
Create modern Web sites using HTML5 and CSS3

Implementing the canvas and video elements in HTML5

Skill Level: Intermediate

[Joe Lennon](#)

Software developer
Freelance

02 Mar 2010

Since the World Wide Web emerged in the early 1990s, HTML has evolved to become a relatively powerful markup language, which, when backed up by its close partners JavaScript and CSS, can be used to create visually stunning and interactive Web sites and applications. This tutorial serves as a hands-on introduction to HTML5 and CSS3. It provides information about the functionality and syntax for many of the new elements and APIs that HTML5 has to offer, as well as the new selectors, effects, and features that CSS3 brings to the table. Finally, it will show you how to develop a sample Web page that harnesses many of these new features. By the time you have finished this tutorial, you will be ready to build Web sites or applications of your own that are powered by HTML5 and CSS3.

Section 1. Before you start

This tutorial assumes some basic experience with HTML, CSS, and JavaScript. It assumes that you are aware of what an HTML element or tag is, what an attribute means, the basic syntax of HTML markup, the general structure of a Web page, and so on. In terms of CSS, you are expected to be familiar with element, class, and ID-based selectors, the syntax of a CSS property, and how to include CSS in your Web pages using inline or external stylesheets. Finally, it is assumed that you have some working knowledge of JavaScript, such as what a variable, function, if statement, and for loop is, as well as how to include JavaScript code in your Web pages. If you feel that you need to brush up on any of these technologies before you

begin, skip ahead to the [Resources](#) section for some useful tutorials and articles that will bring you up to speed on the basics of HTML, CSS, and JavaScript development.

About this tutorial

Over the past ten years or so, concepts such as Web 2.0, Rich Internet Applications (RIAs), and the Semantic Web have all pushed HTML, CSS, and JavaScript to and beyond their limits, often relying on plug-ins such as Adobe® Flash to power components such as video and audio, as well as highly graphical and interactive applications. The Adobe Flex development framework, Microsoft®'s Silverlight platform, and JavaFX have all looked to provide support where HTML's weaknesses made developers' lives difficult. With HTML5, however, the markup language is striking back, with full multimedia support, local storage and offline application support, a native 2D drawing API, and a host of new application development APIs, all provided with the intent of proving that HTML, CSS, and JavaScript can provide a rich front end to your Web sites and applications.

HTML5 is widely regarded as one of the most important new technologies scheduled to emerge in 2010, and there are already several books being written on the subject, some of which are due to be published as early as March of this year. For many years, the Web has relied on external plug-ins to deliver features that cannot be natively supported by the Web browser, particularly in terms of 2D drawing, animation, and multimedia. The latest versions of the HTML and CSS specification aim to remove the need for these additional browser components to facilitate such features, as well as reduce the amount of JavaScript required (or removing the need for JavaScript entirely, in some cases) for such trivial things as row drag and drop, row striping, and more. Follow along in this tutorial to learn how to take advantage of HTML5.

Prerequisites

HTML5 is a relatively young specification, and as a result, browser support is quite limited (at the time of writing). The code presented in this tutorial is built to be as cross-browser compatible as possible, but some features will not work in all browsers. Any features that are currently browser-specific will be clearly identified in the tutorial. To ensure that you can experience all of these new features, it is recommended that you install the latest versions of the following Web browsers on your system when developing HTML5 and CSS3 applications:

- [Mozilla Firefox \(version 3.5+\)](#)
- [Apple Safari \(version 4.0+\)](#)

- [Opera \(version 10.0+\)](#)
- [Google Chrome \(version 3.0+\)](#)

You do not need any specific software to write HTML and CSS code; any basic text editor will do (such as Notepad, vi, emacs, and so on.) In this tutorial, it is assumed that the [source code](#) is stored in a directory on your local computer—you do not need to use a Web server or upload the files to a Web hosting service.

Section 2. New features in HTML5

In this section, you will discover some of the great new features that HTML5 has to offer. You will first learn about the new semantic elements that aim to give meaning to the various parts of a modern Web page: headers, footers, navigation bars, side bars, and so forth. Next, you will learn about the important new `<canvas>` element and the 2D drawing JavaScript APIs that you can use to create shapes, text, animations, transitions, and more. Following this, you will see how the new `<audio>` and `<video>` elements intend on replacing the Web's current dependency on Flash as a multimedia delivery platform. Next, you will be introduced to the local storage APIs and offline applications support that will further bring Web applications in line with their desktop counterparts in terms of functionality, even when not connected to a network or the Internet. This section is wrapped up with a brief overview of the other new elements, attributes, and APIs that are proposed in the current HTML5 specification.

HTML5 fundamentals <http://www.ibm.com/developerworks/training/kp/wa-kp-html5/>

Semantic elements

The HTML5 specification includes a series of new semantic elements that is used to give some meaning to the various sections or parts of a Web page, such as a header, footer, navigation, and so on. In previous versions of HTML, you would typically use `<div>` elements to create these parts, using ID or class attributes to differentiate them from each other. The problem with this is that this has no semantic meaning, as there are no strict rules defined that specify what class names or IDs are to be used, making it extremely difficult for software to determine what the particular area is doing. HTML5 should help alleviate these issues, making it easier for Web browsers to parse the semantic structure of a document.

It is worth pointing out that continuing to use `<div>` elements in HTML5 is perfectly valid, but in order to future-proof your work, it is recommended that you use semantic elements where relevant. On the other side of the coin, it is also suggested

that you avoid using these new elements for purposes other than their intended. For example, the `<nav>` element should not be used for just any group of links; it is intended to surround the main navigation block on the page.

The main semantic elements that HTML5 introduces are:

`<header>`

This element is used to define a header for some part of a Web page, be it the entire page, an `<article>` element, or a `<section>` element.

`<footer>`

Like the `<header>` element, this new element defines a footer for some part of a page. A footer does not have to be included at the end of a page, article, or section, but it typically does.

`<nav>`

This is a container for the primary navigation links on a Web page. This element is not intended for use with all groups of links and should be used for major navigation blocks only. If you have a `<footer>` element that contains navigation links, you do not need to wrap these links in a `<nav>` element, since the `<footer>` element will suffice on its own.

`<article>`

The `<article>` element is used to define an independent item on the page that can be distributed on its own, such as a news item, blog post, or comment. Such items are typically syndicated using RSS feeds.

`<section>`

This element represents a section of a document or application, such as a chapter or a section of an article or tutorial. For example, the section you are reading now could be surrounded by a `<section>` element in HTML5. `<section>` elements typically have a header, although it is not strictly required. The header for the section you are reading now would contain the text "Semantic elements," for example.

`<aside>`

This new element can be used to mark up a sidebar or some other content that is considered somewhat separate to the content around it. An example of this might be advertising blocks.

`<hgroup>`

In some cases, a page, article, or section may require more than one heading, such as where you have a title and a subtitle. This tutorial, for example, has the title "Create modern Web sites using HTML5 and CSS3" and the subtitle "Implementing the canvas and video elements in HTML5." You could wrap these in an `<hgroup>` element, using an `<h1>` element for the main title and an

<h2> element for the subtitle.

