

### 3.4 Updating media types for open source content

We specifically left out Opera and Firefox in our list of browsers that will work as-is in the Visual Studio solution. These browsers are perfectly compatible with HTML5 `<audio>` and `<video>` tags, but when the page is running in your local environment, you may need to make a few tweaks. These tweaks are related to the open source content types, not to any specific server compatibility. Opera and Firefox support the .ogv and .ogg file types by default, so you have to tell the local web server that these are OK to send out.

This section will cover the changes you need to make to the project.

### USING IIS EXPRESS

The first thing you need to do is update your solution so that it uses IIS Express. You could push this all the way into an IIS Server instance, but that's really not necessary for your tests and would generally be the job of a network administration person anyway. To update your AudioVideo project, follow these steps:

- 1 Right-click on your AudioVideo project node in Solution Explorer and select Properties.
- 2 Click on the Web tab on the left side; you should see a screen similar to figure 3.7.
- 3 About halfway down the page, in the Servers section of the page, select Use Local IIS Web Server and then check Use IIS Express. You can leave the default Project URL as is. (If you need more information about installing IIS Express, please refer to appendix C.)

### ASSIGNING CONTENT TYPES

Now that you have set up the solution to run under IIS Express, you can assign the .ogg and .ogv content types for the local server. Currently, this can only be done using the appcmd executable when running IIS Express. Appcmd is a utility program that can be used for editing a number of configuration values, but updating the available content types is the only change we're after.

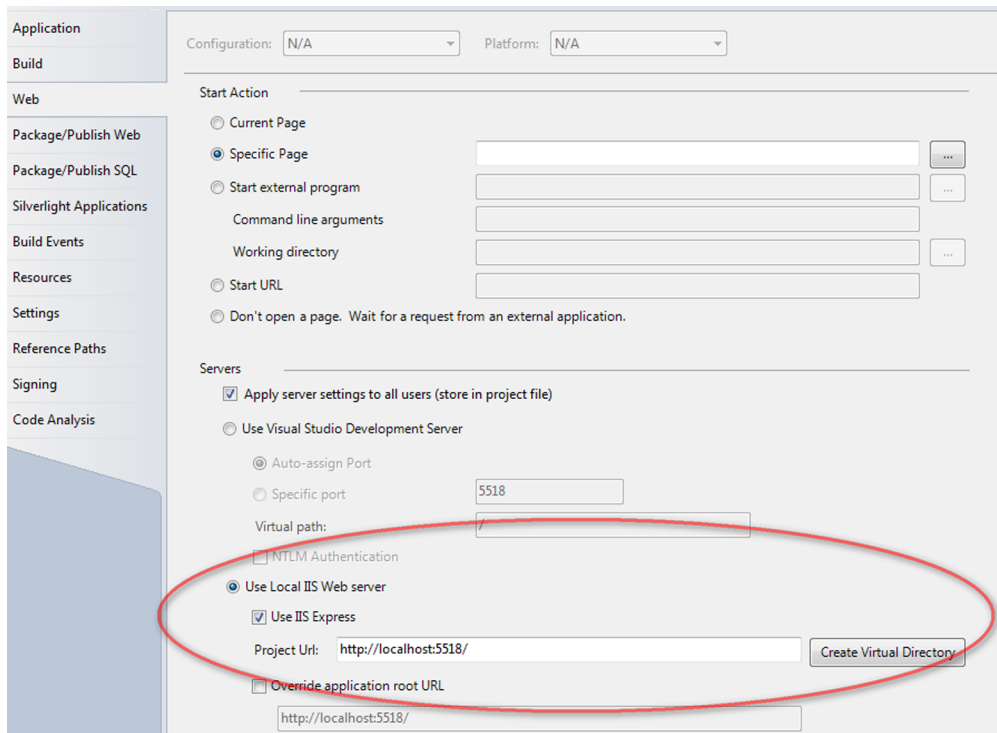
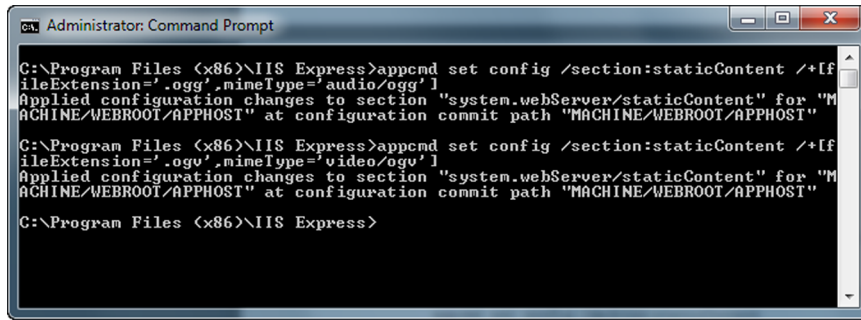


