

# HTML5

## IN ACTION

Rob Crowther  
Joe Lennon  
Ash Blue



MEAP

 MANNING



**MEAP Edition  
Manning Early Access Program  
HTML5 in Action version 5**

Copyright 2012 Manning Publications

For more information on this and other Manning titles go to  
[www.manning.com](http://www.manning.com)

# Table of Contents

## **PART 1: INTRODUCTION**

*1. HTML5: From documents to applications*

## **PART 2: BROWSER BASED APPS**

*2. HTML5 forms*

*3. Creating desktop-style web applications*

*4. Messaging*

*5. Mobile and offline Web applications*

## **PART 3: INTERACTIVE GRAPHICS, MEDIA, AND GAMING**

*6. 2D Canvas*

*7. SVG*

*8. Video and Audio*

*9. WebGL*

*10. The future of HTML5*

# 1

## *HTML5: From documents to applications*

This chapter covers

- What is HTML5 and what's new in this version
- How you can use HTML5 right now
- Tools to make HTML5 development easier
- New HTML elements, and how to use them

HTML5 is one of the hottest topics in Web development right now, and with just cause. Not only is it the latest version of the markup language for the Web, it also defines a whole new standard for developing *applications* for the Web. Previous iterations of HTML (and its rigid XML-based sibling, XHTML) have very much been centered on the concept of HTML as a markup language for documents. HTML5 is the first version that embraces the Web as a platform for Web application development, defining not only a series of new elements that can be used to develop rich internet applications, but also a range of standard JavaScript APIs for browsers to implement natively. A good example of this is the new `<video>` element, which provides a means of playing video content in the browser, without the need for an additional plug-in. Not only does HTML5 provide you with a means of including the video content on the page, but it also defines a series of JavaScript APIs that allow you to control its playback. You can create games. Build mobile apps. And much, much more.

In this chapter, you will learn about all of the great new features introduced in HTML5. You'll then dive right in and discover how you can use it in your Web applications right away, meanwhile learning how to provide fallbacks or workarounds for those users with older or incompatible browsers. Finally, you'll learn about the new elements introduced in HTML5, particularly those that aim to improve document semantics. You'll also learn about how to

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

use ARIA roles, microdata and microformats to further enhance the semantics of your HTML pages. By the end of the first chapter, you should have a broad sense of what HTML5 has to offer, how to use it gracefully in your applications, and what each of the new elements actually does.

## **1.1 *What's new in HTML5?***

From a functionality perspective, HTML5 includes a huge number of improvements over HTML 4.01 and XHTML. Rather than focus solely on the markup language itself, the HTML5 specification also standardizes an array of JavaScript APIs, helping Web application developers build software that will work consistently across a variety of browsers and platforms.

In this section, you will discover some of the exciting new features HTML5 introduces. Firstly, you will learn about the improved document semantics that HTML5 enables via a series of new semantic elements. Second, you will be introduced to the new form input elements and attributes that allow users to easily enter different types of data. Next, you will briefly discover the new `<canvas>` and multimedia elements that make HTML5 a viable replacement for native animations, games and multimedia content. Finally, you will learn about various JavaScript APIs that allow you to build feature-rich client applications with `localStorage`, offline support, drag and drop and more. Many of the features covered in this section are already implemented in at least one of the most popular Web browsers, and you will be able to try them out for yourself over the course of reading this book.

### **1.1.1 *Better document semantics***

Not so long ago, Web developers used complex table structures to define layouts for their Web pages. HTML tables, however, were never designed to be used for layout purposes, and it was semantically incorrect to use them in this way (not to mention the performance implications for rendering layer after layer of nested tables). With the introduction of Cascading Style Sheets, developers turned to clever styling of elements like `<div>` to structure a page's layout, ensuring that tables could be left for representing tabular data.

In turn, this led to pages that contained an abundance of `<div>` elements, typically given an ID or class to indicate what section of the page it represented. Ian Hickson of Google, the editor of the HTML5 specification, did some analysis on which class names are used on the most Web pages. The results can be found at <http://code.google.com/webstats/2005-12/classes.html> and make for interesting reading.