The sample Web site at the end of this tutorial includes several of these new semantic elements, and I will explain their syntax and use in more detail at that point.

The <canvas> element

The <canvas> element was originally developed by Apple® for use in Mac OS X Dashboard widgets and in Safari, but was later adopted by Mozilla® and Opera® in their Web browsers. The element has been standardized and included in the HTML5 specification, along with a series of 2D drawing APIs that can be used to create shapes, text, transitions, and animations inside the element.

Many believe that the <canvas> element is one of the most important aspects of HTML5 as it facilitates the production of graphs, interactive games, paint applications, and other graphics on the fly without requiring external plug-ins such as Adobe Flash.

The <canvas> element itself is quite basic, defining the width, height, and unique ID for the object. The developer must then use a series of JavaScript APIs to actually draw objects on the canvas, typically when the Web page has finished rendering. These APIs allow the developer to draw shapes and lines; apply color, opacity, and gradients; create text; transform canvas objects; and perform animation. The APIs also allow the <canvas> to be interactive and respond to user input such as mouse events and key events, facilitating the production of games and Web applications on the canvas. You will see an example of the <canvas> element in action in the sample HTML5/CSS3 Web site later in this tutorial.

Playing <audio> and <video>

In recent years, the popularity of video sharing sites such as YouTube and content delivery platforms like Hulu has seen a huge explosion in the use of the Web for multimedia streaming. Unfortunately, the Web was not built with such content in mind, and as a result, the provision of video and audio has by and large been facilitated by the Flash Video (.flv) file format and the Adobe Flash platform.

HTML5, however, includes support for two new elements, <audio> and <video>, which allow Web developers to include multimedia content without relying on the user to have additional browser plug-ins installed. Several browsers, including Mozilla Firefox, Apple Safari, and Google Chrome, have begun supporting these new elements and providing standard browser playback controls, should the user choose to use them. In addition, a set of standard JavaScript APIs has been provided to allow developers to create their own playback controls, should they wish

to do so. A key advantage to native multimedia playback is that it theoretically requires less CPU resources, which can lead to energy savings.

A key issue with these new multimedia elements, however, is the file formats supported by each browser and the patent licensing issues that go along with the various codecs that these files can be encoded with. Mozilla and Opera want to use the open source Theora video container and codec, which does not require patent licensing for the inclusion of the codecs in the Web browser. On the other hand, Apple and Google are not happy with the quality of Theora, particularly for the delivery of high definition (HD) content on the likes of YouTube. They prefer the H.264 codec, typically contained in MP4, MOV, or MKV files.

The issue is not just with video however, as the same problems reside with audio codecs. The MP3 and AAC formats are restricted by patents, whereas the Vorbis format is not. The problem with Vorbis audio is that it is not in widespread use, as portable media players and many media software applications do not support it.

There are many decisions to be made about HTML5 `<video>` and `<audio>` in the near future, and it will be interesting to see what codecs and formats are facilitated in the final recommendation. In the meantime, you can try to support all browsers by making video available in a variety of formats and by providing Flash video as a fallback. Let's hope that a final decision is made, and that it is not left to browser vendors to decide which formats to support, as that would essentially render these new elements useless.

Again, you will see the `<video>` element in action later in this tutorial.

Local storage and offline applications

Web developers have traditionally used cookies to store information on a visitor's local machine, allowing a Web page to read this information back at a later point. While cookies are very useful for storing basic data, they are limited by the fact that Web browsers are not required to keep more than 20 cookies per Web server or more than 4KB of data per cookie (including both name and value). In addition, they are sent to the Web server with every HTTP request, which is a waste of resources.

HTML5 provides a solution for these problems with the Local Storage APIs, which are covered in a separate specification to the main HTML5 document. This set of APIs allows developers to store information on the visitor's computer while remaining reasonably confident that they will still be there at a later date. In addition, the information is accessible at any point (even after the page has rendered) and is not loaded automatically with each HTTP request. The specification includes same-origin restrictions, which prevent Web sites from reading or changing data stored by other Web sites.

Most browsers store Web pages in local cache, allowing them to be viewed even if the user is offline. This works fine for static pages, but it is not available for dynamic content that is typically database-driven, such as Gmail, Facebook, or Twitter. HTML5 provides support for offline applications, where the browser downloads all the files necessary to use the application offline, and when the user uses the application offline, the browser can allow any changes made in the process to be uploaded to the server when they reconnect to the Internet.

Web form enhancements

If you have created Web applications before, you are more than likely familiar with HTML's set of form controls, some of which are implemented using the `<input>` element. In HTML 4, the following input types were supported:

- button
- checkbox
- file
- hidden
- image
- password
- reset
- radio
- submit
- text

In addition, there are some other elements that are used in forms such as `<select>` and `<textarea>`. These form controls provide plenty of function for basic form fields such as name, phone number, and address—like you might find on a contact form. But, the Web as a platform has grown far beyond the stage where HTML forms are used to submit contact forms—now they are used to submit application data for server-side processing. As a result, Web application developers find themselves continually in need of some more sophisticated form controls, such as spinners, sliders, date/time pickers, color pickers, and so on.

In order to tap into these types of controls, developers needed to use an external JavaScript library that provided UI components, or else use an alternative development framework such as Adobe Flex, Microsoft Silverlight, or JavaFX. HTML5 aims to fill some of the gaps left by its predecessor in this space by providing a whole range of new form input types:

- color
- date
- datetime
- datetime-local
- email
- month
- number
- range
- search
- tel
- time
- url
- week

At the moment, support for these new form fields is quite limited. The Mobile Safari browser on the iPhone makes use of some of these new types to change the type of keyboard presented to the user (for example, with the e-mail type, the @ symbol and .com shortcuts will be shown). Also, Opera provides some new widgets for many of these controls, including a spinner for the number type and a calendar date picker for the date-related types. The most widely available type of these new offerings is the range type, which is rendered as a slider by Opera, Safari, and Google Chrome.

In addition to these new input types, HTML5 also supports two major new features for form fields. The first of these is autofocus, which tells a browser to automatically give focus to a particular form field when the page has rendered, without requiring JavaScript code to do so. The second enhancement is the placeholder attribute, which allows the developer to define the text that will appear in a textbox-based control when its contents are empty. An example of this would be a search box where the developer would prefer not to use a label outside the box itself. The placeholder attribute allows the developer to specify text that will show when the value of the control is empty and the control does not have focus. An example of this is shown in [Figure 1](#).

Figure 1. The placeholder attribute in action

A screenshot of a web form with a light gray background and rounded corners. It contains five input fields and a button. The first field is labeled 'Name:' and contains the text 'Joe Lennon'. The second field is labeled 'E-mail:' and has the placeholder text 'Enter your email address' in gray. The third field is labeled 'Phone:' and has the placeholder text 'Enter your phone number' in gray. The fourth field is labeled 'Callback on:' and is empty. The fifth field is labeled 'Priority:' and is a range input represented by a horizontal slider with a blue circular handle. Below these fields is a button labeled 'Request Call'.