Figure 3.7 Set up the application to run using the local IIS Express instance.



**Figure 3.8** The `appcmd` utility program can add the proper content types to the local IIS Express instance. This setting will then work across all applications that use IIS Express on the local machine.

Follow these steps:

- 1 Open a command prompt as an administrator.
- 2 Navigate to either the Program Files or Program Files (x86) folder.
- 3 Navigate into the IIS Express folder.
- 4 Run the following command to update the .ogg content type:

```
appcmd set config /section:staticContent  
/+ [fileExtension='.ogg',mimeType='audio/ogg']
```

- 5 Run the following command to update the .ogv content type:

```
appcmd set config /section:staticContent  
/+ [fileExtension='.ogv',mimeType='video/ogv']
```

You should see a screen similar to figure 3.8 in the command-line window.

Run your program now using any browser you like, and you should get an identical experience! You're playing audio and video content with no plugins and with very little extraneous code. This is a huge leap forward from what was available just a couple of years ago, and it's only the beginning of what will probably be available in the coming years, as formats and specifications stabilize.

### 3.5 Summary

Streaming audio and video content may be the core of what you want to accomplish with your website or HTML application, or it may add the final touch of interactivity, interest, and professionalism to your site. Regardless of your reasons for using the `<audio>` and `<video>` tags, their current compatibility levels point new applications toward a plugin-free experience, with Flash or Silverlight only being necessary as a fall-back until the older browsers die off. The sample application in this chapter gives you a solid foundation to continue building upon. Finding a specific portion of the content using the `seek` function and monitoring the caching process with various events are some possible directions you could look in for additional studies.

In the next chapter, we'll dig into the basics of drawing on the web using the HTML5 Canvas API. This will be a deeper topic on the JavaScript front, and the project in that chapter should be a really fun way to start learning the correlation between markup and code.

### 3.6 Complete code listings

The following code is provided to let you check your work or build the project from scratch if you haven't been building along.

#### Listing 3.8 The complete Players.cshtml code

```
@{ ViewBag.Title = "Players"; }
<div id="content">
  <audio id="audio">
    <source src="@Url.Content("~/Content/gwt.ogg")" type="audio/ogg" >
    <source src="@Url.Content("~/Content/gwt.mp3")" type="audio/mp3" >
  </audio>
  <video id="video">
    <source src="@Url.Content("~/Content/lego.ogv")" type="video/ogg" >
    <source src="@Url.Content("~/Content/lego.mp4")" type="video/mp4" >
  </video>
  <div id="video-controls">
    Video:
    <button class="play">play</button>
    <span class="time"></span>
  </div>
  <div id="audio-controls">
    Audio:
    <button class="play">play</button>
    <span class="time"></span>
    <div class="secondary-controls">
      Volume:
      <button id="volume-up">+</button>
      <button id="volume-down">-</button>
      <button id="mute">mute</button>
      <span id="volume"></span>
    </div>
  </div>
  <div id="avfooter">
    <p>Video copyright 2012,
      <a href="http://iangilman.com">Ian Gilman</a>.</p>
    <p>Audio by <a href="http://gwaxtrio.bandcamp.com">
      Gennaro's Wax Trio</a>, copyright 2008, BMI, BLZDub Music;
      used by permission.</p>
  </div>
</div>
<script src="@Url.Content("~/Scripts/main.js")"
  type="text/javascript"></script>
```