HTML5 aims to improve the semantics of Web documents by introducing a series of new elements that can be directly used in place of the `<div>` element. Rather than authors picking and choosing their own class or ID values to give to sections of the page, they will use a standard set of elements, making it easier for applications like Web spiders and search crawlers to understand the different parts of the page, allowing them to provide better search results. Interestingly, many of the top 20 class names in the aforementioned Google

study are catered to by these new elements. You will learn much more about the new semantic elements later in this chapter.

### 1.1.2 Improved web forms

It rarely receives acclaim, but the humble Web form has played a major role in the emergence of the Web as a platform for application development. Web applications chiefly rely on the ability to accept user input and store this data in a database – and without form elements, this would not be possible.

As HTML5 is highly focused on Web applications, there are many improvements in Web forms, all of which can be used without breaking compatibility with older Web browsers. The basic text field has been used far beyond its primitive capabilities, and HTML5 aims to ease the burden by offering a series of compatible types, each of which provides enhancements over the simple text field. Table 1.1 identifies the new input types that are available.

**Table 1.1. The new form input types introduced in HTML5**

color	date	datetime	email	localtime
month	number	phone	range	search
time	url	week		

You can use all of these new input types in your Web pages right now, as older browsers will fall back to a standard text input type where it finds a type it doesn't understand. Some of the new input types will allow browsers to provide standard widget controls for given types of form field. For example, the "date" type should display some form of popup date picker. In fact, this feature is already available in the Opera browser, albeit with a horrendously ugly control (see figure 1). The "color" type should offer a color picker, like you might find in a graphics manipulation application. Once again, browser adoption of this feature has been slow, although Opera have supported it since version 11. Later in this chapter you will learn about Modernizr, an HTML5 feature detection script. Using Modernizr, you will be able to detect if a browser supports a given input type, providing a fallback JavaScript-based widget if required.

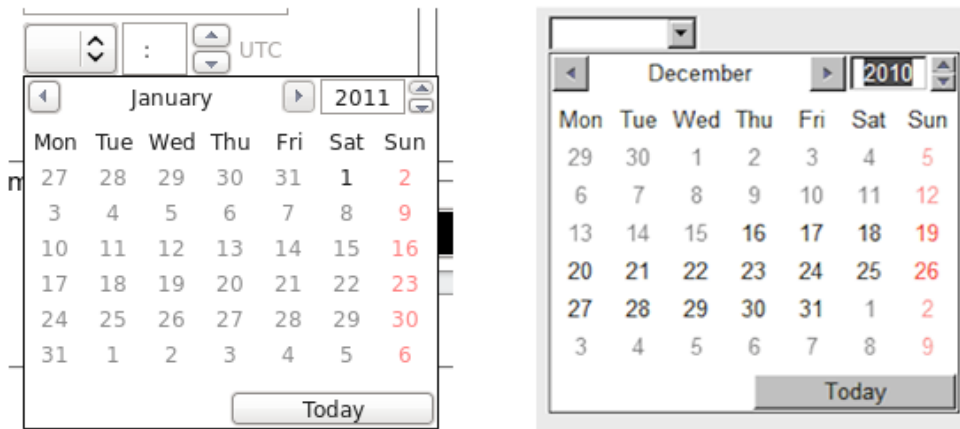


Figure 1.1. The HTML5 date widget in Opera running on Linux (left) and its ugly Mac OS X version (right)

In addition to new form field types, HTML5 also introduces a series of new common attributes that allow you to alter the behavior of a given field. For example, the new “placeholder” attribute can be used to provide a piece of text that should be displayed in a field when it is empty and does not have focus (see figure 2). This text is typically gray, and will be removed when the field is not empty or has focus.



Figure 1.2. HTML5 placeholder attribute in Opera

Table 1.2 is a complete list of the new attributes. You will learn about each of these attributes in detail, complete with usage examples, in Chapter 2, “HTML5 Forms”.