As Figure 1 shows, the placeholder text for e-mail address and phone number appears in grey, while the field is empty and does not have focus. This screenshot also shows an example of a range input type, represented here by a slider in the Safari browser. This screenshot is taken from the sample Web page discussed later in this tutorial.

Other new features

HTML5 includes so many new features; it's impossible to cover them all in this tutorial. This section provides a brief overview of some of the other enhancements in the specification.

Web worker

This allows JavaScript code to be set to run in a background process facilitating the development of multi-threaded applications. The chief benefit that Web workers offer developers is that intensive calculations can be processed in the background without adversely affecting the speed of the user interface.

Geolocation

HTML5 includes a geolocation API that allows a Web application to determine your current geographical location, assuming the device you are targeting provides features for finding such information (for example, GPS on a cellphone). If you do not have a device that supports this feature (such as an iPhone or an Android 2.0-based smartphone), you can use Firefox and download a plug-in that allows you to set your location manually.

Drag and Drop

Another interesting feature is the inclusion of a drag and drop API. Up until now, implementation of drag and drop without plug-ins was dependent on

some very complex JavaScript or the use of a JavaScript library such as script.aculo.us.

Cross-document messaging

This allows documents in different windows (and iframes, for that matter) to send and receive messages to one another. This feature could prove very useful for the development of widgets and applications that are hosted on servers other than the primary Web page's server (similar to Facebook applications).

And more

Other new features introduced by HTML5 include MIME types and protocol handler registration, so Web applications can be registered as the default application for a particular file type or protocol; browser history management, which until now needed to be implemented manually or using an external JavaScript framework; and a host of other new elements and attributes that make Web developers' lives easier.

Section 3. New features in CSS3

This section introduces you to the new features that can be found in the CSS level 3 specifications—including new CSS selectors such as structural, state-based, and negation pseudo-classes, as well as other new types of selectors. It also shows many of the effects that CSS3 offers that would previously have required images to be created using a separate application and saved as GIF, JPG, or PNG. Such effects include drop shadows on text and boxes, rounded corners on borders, and the use of opacity to create a translucent appearance. Many of these features (such as opacity and rounded corners) are relatively widespread in use due to the fact that they degrade very gracefully in legacy Web browsers. Next, you will learn about the new multicolumn layouts that can be created using CSS3. These layouts are a throwback to the newspaper layout where text will stretch over a set number of columns or to a particular column width as required. Another feature that will be discussed is the issue of including non-standard Web fonts using the @font-face tag. Finally, some of the other new CSS3 features will be introduced, such as support for HSL (Hue, Saturation, and Lightness) and RGBA (Red, Green, Blue, and Alpha) color models.

New selectors

A CSS selector refers to the manner in which HTML element(s) are styled using a

stylesheet. For example, to put a border around all `<div>` elements you would use the selector `div:div { border: 1px solid #000; }`.

To apply a background color to all elements with the class `highlight` you would use the selector `.highlight: .highlight { background-color: yellow; }`.

Finally, to change the width of an element with an ID attribute value of `myDiv`, you would use: `#myDiv { width: 250px; }`.

Of course, these selectors can be combined, so to select all `<div>` elements with the class `highlight`, you would use `div.highlight`, or to select the `<div>` element with the ID `myDiv` you would use `div#myDiv`.

In addition to these straightforward selectors, CSS also includes (and has done since previous versions) a series of more complex selectors. For example you can use the selector `input[type="text"]` to select only the input elements that contain an attribute type with the value of *text*. Or you can use the pseudo-class `:hover` to style an element when the mouse is over it, for example: `a:hover { color: red; }`.

There are many more of these selectors, all of which are provided to make it easier to select an element to style. With CSS3, even more new selectors have been added to the mix, all of which make developers' lives easier and reduce the amount of HTML and JavaScript they need to write.

Attribute selectors

E[foo^="bar"]

Select an element, E, whose `foo` attribute begins exactly with the string `bar`

E[foo\$="bar"]

Select an element, E, whose `foo` attribute ends exactly with the string `bar`

E[foo*="bar"]

Select an element, E, whose `foo` attribute contains the string `bar`

Structural pseudo-classes

E:root

Select an element, E, the root of the document (the `<html>` tag)

E:nth-child(n)

Select an element, E, the *n*-th child of its parent element

E:nth-last-child(n)

Select an element, E, the *n*-th last child of its parent element

E:nth-of-type(n)

Select an element, E, the n-th sibling of its type

E:nth-last-of-type(n)

Select an element, E, the n-th last sibling of its type

E:last-child

Select an element, E, whose is the last child of its parent element (Note that E:first-child was previously defined in CSS2)

E:first-of-type

Select an element, E, the first sibling of its type

E:last-of-type

Select an element, E, the last sibling of its type

E:only-child

Select an element, E, the only child of its parent element

E:only-of-type

Select an element, E, the only sibling of its type

E:empty

Select an element, E, that has no children (including text nodes)

The target pseudo-class**E:target**

Select an element, E, who is the target of the referring URI

UI element states pseudo-classes**E:enabled**

Select a user interface element, E, which is enabled

E:disabled

Select a user interface element, E, which is disabled

E:checked

Select a user interface element, E (a radio button or checkbox), which is checked or selected

Negation pseudo-class**E:not(s)**

Select an element, E, which does not match the simple selector s

General sibling combinator

E ~ F

Select an element, F, which is preceded by an element, E

Browser support for the new attribute selectors and the general sibling combinator is excellent, as they work on almost all modern Web browsers. Support for the new pseudo-classes is included with the latest versions of most browsers, but you may need to include fallbacks for older browsers such as Internet Explorer 6/7 and Firefox 3.0.

New effects

Although the new selectors are probably the features that reduce developers' headaches the most, the enhancements people most want to see are the shiny new effects that are available. These are facilitated through a slew of new CSS properties, including:

- background (now supports multiple backgrounds)
- background-clip
- background-origin
- background-size
- border-radius (rounded corners)
- border-image
- border-color (gradient borders)
- box-shadow (drop shadow on boxes without images)
- box-sizing
- opacity
- outline-offset
- resize
- text-overflow
- text-shadow
- word-wrap

The sample HTML5/CSS3 Web page you'll create at the end of this tutorial will show some of these new effects in action.

Multicolumn layouts

CSS3 multicolumn layouts allow for text to be spread over a number of columns, like you might find in a newspaper. This can be done in two ways, either using the column-width property, where you define how wide each column should be (the number of columns is determined by the space available to the container), or using column-count, where you define the number of columns that should be used. The width will then vary based on the space available to each column.

Other features of multicolumn layouts include the column-gap property, which allows the developer to define the space that should be present between columns when the column-width method is used. Another useful addition is the column-rule property, which allows a border-style rule to be placed between columns. Finally there is the column-space-distribution property, which determines how left over space should be allocated between the columns.

Multicolumn layouts are currently supported by Mozilla and WebKit browsers. At present, these are implemented by means of temporary proprietary properties prefixed with -moz or -webkit, respectively. When the specification has been finalized, these browsers will eventually move to the CSS standard properties.

Web fonts

Web fonts were actually proposed for CSS2 and have been available in Microsoft Internet Explorer since version 5. Unfortunately, this required the use of the proprietary .eot (Embedded Open Type) font format, and none of the other browser vendors chose to implement it. As a result, Web fonts never really took off on CSS2-based Web sites.