## Listing 3.9 The complete code listing for main.js

```

$(document).ready(function () {
    Main.init();
});

window.Main = {

    //-----
    init: function () {
        var self = this;

        if (!Modernizr.audio) {
            alert("Audio tag not supported.");
            return;
        }

        if (!Modernizr.video) {
            alert("Video tag not supported.");
        }

        this.video = this.initMedia("video");
        this.audio = this.initMedia("audio");

        this.$volume = $("#volume");

        this.$volumeUp = $("#volume-up")
            .click(function () {
                self.audio.media.muted = false;
                self.audio.media.volume += 0.1;
            });

        this.$volumeDown = $("#volume-down")
            .click(function () {
                self.audio.media.muted = false;
                self.audio.media.volume -= 0.1;
            });

        this.$mute = $("#mute")
            .click(function () {
                self.audio.media.muted = !self.audio.media.muted;
            });

        this.audio.$media
            .bind("volumechange", function () {
                self.showVolume();
            });

        this.showVolume();
    },

    //-----
    initMedia: function (name) {
        var result = {};
        result.$media = $("#" + name);
        result.media = result.$media[0];
        result.$controls = $("#" + name + "-controls");
        result.$play = result.$controls.find(".play");
        result.$time = result.$controls.find(".time");
    }
};

```

```

result.$play.click(function () {
    if (result.media.paused)
        result.media.play();
    else
        result.media.pause();
});

result.$media
    .bind("playing", function () {
        result.$play.text("pause");
    })
    .bind("pause", function () {
        result.$play.text("play");
    })
    .bind("ended", function () {
        result.media.play();
    })
    .bind("timeupdate", function () {
        var prettyTime =
            Math.round(result.media.currentTime * 100) / 100;
        result.$time.text("time: " + prettyTime + "s");
    });

result.media.play();
return result;
},

//-----
showVolume: function () {
    var prettyVolume =
        Math.round(this.audio.media.volume * 10) / 10;
    if (this.audio.media.muted) {
        prettyVolume = 0;
        this.$mute.text("unmute");
    }
    else {
        this.$mute.text("mute");
    }
    this.$volume.text(prettyVolume);
}
};

```

**Listing 3.10** Styles added to site.css to support audio/video formatting

```

/*--- audio/video ----*/

#content {
    width: 100%;
    max-width: 400px;
    margin: 10px auto;
}

#video {
    width: 400px;
    height: 400px;
}

```

```
#audio {
    display: block;
}

button {
    padding: 5px;
}

#video-controls {
    margin-top: 25px;
}

#audio-controls {
    margin-top: 25px;
}

.secondary-controls {
    margin-top: 10px;
}

#avfooter {
    margin-top: 50px;
    font-size: 12px;
    color: #888;
}

#avfooter a,
#avfooter a:visited {
    color: #555;
}
```

# HTML5 for .NET Developers

Jackson • Gilman



A shift is underway for Microsoft developers—to build web applications you’ll need to integrate HTML5 features like Canvas-based graphics and the new JavaScript-driven APIs with familiar technologies like ASP.NET MVC and WCF. This book is designed for you.

**HTML5 for .NET Developers** teaches you how to blend HTML5 with your current .NET tools and practices. You’ll start with a quick overview of the new HTML5 features and the semantic markup model. Then, you’ll systematically work through the JavaScript APIs as you learn to build single page web apps that look and work like desktop apps. Along the way, you’ll get tips and learn techniques that will prepare you to build “metro-style” applications for Windows 8 and WP 8.

## What's Inside

- HTML5 from a .NET perspective
- Local storage, threading, and WebSockets
- Using JSON-enabled web services
- WCF services for HTML5
- How to build single page web apps

This book assumes you’re familiar with HTML, and concentrates on the intersection between new HTML5 features and Microsoft-specific technologies.

**Jim Jackson** is a software consultant and project lead specializing in HTML5-driven media. **Ian Gilman** is a professional developer passionate about open technologies and lively user interfaces.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit [manning.com/HTML5for.NETDevelopers](http://manning.com/HTML5for.NETDevelopers)

“Speaks directly to the interests and concerns of the .NET developer.”

—From the Foreword by  
Scott Hanselman, Microsoft

“Looks under the hood of HTML5 to teach more than just pretty pages.”

—Joseph M. Morgan, Amerigroup

“A comprehensive jumpstart for the .NET developer looking to make a leap into HTML5.”

—Peter O’Hanlon  
Lifestyle Computing Ltd

“A great HTML5 and API learning resource!”

—Stan Bice  
Applied Information Sciences