Table 1.2. New input element attributes introduced in HTML5

autocomplete	autofocus	list	max	min
multiple	pattern	placeholder	required	step

Finally, HTML5 also introduces validation for form fields that does not require JavaScript. For example, if you have an email field, the browser will validate that the text entered by the user is a valid e-mail address. Of course, although this is not JavaScript validation, it is still client-side, and should never be trusted – you must still validate all your form submissions

on the server-side. You will learn about form validation and all of the new Web form features in great detail in Chapter 2, “HTML5 Forms”.

### 1.1.3 A new canvas for the Web

HTML provides a ton of different elements to allow you to present information on a Web page. These can be styled in many different ways, and you can use JavaScript to animate them and apply dynamic effects. If you are comfortable with complex JavaScript code (and are comfortable that your users will have a high-performance browser), you can even do pretty amazing things with simple HTML and JavaScript.

The problem is, there are many things that designers and developers may want to implement that HTML just doesn't cater to. What if you want to insert a circle, square or other shape? What if you want to display an image and dynamically alter it based on user selections, on-the-fly? Of course, you could use static images or a server-side solution, but these are less than ideal in reality. The only viable solution in the past was to use a third-party plugin such as Adobe Flash.

Thankfully, HTML5 introduces the `<canvas>` element and a series of related drawing APIs that will allow you to do some amazing things without requiring the user to have a particular plugin installed. The `<canvas>` element is very well described by its name – it quite simply is a canvas for your Web pages. In figure 1.3 is a screenshot of a Breakout-style game, that was created entirely in HTML5 and JavaScript, with the game's visuals output on a `<canvas>` element. Pretty neat, huh? Even neater is that you will learn how to build this game yourself in Chapter 6, “2D Canvas”.



Figure 1.3. `<canvas>` allows developers to create some amazing things. You will learn how to build this Breakout-style game in Chapter 6, “2D Canvas”

Chapter 7 follows on from this by describing how to build a game using Scalable Vector Graphics (SVG) and JavaScript, illustrating that the Web is on the path to being a viable platform for the development of games. Chapter 8 takes things a step further by introducing the 3D canvas and OpenGL, showing you how to create a simple 3D world that you can navigate around using your browser.

Canvas is an exciting aspect of HTML5 – it opens up the Web to a huge number of possibilities. In chapter 6 you will learn how to use `<canvas>` in your own applications. With this information, and your imagination, the sky really is the limit when it comes to what you can achieve.

### ***1.1.4 Native browser video and audio (with no added sugar)***

One of the biggest growth areas on the Web in recent years has been streaming video, and to a lesser extent, audio. Both of these technologies are nothing new, and thanks to multimedia plugins like Shockwave, Flash, RealPlayer, Windows Media Player, QuickTime and others, the Web has been a viable medium for multimedia delivery for well over a decade.

With all of that said, restrictions on connection speeds and bandwidth severely limited the usefulness of such technology, restricting it to low quality streams and short clips. When MP3 came along and miniaturized the file size of high quality audio files, it changed the landscape of the music industry, causing the consumer market to shift towards digital as the industry stumbled to try and prevent a wave of mass piracy from crippling their profits.

Next, along came YouTube, which, coupled with the increasingly growing level of availability of high speed Internet access, led to a revolution in how people watched video. Flash-based video streaming took the Web by storm, and continues to lead the way today. What started off as a platform for self-publishing has moved to the streaming distribution of movies and TV shows via sites like Netflix and Hulu. With Internet access speeds climbing at a fast pace, streaming high definition (HD) content over the Web is steadily becoming the minimum acceptable standard.

All of this has been made possible thanks to browser plugins. Today, the majority of Web video is deployed in Flash video (FLV) format, an Adobe Flash container for various types of video codec. If the end user has a Flash plugin installed, they can view the video. Some have raised questions about the security and performance of Flash as a platform for video delivery, however, and are looking for alternative solutions. HTML5 provides such a solution through the new `<video>` and `<audio>` elements, which allow supported multimedia files to be played back natively by the browser (see figure 1.4 for a YouTube HTML5 video of Gmail Motion, a pretty funny April Fool's prank by Google), and controlled via a series of JavaScript APIs. This means that visitors can now view video content on your website, even if they don't have third-party plugins installed. This is particularly important if you want your content to be accessible by people using a device such as an iPhone or iPad, which do not support Flash at all.