With the latest versions of Firefox, Safari, Chrome, and Opera, however, you can now use the @font-face rule to use any licensed .ttf (TrueType) or .otf (OpenType) font on your Web page. An example of the @font-face rule is as follows:

```
@font-face { font-family: Alexa; src: url('Alexa.otf'); }
```

You can now use this font in your own CSS rules, such as:

```
article p {  
font-family: Alexa, Arial, Helvetica, sans-serif; }
```

Remember that fonts are like images—if they are not your own you may need permission to use them on the Web. Alternatively, you can pay for (or download for free) some royalty-free fonts that can be included on your Web pages as you please.

Other new features

CSS3 also includes many other new features, including support for new color values, notably HSL (Hue, Saturation, Lightness) and two color values with alpha properties—RGBA (Red, Green, Blue, Alpha) and HSLA (Hue, Saturation, Lightness, Alpha). Media queries will allow you to define different styles for different devices based on their viewport size. For example, you can provide specific styles for devices with a viewport of less than 500 pixels (such as a smartphone, PDA, or other mobile device). CSS3's speech module allows you to control properties of the speech of a screen reader, including voice volume, balance, rate, stress, and more.

Section 4. Putting it all together: The sample HTML5/CSS3 page

At this stage, you must be anxious to get your hands dirty and start creating pages that use the great new features that HTML5 and CSS3 have to offer. In this section, you will create an HTML page that contains many of these new features. This section of the tutorial will divide the development of the page into subsections as follows:

- The basic HTML5 page structure
- Using the new semantic elements
- Introducing the <video> element
- Web form enhancements
- The <canvas> element and the 2D drawing API

The entire source will be built into a single HTML file, named `index.html` (see [Downloads](#)). You can open this file from any location on your computer; it does not need to be deployed to a Web server to load. Some browsers may give you warnings about running scripts locally, so be sure to keep dynamic scripting features on if asked.

Along the way, the CSS rules for the page will be added to an external stylesheet file, named `styles.css`. You will see some examples of new CSS3 properties such as `border-radius`, `text-shadow`, and `box-shadow` in this sample page.

Anyway, that's enough of the housekeeping, let's get started with some HTML5!

The basic HTML5 page structure

As you may be aware, Web browsers can operate in different modes depending on whether a valid doctype is available in an HTML document. If no valid doctype is found, the browser will operate in quirks mode, where some non-standard features are maintained for backwards-compatibility. If a valid doctype is found, the browser operates in standards mode, according to W3C and IETF standards.

The doctype for HTML5 is very simple: `<!doctype html>`.

You should include this line at the top of every HTML5 page you create. Let's continue and define the basic outline for the index.html page. The code for this can be found in [Listing 1](#).

Listing 1. Basic HTML5 document structure

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>HTML5 Demo</title>
    <link rel="stylesheet" href="styles.css" />
    <!--[if IE]>
      <script src="http://html5shiv.googlecode.com/svn/trunk
/html5.js"></script>
    <![endif]-->
  </head>
  <body>

  </body>
</html>
```

You may notice that unlike the recent XHTML flavors of HTML, the opening `<html>` tag no longer requires the `xmlns` or `xml:lang` attributes, and the `lang` attribute alone suffices. Similarly, the `<meta>` element features a new shorthand attribute `charset` for defining the page's character encoding. It is worth pointing out that these changes just make it easier and reduce the amount of code required to perform simple tasks. The old methods are still perfectly valid.

You may also be wondering why the `<link>` element doesn't feature a `type` attribute. CSS is the only supported stylesheet type at present, and all modern browsers will assume the type is `text/css` if none is supplied, so it is not required. Again, it is perfectly acceptable to provide the `type` attribute if you so wish.

The final point of note in [Listing 1](#) is the `<script>` element, which loads the externally hosted JavaScript file `html5.js`. As Microsoft Internet Explorer browsers (even version 8) do not support the new HTML5 elements, the browser does not recognise the likes of `<article>`, `<section>`, `<header>`, and so on. Not only does it not recognise these elements, but it also prevents them from being styled. A known hack for circumventing this issue is to use the JavaScript function `document.createElement()` to create dummy elements for each tag. This script will do this for every new HTML5 element so you don't have to worry about it. I highly

recommend including this script in all of your HTML5 work.

Next, you will add some of the new semantic elements to the page to create the page's visual structure. You will also create some stylesheet rules to make the page look more presentable.

Using the new semantic elements

To illustrate how the new semantic HTML5 elements should be used, you will now add these to the index.html file. The basic structure of these elements will adhere to the following outline:

- header
 - hgroup
- nav
- article
 - header
 - section
 - header
- footer

As you can see, the page itself opens with a header, followed by a nav, then an article, and finally a footer. The header will have multiple headings using the hgroup element. The article itself will have a header as well as a section element, which too has its own header. The code for this is shown in [Listing 2](#), and should be added between the opening and closing <body> tag from [Listing 1](#).

Listing 2. Using HTML5 semantic elements

```
<header>
  <hgroup>
    <h1>Company Name</h1>
    <h2>An example of HTML5 and CSS3 in action</h2>
  </hgroup>
</header>

<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About Us</a></li>
    <li><a href="#">Contact Us</a></li>
  </ul>
</nav>

<article>
```

```
<header>
  <time datetime="2010-01-12" pubdate>
    <span>Jan</span> 12
  </time>
  <h1>
    <a href="#" title="Link to this post"
rel="bookmark">Article Heading</a>
  </h1>
</header>
<p>This is an article that demonstrates some of the new features
that HTML5 and CSS3 has to offer. This article contains several sections, each
with its own heading, as well as a video element which will play a video without
Flash on browsers that support it.</p>
<section>
  <header>
    <h1>This is a section heading</h1>
  </header>
  <p>This is an example of a basic section of a document.
A section can refer to different parts of a document, application, or page.
According to the draft W3C spec, HTML5 sections usually have headings.</p>
</section>
</article>

<footer>
  <p>&copy; 2009 Company Name. All rights reserved.</p>
</footer>
```

The code in [Listing 2](#) should be relatively self-explanatory. The main `<header>` element consists of a `<hgroup>` element with two headings: a `<h1>` title and a `<h2>` subtitle element. The main `<nav>` element is an unordered list with three items, each a link to a fictitious page on the site. The `<article>` element contains its own `<header>`, with a `<time>` element for the publication date of the article. You will notice that this element contains an attribute `datetime`, which specifies a standardized form of the date of the post that is easy for computers to read. The content of the time element is "Jan 12," which humans will find easier to read. The `pubdate` attribute indicates that this is a publication date for the article in question.

Beneath the header is a normal HTML paragraph, and this is followed by a new `<section>` element, which contains a `<header>` with the section's title and a paragraph. You will create more sections for the additional areas of the page created in the remaining sections of this tutorial.

If you open this page in your browser, it won't look very pretty, as it has not yet been styled with CSS. Let's add some styles to make the page look a bit better. Add the code from [Listing 3](#) to the `styles.css` file.

Listing 3. CSS styles for the new semantic HTML5 elements