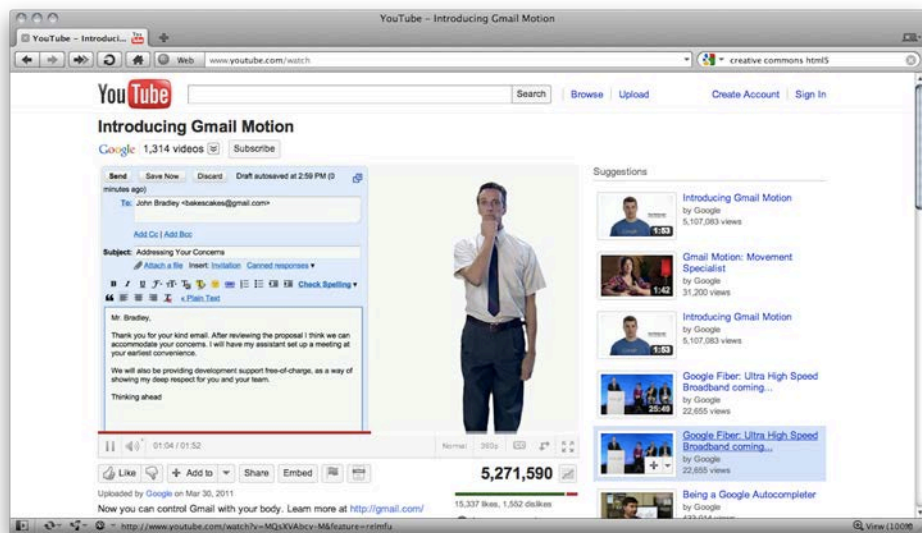


Figure 1.4. YouTube HTML5 video in action. The YouTube video in this screenshot does not use the Adobe Flash plug-in, but is fully implemented using the HTML5 <video> element and related APIs

HTML5 multimedia support is not limited to video content – the new <audio> element provides a similar feature set for audio playback. You will learn all about HTML5 video and audio in Chapter 8, “Video and audio”.

### 1.1.5 Offline applications in an online world

One of the problems with Web applications is their dependence on being connected to the Internet at all times. While you can save Web pages for use offline, they are only usable if they are static and not reliant on a back-end database. Otherwise, you might be able to view the Web page, but as soon as you try to perform a query or update that requires server-side support, you'll run into trouble. In recent years, a number of solutions for working offline have emerged, including Google's Gears client-side database. Web applications that support Gears allow a user to use the application when offline, storing data locally, and when a connection becomes available, synchronize this data back to the main server. Of course, the problem here is that this offline support is completely dependent on the user having Google Gears installed, which the majority of Web users do not.

In HTML5, there are several new features that provide functionality allowing developers to enable access to their applications offline. First up is client-side data storage via the local storage and session storage APIs. This allows an application to store a reasonable amount of key-value pair data on the client-side, retrieving it as required. With the local storage API, the data will be kept on the client until it is either removed by the application or by the user.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

The session storage API allows data to be kept only for as long as the browser session is kept open, it is automatically cleared when the session is closed. By default, browsers will allow an application to store up to 5 megabytes of data on the client-side, after which it will request permission from the user to grant access to more disk space.

In addition to this basic key-value storage mechanism, HTML5 also defines APIs for a more feature-rich client-side database. Originally, this was referred to as Web SQL, and was implemented by various browser vendors through the SQLite database system. The W3C did not want a single database system becoming the de-facto for all browsers, and preferred alternative implementations to be provided, and as a result, Web SQL was scrapped altogether. In its place, the W3C has provided IndexedDB, a document-oriented database API that behaves quite differently to traditional relational SQL-based databases. This new approach will allow an application developer to store complex data structures on the client-side, further enabling powerful offline Web applications.

Client-side storage solves the issue of accessing data while offline, but this is still not very useful unless the application itself is also accessible offline. It would be tedious if a user of your application needed to manually save your application for offline use, and it would become even more problematic when it comes to delivering updated versions of the application. Thankfully, HTML5 provides support for offline applications using an application cache manifest file and API. To use this functionality, your Web application must provide a manifest file, outlining the URIs (Uniform Resource Identifiers) that should (and should not) be cached by the browser for offline use. You can even define fallback files that should be served up in the case that a particular file is attempted to be accessed, enabling you to allow some parts of the application to be used offline, while requiring the user to be online for other parts. For example, you might have a word processing application, where the user can create and save documents offline, but in order to store these documents in the cloud or share them with others, they need to be working online. In this case, you would use localStorage to save the data on the client-side, using a cache manifest to ensure that the HTML and JavaScript files can be used offline. Any APIs that require online use could have a fallback that informs the user that they need to connect to the Internet in order to use that particular feature.

You will learn much more about local storage, session storage, IndexedDB and application cache in Chapter 5, "Offline and mobile applications".

### **1.1.6 *Look ma, no jQuery!***

Because previous versions of HTML have lacked in any real JavaScript APIs for Web applications, developers have become increasingly dependent on external JavaScript libraries and frameworks such as jQuery, Prototype and so on. While these frameworks provide some excellent features, the increasing over-dependence on them is leading to new JavaScript developers becoming experts in JavaScript libraries rather than in JavaScript itself. This often results in highly inefficient coding practices and a lack of basic understanding in the principles of JavaScript on behalf of JavaScript developers.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

The title of this section may be a little misleading – HTML5 is unlikely to eradicate the need and dependency on the likes of jQuery – but it does promise to at least make some of the features available using standardized, native and faster APIs. You can expect that libraries such as jQuery will be modified so that they use HTML5-enabled features by default, and fall back to other techniques for non-HTML5-friendly browsers.

The following HTML5 features are just some of those that would typically have required an external library in the past.

- Drag and drop
- History and state management
- hashchange event
- Basic data storage (HTML5 data-\* attributes)
- Inline editing
- Touch events
- Undo changes

These features (and some other nice new features, such as the File API, and multithreaded Web development using Web Workers) will be covered in more detail in Chapter 4, “Creating desktop style Web applications”.

### ***1.1.7 And if that isn't enough...***

At this point, we're sure you'll agree that HTML5 is bursting with new features and improvements for you to use in your own applications. It is important to remember, however, that the specification is far from finalized at this point, and it is likely to change quite substantially in the near future. HTML5 is regarded by many as not just a Web standard, but a term to describe a new paradigm for Web application development. The likes of Apple commonly refer to HTML5 in a way that describes any Web application that uses HTML5, CSS3, SVG, WebGL and other related technologies. In the final chapter of this book, Chapter 10, “The future of HTML5”, you will discover some features that may find their way into the specification before it is completed, such as the <device> element and the ping attribute, and also some related technologies that can be used in conjunction with HTML5 to provide an even better user experience. In the next section, you will learn how you can start using HTML5 in your Web applications right away, meanwhile ensuring that users with older, non-HTML5 browsers are not left behind.

## ***1.2 Using HTML5 right now***

Despite the HTML5 specification's relative youth in the W3C standards process, Web browser support is improving rapidly. This means that there is no reason why you can't start using HTML5 in your web applications right away. Before jumping in and adding HTML5 features to your heart's content, however, there are a number of things you need to be aware of. This section will guide you through structuring HTML5 documents, providing support for older

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

versions of Internet Explorer, detecting browser support for particular features and introduces a starter template framework that allows you to develop cross-browser HTML5 applications quickly.