```
* {
  font-family: Lucida Sans, Arial, Helvetica, sans-serif;
}

body {
  width: 480px; margin: 0px auto;
}
```

```
header h1 {
  font-size: 36px; margin: 0px;
}

header h2 {
  font-size: 18px; margin: 0px; color: #888;
  font-style: italic;
}

nav ul {
  list-style: none; padding: 0px; display: block;
  clear: right; background-color: #666;
  padding-left: 4px; height: 24px;
}

nav ul li {
  display: inline; padding: 0px 20px 5px 10px;
  height: 24px; border-right: 1px solid #ccc;
}

nav ul li a {
  color: #EFD3D3; text-decoration: none;
  font-size: 13px; font-weight: bold;
}

nav ul li a:hover {
  color: #fff;
}

article > header time {
  font-size: 14px; display: block; width: 26px;
  padding: 2px; text-align: center; background-color: #993333;
  color: #fff; font-weight: bold; -moz-border-radius: 6px;
  -webkit-border-radius: 6px; border-radius: 6px; float: left;
  margin-bottom: 10px;
}

article > header time span {
  font-size: 10px; font-weight: normal;
  text-transform: uppercase;
}

article > header h1 {
  font-size: 20px; float: left;
  margin-left: 14px; text-shadow: 2px 2px 5px #333;
}

article > header h1 a {
  color: #993333;
}

article > section header h1 {
  font-size: 16px;
}

article p {
  clear: both;
}

footer p {
  text-align: center; font-size: 12px;
  color: #888; margin-top: 24px;
}
```

Most of these CSS rules contain properties that have been available in CSS for a while now, but you may notice that the rule `article > header time` contains

border-radius properties (including browser-specific properties for Mozilla and WebKit-based browsers). This adds a rounded corner to the date/time box on supported browsers, such as Firefox, Safari, and Chrome.

You'll also notice the use of the text-shadow property in the `article > header h1` rule. Most modern browsers support this property.

The nice thing about both of the CSS3 properties introduced in this section is that they degrade gracefully. In other words, if the visitor's Web browser doesn't support these new properties, they will simply be ignored and the elements will be styled according to any other supported properties. The page including its current contents should look like the screenshot in Figure 2 (this is how it looks in Safari 4 on Mac OS X, at least).

Figure 2. Semantic HTML5 elements in action



Next, you will be introduced to the `<video>` element and how to use it to include a Theora video file in your HTML5 page.

Introducing the <video> element

Now you will add a video to the index.html page along with a set of browser-supplied playback controls. Please note that the video included in this sample is a Theora video, which can be played on Firefox and Google Chrome only. Safari currently only supports H.264 encoded videos. Add the code from [Listing 4](#) beneath the last closing </section> tag from [Listing 2](#), but above the closing </article> tag.

Listing 4. Adding a video to the page

```
<section>
  <header>
    <h1>This is a video section</h1>
  </header>
  <p>
    <video src="http://www.double.co.nz/video_test/transformers480.ogg"
controls autoplay>
      <div class="no-html5-video">
        <p>This video will work in Mozilla Firefox or Google
Chrome only.</p>
      </div>
    </video>
  </p>
</section>
```

As you can see, you first create a new section where the video will be shown on the page. This has a header followed by the video itself. The video I'm using here is a trailer for the first Transformers movie and is loaded from an external Web site using the `src` attribute. The `controls` attribute tells the browser to display the browser's standard controls for video playback. And the `autoplay` attribute, well, tells the browser to start playing the video automatically.

Between the opening and closing <video> tags I have created a <div> element with the class `no-html5-video`, which will display a message to users who try to view the video in a browser that does not support the <video> element or does not support the Theora video codec.

Before you go and fire up the page in your browser, let's add some new rules to the styles.css page. Just add the rules from [Listing 5](#) to the bottom of the file.

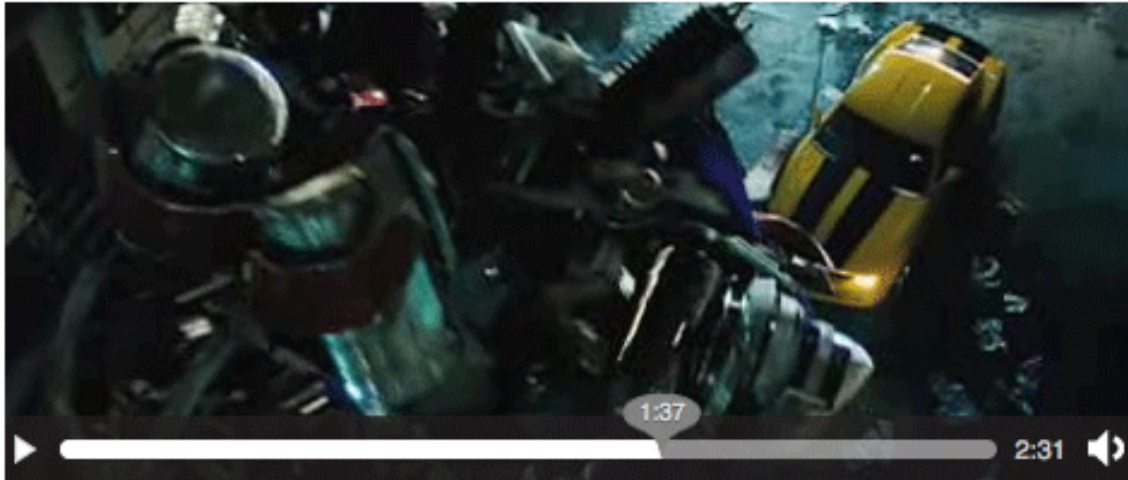
Listing 5. CSS rules for the video section

```
article > section video {
  width: 480px; height: 200px;
}
article > section div.no-html5-video,
article > section div#no-canvas {
  width: 480px; height: 40px; border: 1px solid #993333;
  text-align: center; color: #993333; font-size: 13px;
  font-style: italic; background-color: #F7E9E9;
}
```


These rules simply define the size of the video container element, as well as the error message for those visitors using browsers that do not support HTML5 video or the Theora format. If you open the page in Firefox or Chrome you should see something like the following (see [Figure 3](#)).

Figure 3. HTML5 Video in action

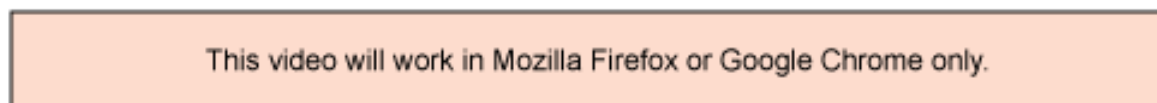
This is a video section



Neat, huh? Now try opening it in a browser like Internet Explorer or Opera. You should see an error like the one shown below in [Figure 4](#). Note that if you try to view it in Safari it may show the browser video player, but it will not play the video.

Figure 4. No HTML5 video support error

This is a video section



Next, you'll create a form that has some of the new Web form features such as placeholder text, autofocus, and the new input types such as range, datetime, and email.

Web form enhancements

One of the more underrated aspects of HTML5 is the introduction of several new form controls that will make developers' lives much easier when creating forms-based Web applications. At present, support for these new controls in terms of

browser widgets and functionality is fairly sparse, but they degrade gracefully as regular text boxes so you can safely use them in your code now, and when browser support improves, the features will automatically enable.

Add the code from [Listing 6](#) directly below the closing `</section>` tag for the video section you created in [Listing 4](#).

Listing 6. Adding a Web form

```
<section>
  <header>
    <h1>This is a feedback form</h1>
  </header>
  <p><form>
    <label for="contact_name">Name:</label>
    <input id="contact_name" placeholder="Enter your name"
autofocus><br />
    <label for="contact_email">E-mail:</label>
    <input type="email" id="contact_email" placeholder="Enter
your email address"><br />
    <label for="contact_phone">Phone:</label>
    <input type="tel" id="contact_phone" placeholder="Enter your
phone number"><br />
    <label for="contact_callback">Callback on:</label>
    <input type="datetime" id="contact_callback"><br />
    <label for="contact_priority">Priority:</label>
    <input type="range" min="1" max="5" value="1"
id="contact_priority"><br /><br />
    <input type="submit" value="Request Call">
  </form></p>
</section>
```

The first form element in [Listing 6](#) does not have any `type` attribute, and as a result it will default to a text control. You'll notice that this has the placeholder text "Enter your name" and is set to `autofocus`. Supporting browsers will automatically switch focus to this element when the page has been rendered.

The next element is of type "email" and again contains a placeholder text value. The only browser that recognizes this type of input element as anything special is Opera, which displays a mail icon in the field to signify that it's an email field.

The next field of interest is the `datetime` field with the label "Callback on:". In supporting browsers, this will display a date and time picker. In Opera, this displays as two controls, a textbox with a datepicker and a spinner for the time.

Finally, you'll see a control of the type `range`, with `min`, `max`, and `value` attributes set. This control will be rendered by Safari, Chrome, and Opera as a slider, with a minimum value of 1, selected by default, and a maximum value of 5. Unsupported browsers will simply display this as a textbox with the text value set to 1.

Next, let's add some flair to what is otherwise a fairly dull and boring form. Add the following rules (see [Listing 7](#)) to your `styles.css` file.

Listing 7. Web form CSS rules

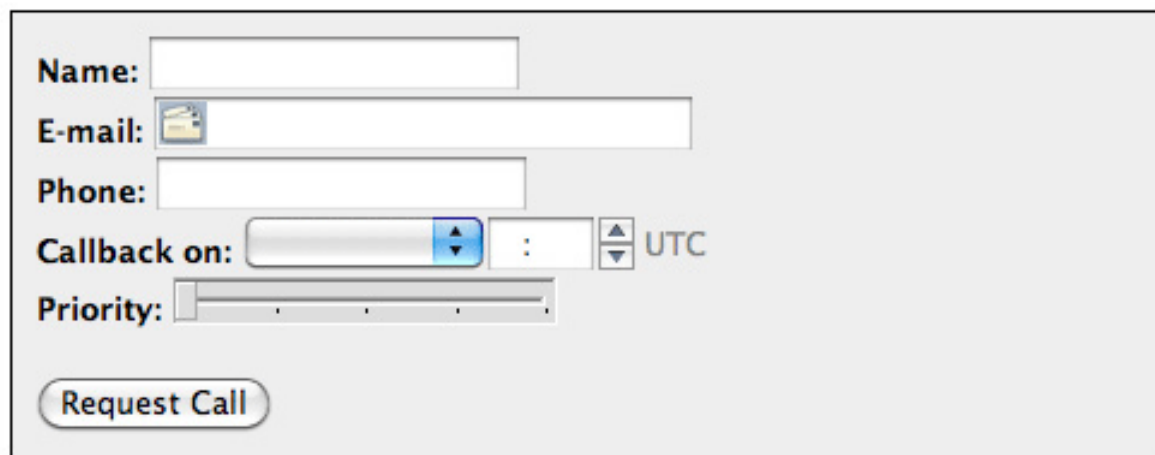
```
article > section form {  
  border: 1px solid #888;  
  -moz-border-radius: 10px;  
  -webkit-border-radius: 10px;  
  border-radius: 10px;  
  -moz-box-shadow: 10px 10px 5px #888;  
  -webkit-box-shadow: 10px 10px 5px #888;  
  box-shadow: 10px 10px 5px #888;  
  background-color: #eee;  
  padding: 10px; margin-bottom: 30px;  
}  
  
article > section label {  
  font-weight: bold; font-size: 13px;  
}  
  
article > section input {  
  margin-bottom: 3px; font-size: 13px;  
}
```

Looking at [Listing 7](#) in more detail, you'll see that a border radius has been added to the form container to round the box corners. In addition, a box shadow has been added to this element as well as a soft grey background color and some padding.

First, let's look at the form as it appears in Opera, the browser that includes the most support for these new HTML5 Web form input types (see [Figure 5](#)).

Figure 5. Web form enhancements in Opera

This is a feedback form

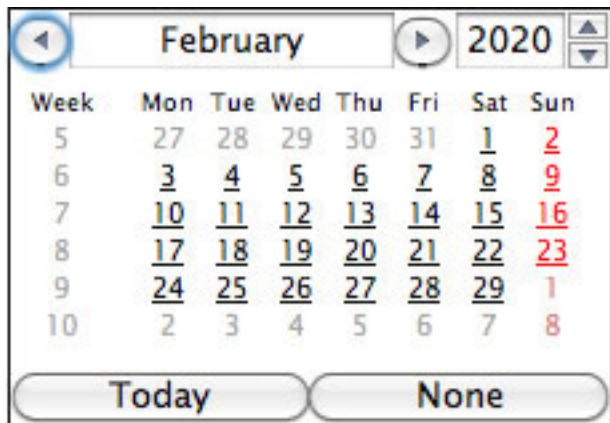


The screenshot shows a web form titled "This is a feedback form" within a browser window. The form is styled with rounded corners, a light grey background, and a subtle box shadow. It contains the following fields and controls:

- Name:** A standard text input field.
- E-mail:** A text input field with a small envelope icon to its left.
- Phone:** A standard text input field.
- Callback on:** A datetime-local input field with a blue spinner icon, followed by a colon, another spinner icon, and the text "UTC".
- Priority:** A range input field with a horizontal slider and tick marks.
- Request Call:** A rounded button with a blue border and the text "Request Call".

You can see the mail icon next to the email field, and the new controls for the datetime field. Clicking the spinner drop-down field opens Opera's calendar widget, as shown in [Figure 6](#) below.

Figure 6. Opera's calendar widget



You can also see a rather unattractive slider control as Opera has rendered it. Frustratingly, however, Opera does not support the border-radius or box-shadow properties, and as a result, neither of these effects has been applied to the form. Let's open it in Safari now to see what these CSS3 effects look like (see [Figure 7](#)).

Figure 7. Web Form enhancements in Safari

This is a feedback form

The WebKit-based Safari has correctly rendered the rounded corners and the box shadow effect, and it has a stunning but simple visual effect. The same effects would work in Mozilla Firefox and Google Chrome. You may notice that Safari supports the placeholder text feature and also implements a slider (a far prettier alternative to Opera's, it must be said) in place of the range input type. In the final section of this tutorial, you will learn how to use the `<canvas>` element and the 2D drawing API to create a simple smiley face bitmap image.

The `<canvas>` element and the 2D drawing API

To get started with `<canvas>`, you first need to add the element itself to the page. Directly beneath the code you added to `index.html` for the Web form in the previous section, add the code from [Listing 8](#).

Listing 8. Adding the `<canvas>` element to the page