### 1.2.1 Basic structure of an HTML5 document

HTML5 documents are structured in the same way as older versions of HTML – you declare a doctype at the top of the document, and open and close the HTML document with a pair of `<html>` and `</html>` tags. Between these tags, you have two distinct sections, a `<head>` section where you place meta information and other non-content items such as stylesheets, and a `<body>` section where your page content should go. If you've written HTML pages or applications before, none of this will be new to you, but there are some subtle differences that you need to be aware of, which we will cover in this section. These are the HTML5 doctype declaration syntax, how to use the opening `<html>` element, and the shorter versions of the various elements you can use in the `<head>` section.

#### DOCTYPE DECLARATION

The first difference of note is in the doctype declaration on the first line of the document. In previous versions, the (X)HTML doctype contained long DTD (document type declaration) references that were difficult to remember and defined whether the document should be validated using the transitional, frameset or strict version of the HTML syntax. The following code is the declaration for an XHTML 1.0 Strict document:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

If, like us, you are not a fan of this declaration, you'll be delighted to learn that for HTML5 documents the following doctype is valid:

```
<!DOCTYPE html>
```

This is shorter and more memorable, but most importantly, it is enough to force standards mode in all modern browsers, even those that don't support the specific features of HTML5. This means that you can safely start using the HTML5 doctype in your pages right now, and be safe in the knowledge that updating the doctype will not wreak havoc in older browsers.

#### THE `<HTML>` ELEMENT

The doctype isn't where HTML5's preference for concise code ends. If you have created XHTML pages, you may be familiar with the `xmlns` attribute that was added to the `<html>` element, defining the XML namespace that the XHTML document fit into. As a result, the majority of web pages written in XHTML included the following opening `<html>` code:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
```

The `xmlns` attribute is actually invalid in HTML documents, and it was never really needed anyway. If an XHTML document omitted it, the value `http://www.w3.org/1999/xhtml` was used by default. Predictably enough, you should drop this attribute in HTML5 documents.

HTML 4 introduced the `lang` attribute for many elements, including `<html>`. The attribute's purpose was to be used by browsers to determine how to display parts of the page depending on the language specified. In practice, this was rarely used, but the `lang` attribute proved very useful for screen readers and other assistive technology. Take the word

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

“chat” for example, which in English means informal conversation, but in French means cat. This word is also pronounced differently based on what language it is spoken in. Screen readers such as JAWS (Job Access with Speech) will pay heed to the lang attribute of the <html> element when deciding how to pronounce words such as this. As a result, we highly recommend that you use the lang attribute in your HTML5 documents. The following is an example of the recommended opening <html> element for an English document:

```
<html lang="en">
```

The value of the lang attribute should equate to the two-letter ISO language code for the language the document is written in. See <http://reference.sitepoint.com/html/lang-codes> for a complete list of these codes.

### THE <HEAD> SECTION

The <head> section of an HTML document is where you place meta information such as the character set encoding, the document’s title, any stylesheets required for the document’s styling, and depending on your preferences, any <script> tags for your JavaScript code. Nothing’s really changed here, but HTML5 does allow you to be more concise. Take the following example of an XHTML <head> section:

```
<head>
  <title>My Web page</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <link rel="stylesheet" type="text/css" href="style.css" />
  <script type="text/javascript" src="script.js"></script>
</head>
```

This <head> section contains the page’s title, character set, a link to an external CSS stylesheet and a reference to an external JavaScript source file. It is worth pointing out that all of this code is perfectly valid in HTML5, but it is a little more verbose than it need be.

HTML5 introduces a new charset attribute for the <meta> element which allows you to be far more concise when defining the character set your page uses. In addition, because HTML5 is not an XML language, you no longer need to close empty tags like <meta />. You can further reduce the character count by removing the type attributes from the lines that reference the external CSS and JavaScript files. All browsers will assume that a stylesheet is of type text/css unless otherwise specified, and similarly all <script> elements are assumed to be of type text/javascript by default. This leaves you with the equivalent HTML5 <head> section:

```
<head>
  <title>My Web page</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css">
  <script src="script.js"></script>
</head>
```

More concise code is easier to read and maintain, and that is always a good thing. The code above will work fine in all modern Web browsers, so you don’t need to worry about breaking compatibility in your existing documents. If you insist on worrying, fear not, the next section will introduce some problems that will surface with every Web developer’s “favorite” browser.

## 1.2.2 Oops! IE did it again...

As you would expect, any new version of HTML will bring with it a series of new elements that can be used in your documents, and HTML5 is no different. You will learn about these new elements (and which ones have been dropped) in detail later in this chapter. For now, there's a little bit of housekeeping required in order to get all browsers to play friendly with the new elements.

In most modern browsers, all of the new elements will render fine, even in Internet Explorer (IE). However, when you try to style any of these elements, you'll find that IE decides to throw its toys out of the pram and sulk in the corner. Even if you have taken the noble action of de-supporting IE6 in your pages, you'll be shocked to read that IE7 and IE8 also react negatively to styles applied to HTML5-specific elements.

Fortunately, there are a few ways of alleviating this problem. The first option requires the use of JavaScript. If you want to use the element `<header>` on your page and need to apply some CSS styles to it, executing the following JavaScript code in the `<head>` section of your page will force Internet Explorer to apply the CSS rules to the tag, even if the version of IE used does not support a particular element natively:

```
document.createElement("header");
```

You will need to execute this for every HTML5-specific element you wish to use in your page. This will cause IE to render the style correctly, but the problem will persist if you attempt to print the page. Fortunately, a solution known as IE Print Protector can be used to fix the printing issue. Rather than reinvent the wheel, you should use Remy Sharp's HTML5 shiv script to do all this for you (or just use Modernizr, which we will cover in the next section). For more information on Remy's script, see <http://remysharp.com/2009/01/07/html5-enabling-script/>.

The problem with the previous solution is that it requires JavaScript to be enabled for it to work. If your page or application is heavily dependent on JavaScript, this might be an option for you, but if you need to support non-JavaScript visitors, you'll need an alternative solution.

The first non-JavaScript method is pretty ugly, to put it lightly. Basically, it involves serving the new HTML5 elements to anyone using a supporting browser, and serving good ol' `<div>` elements to IE. This can be achieved using conditional comments, which will only be parsed by Internet Explorer, and ignored by all other browsers. By giving the HTML5 and `<div>` element the same class, you will only need one CSS rule, which will apply to both approaches. Let's take a look at this (scary looking) markup.

### Listing 1.1. Using conditional comments to support older versions of IE

```
<!--[if lt IE 9]><div class="header"><![endif]-->
<!--[if (gte IE 9)|(!IE)]><!--><header class="header"><!--<![endif]--> #1
  <h1>This is my header</h1>
<!--[if lt IE 9]></div><![endif]-->
<!--[if (gte IE 9)|(!IE)]><!--></header><!--<![endif]-->
```

Believe it or not, that code is perfectly valid HTML5, and will pass the HTML5 validator at <http://html5.validator.nu> with flying colors. First, the code checks if the visitor is using some IE browser less than version 9, and if this evaluates to true, a <div> element is used. Next, the code checks if the user has IE 9 or later, or is not using IE at all. If this is the case, the browser renders an HTML5 <header> element instead. At this point, you might be thinking – “wait a minute, I thought other browsers don’t read conditional comments? How do they know to use <header>?” Notice how after the opening conditional comment tag we have an additional comment tag that immediately closes? #1 IE sees this as a nested comment and will read the code inside, but other browsers will close the conditional comment altogether, allowing the inner section to be parsed (and validated) correctly. This is repeated after the use of the <header> element to allow for the closing conditional comment tag.

A far less verbose option for enabling HTML5 elements to be styled in IE is to not use HTML5 at all, but rather opt for its less-popular sibling, XHTML5. The benefit of this approach is that all the new features of HTML5 are available to you, while maintaining backward compatibility for styling new semantic elements in Internet Explorer. The primary drawback is that you’ll need to adhere to a strict standard of markup if you want your XHTML5 document to validate. In addition, you’ll need to do some trickery on the server-side to send different MIME types to Internet Explorer and other browsers, which is less than ideal. For an excellent overview of this technique, see Eric Klingen’s post on the subject at <http://www.debeterevormgever.nl/www/html5-ie-without-javascript/>.

### 1.2.3 Feature detection with Modernizr

One of the problems with using HTML5 right now is the lack of consistent browser support for the various features defined in the specification. For example, the new autofocus attribute for input elements works in Firefox 4 but not in Firefox 3.6. Safari 4 did not have support for Web Sockets, these were introduced in Safari 5. With the ever-expanding set of features in HTML5, and the ever-changing state of browser support amongst the major vendors, it would be exhausting trying to maintain a list of which browser supports which feature.

You can use JavaScript to easily detect if the visitor’s browser supports a particular feature. For example, to check if they have support for offline applications, you would use the following code