```
<section>
  <header>
    <h1>This is an 2D Canvas section</h1>
  </header>

  <p>
    <canvas id="my_canvas" width="480" height="320"></canvas>
    <div id="no-canvas">
      <p>You need a Canvas-compatible browser to view this area.</p>
    </div>
  </p>
</section>
```

This code is relatively straightforward. You simply declare the canvas element, giving it a width, height, and a unique ID attribute. Mozilla and WebKit browsers have different opinions on whether the `<canvas>` tag should have a separate closing tag, but both will be rendered the same way so long as you do not place any code between the opening and closing tags. You will use JavaScript to display the no-canvas error message to those users who are using an unsupported browser in a moment.

Next, you need to add the JavaScript code that harnesses the 2D drawing APIs to create the image on the canvas. In the `index.html` file, just before the closing `</head>` tag near the top of the file, add the following code (see [Listing 9](#)):

Listing 9. Scripting the canvas element

```
<script>
function loadCanvas() {
  var canvas = document.getElementById('my_canvas');
  if(canvas.getContext) {
    var ctx = canvas.getContext('2d');
    ctx.beginPath();
    ctx.arc(245,160,150,0,Math.PI*2,true);
    ctx.fillStyle = "rgb(255,255,204)";
    ctx.fill();
    ctx.stroke();

    ctx.fillStyle = "black";
    ctx.beginPath();
    ctx.arc(200,110,15,0,Math.PI*2,true);
    ctx.fill();

    ctx.beginPath();
    ctx.arc(280,110,15,0,Math.PI*2,true);
    ctx.fill();

    ctx.beginPath();
    ctx.arc(240,170,20,4,Math.PI*2,true);
    ctx.stroke();
  }
}
```

```

        ctx.beginPath();
        ctx.arc(240,190,60,0,Math.PI,false);
        ctx.stroke();
    } else {
        document.getElementById('my_canvas').style.display = 'none';
        document.getElementById('no-canvas').style.display = 'block';
    }
}
</script>

```

Listing 9 adds a block of JavaScript to the page's head element, defining a function named `loadCanvas`. This function instantiates a variable named `canvas` by selecting the element with the unique ID `my_canvas`. It then uses an if statement to determine if it can get a canvas context from this element. If it can, then the `<canvas>` element is supported by the browser; if not, there is no support for the browser and the canvas should be hidden and replaced by the error message. This is taken care of in the else block of the if statement, towards the end of the function.

If canvas supported is detected, the function gets a 2D context named `ctx`, which is then used to draw shapes on the canvas. In this function, five shapes are created. First, the `ctx.beginPath()` function is called to start a path drawing. Next a full circular arc is drawn, with a radius of 150 pixels, near the center of the canvas, using the `ctx.arc()` function. The fill style is set to a yellow color, before the `ctx.fill()` and `ctx.stroke()` functions are called, creating a large yellow circle with a black outline. This is the main face.

The fill style is set to black as the next shapes to be drawn are the eyes. This time full circles are created using `ctx.arc()` which have a radius of just 15 pixels, and the `ctx.fill()` function is called to draw them.

The next block of code creates the nose, which is an arc that is not a full circle and is placed below and centered between the eyes that were just created. The nose is not to be filled, so the `ctx.stroke()` function is used instead.

Finally, a large semi-circle arc is stroked below the nose to create the mouth. The result should look the same in all browsers that support the `<canvas>` element (most modern browsers including Firefox, Safari, Chrome, and Opera—even Internet Explorer—can support it with the help of some clever JavaScript libraries).

The final change you need to make to `index.html` is to change the `<body>` element to call the `loadCanvas()` function when the window has finished loading. Simply replace the opening `<body>` tag with the following line: `<body onload="loadCanvas();" >`.

Finally, just add the following rules to the `styles.css` file to add some pretty box shadowing to the canvas container (see [Listing 10](#)).

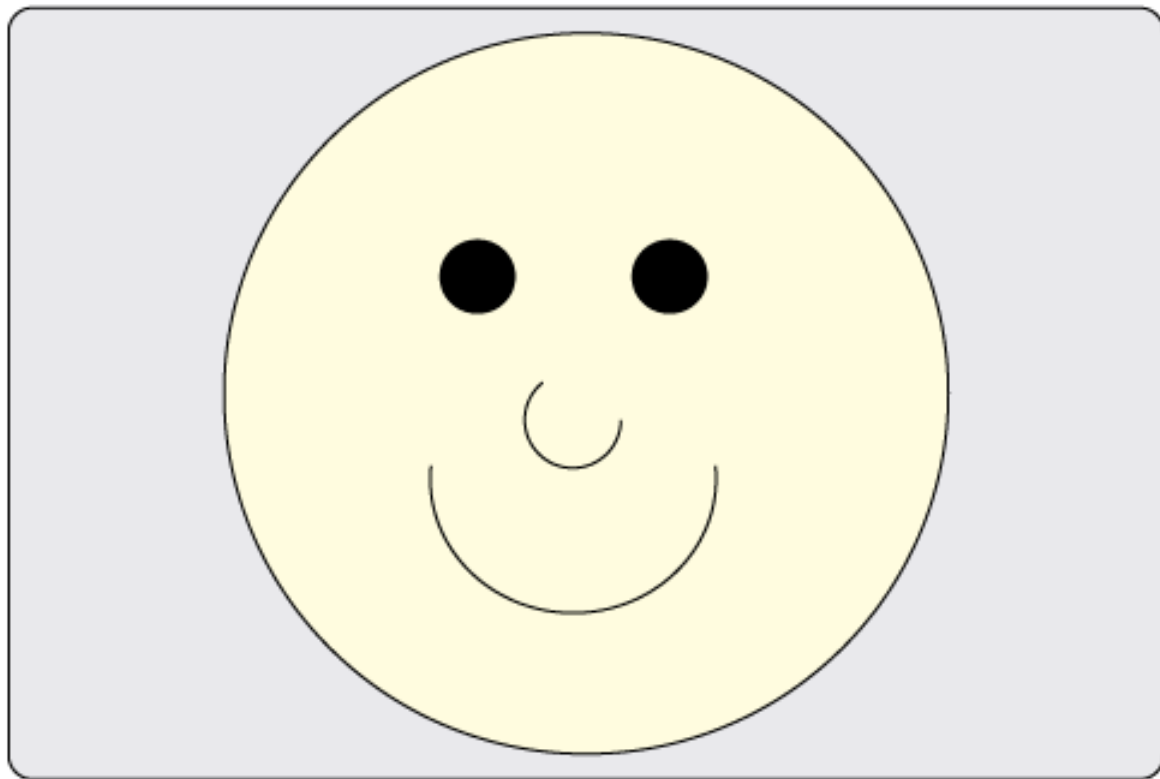
Listing 10. Canvas CSS rules

```
article > section div#no-canvas {  
    display: none;  
}  
  
canvas {  
    border: 1px solid #888;  
    -moz-border-radius: 10px;  
    -webkit-border-radius: 10px;  
    border-radius: 10px;  
    -moz-box-shadow: 10px 10px 5px #888;  
    -webkit-box-shadow: 10px 10px 5px #888;  
    box-shadow: 10px 10px 5px #888;  
    background-color: #eee;  
}
```

The final result looks like the image in [Figure 8](#).

Figure 8. The smiley face canvas

This is a 2D Canvas section



That concludes the sample page for this tutorial. In this section, you have used HTML5 and CSS3 to work with the new HTML5 semantic elements, harnessed some of CSS3's pretty new effects, watched video in the browser without any Flash plug-in, seen some new form widgets, and created a smiley face graphic on a canvas using JavaScript APIs.

Section 5. Summary

This tutorial serves as a hands-on introduction to HTML5 and CSS3 development. HTML5 is very much at an early stage of development, and it will be interesting to see how the new features it proposes are adopted by the different browser vendors. At present, of the major browsers, Opera, Safari, Firefox, and Chrome are providing support for more enhancements with each release, and one would hope to see the bulk of HTML5 features supported by the end of 2010.

Several issues may stop HTML5 from becoming widespread in the near future, however. The first real issue is the lack of support of it from Microsoft's Internet Explorer, the most widely used Web browser on the market. It is unlikely that developers will be able to test any HTML5 features on IE until the first beta version of IE9 is released. At least for now, sites developed for HTML5 degrade quite gracefully on IE8, and with a bit of extra work, fallbacks can be put in place to provide workarounds for IE users.

Another major issue is the one surrounding video codecs and containers. The way things stand, the <video> element will not replace Flash video as the video standard for the Web. With different browsers backing different codecs, it's still much easier to use Flash than it is to encode your videos for Theora and H.264. Here's hoping that some kind of breakthrough is made this year on HTML5 video. In summary, HTML5 and CSS3 are exciting standards, and you can start future-proofing your Web sites to be compliant with these new specifications right now. Following the steps outlined in this tutorial, you should be well versed to move forward and explore some of the other interesting features HTML5 has to offer.

Downloads

Description	Name	Size	Download method
HTML5 examples	html5.examples.zip	3KB	HTTP

[Information about download methods](#)

Resources

Learn

- "[HTML V5 and XHTML V2](#)" (Adriaan de Jonge, developerWorks, November 2007): While the intention of both HTML V5 and XHTML V2 is to improve on the existing versions, the approaches the developers chose to make those improvements is very different. And with differing philosophies come distinct results. For the first time in many years, the direction of upcoming browser versions is uncertain. Uncover the bigger picture behind the details of these two standards.
- "[New elements in HTML5](#)" (Elliott Rusty Harold, developerWorks, August 2007): Get an overview of some of the new elements in HTML 5.
- "[An Introduction to MathML](#)" (David Carlisle, developerWorks, December 2009): MathML is a W3C Recommendation defining an XML vocabulary for marking up mathematical expressions. With HTML 5, it will hopefully be possible to place MathML directly on the Web without needing to worry about mime types and server configurations.
- "[Android and iPhone browser wars, Part 1: WebKit to the rescue](#)" (Frank Ableson, developerWorks, December 2009): This article is the first in a two-part series on developing browser-based applications for iPhone and Android.
- "[The future of HTML, Part 1: WHATWG](#)" (Ed Dumbill, developerWorks, December 2005): In this two-part series, Edd Dumbill examines the various ways forward for HTML that Web authors, browser developers, and standards bodies propose.
- This [HTML tutorial](#) from [w3schools.com](#) teaches you everything about HTML.
- This [CSS tutorial](#) teaches you how to use CSS to control the style and layout of multiple Web pages all at once.
- [JavaScript tutorial](#): Learn how to use the scripting language of the Web.
- [HTML5: A vocabulary and associated APIs for HTML and XHTML: Editor's Draft 13 January 2010](#): Read the most recent Editor's Draft for the HTML5 specification.
- Read excerpts from the upcoming [Dive Into HTML5](#) by Mark Pilgrim from O'Reilly Media.
- [HTML5 gallery](#) is a showcase of sites using HTML5 markup.
- [HTML5 Tag Reference](#): See an alphabetically-ordered reference for HTML5 tags.
- [HTML5 doctor Web site](#) is loaded with information to help you implement HTML

5 today.

- The [HTML 5 Cheat Sheet](#) lists all currently supported tags, their descriptions, their attributes, and their support in HTML 4.
- [CSS 3 Cheat Sheet](#): Here are the main features of CSS 3 in a handy, printable reference card.
- [HTML5 demos Web site](#) has links to various HTML 5 experiments and demos.
- This demo demonstrates the [potential of rendering 3D graphics in the browser](#), using O3D, an open-source web API for creating rich, interactive 3D applications in the browser.
- Read [How HTML5 Will Change the Way You Use the Web](#) from [lifehacker.com](#).
- [Yes, You Can Use HTML5 Today!](#): Another excellent introductory article to the world of HTML5.
- [css3.info.com](#): Everything you need to know about CSS3.
- [Introduction to CSS3](#): Read the W3C Working Draft of this upcoming specification.
- [Introducing the CSS3 Multi-Column Module](#) looks at the W3C multi-column module. The module's intent is to allow content to flow into multiple columns inside an element.
- Learn about [CSS3 Columns](#) from Mozilla.org.
- [CSS3 transitions and 2D transforms](#): This article teaches you about CSS3 transitions and transforms in Opera. See the SVG and SMIL corollaries to them, too.
- This [Canvas tutorial](#) from Mozilla.org describes how to implement the <canvas> element in your own HTML pages.
- [Using the Canvas](#) from Apple Developer Central looks at Safari, Dashboard, and WebKit-based applications support for the JavaScript canvas object, which allows you to easily draw arbitrary content within your HTML content.

Get products and technologies

- [Theora](#) is a free and open video compression format from the [Xiph.org](#) Foundation.
- [O3D AP](#): O3D is an open-source Web API for creating rich, interactive 3D applications in the browser.
- Download [IBM product evaluation versions](#), and get your hands on application development tools and middleware products from DB2, Lotus, Rational, Tivoli, and WebSphere

- Innovate your next open source development project with [IBM trial software](#), available for download or on DVD.

Discuss

- Discuss anything and everything on developerWorks on the [developerWorks blog collection](#).

About the author

Joe Lennon

Joe Lennon is a 24-year-old software developer from Cork, Ireland. Joe is author of the forthcoming Apress book [Beginning CouchDB](#), and has contributed several technical articles and tutorials to IBM developerWorks. In his spare time, Joe likes to play football (soccer), tinker with gadgets, and work on his Xbox 360 gamerscore. He can be contacted via his Web site at www.joelennon.ie.