```
!!window.applicationCache
```

This statement will evaluate to true if the HTML5 application cache is supported, or false if it is not. Unfortunately, not every HTML5 feature is detected in the same way. Local storage is also implemented as a property of the window object. As such, you might expect the following to work:

```
!!window.localStorage
```

This will work in many places, but if you try to use it in a debugging tool like Firebug, it will raise a security exception. Instead, you can consistently use the following statement.

```
'localStorage' in window
```

To detect some features, you have to go to much more trouble than the above approach. Let’s take the canvas element as an example.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

```
!!document.createElement('canvas').getContext
```

This code basically creates a dummy canvas element and calls the `getContext` method on it. The double-negative prefix on this statement will force the result of this expression to evaluate to either true or false, in this case informing you of whether or not the browser supports the canvas element. As a final example, let's look at how you would detect one of the new HTML5 form input element types, in this case the date type:

```
var el = document.createElement('input');
el.setAttribute('type', 'date');
el.type !== 'text';
```

Pretty ghastly stuff, huh? Of course, you could wrap this in a function to make it re-usable, but writing functions for each and every HTML5 feature would be painstaking. Thankfully, there is a JavaScript library named Modernizr that does all this for you.

To use Modernizr, grab the minified JavaScript source file for the library from <http://www.modernizr.com>, and include it in your HTML document by adding it to the `<head>` section, for example:

```
<script src="modernizr-1.7.min.js"></script>
```

You'll also need to add a class attribute to your document's `<html>` element, with the value `no-js`, as follows:

```
<html lang="en" class="no-js">
```

You can now use Modernizr to detect support for various HTML5 features. Let's see how you would use it to detect the four features we detected earlier in this section.

```
Modernizr.applicationcache //true if offline apps supported
Modernizr.localstorage //true if local storage supported
Modernizr.canvas //true if canvas supported
Modernizr.inputtypes.date //true if date input type supported
```

We're sure you'll agree that this is much easier to remember and read. Modernizr also adds a host of CSS classes to the `<html>` element of your document to indicate if a particular feature is available in the visitor's browser. This allows you to serve up different styles to users based on whether their browser supports a given feature. For further information on the Modernizr library, visit the project's website at <http://www.modernizr.com>.

### 1.2.4 HTML5 Boilerplate

If you are building an HTML5 application from scratch, there is quite a lot to watch out for. Ensuring your app is cross-browser compatible, supporting caching in an efficient manner, optimizing for mobile browsers, performance profiling, unit testing, writing printer-friendly styles – these are just a sample of the various complexities that come with the territory when building modern Web applications.

Rather than learning about and catering to all of these issues individually, wouldn't it be nice if you could get up and running quickly using a template that takes care of all of this for you? This is exactly what the HTML5 boilerplate does. The following is just a snippet of the features the boilerplate includes:

- Modernizr
- jQuery (hot-links to a Google-hosted file for performance, with a local fallback)

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

- Optimized code for including Google Analytics
- Conditional comments to allow for Internet Explorer-specific styling
- CSS reset, printer-friendly styles
- Google-friendly robots.txt file
- .htaccess file jam packed with site optimization goodness

We highly recommend using HTML5 Boilerplate as a starting point for all of your HTML5 applications. As the creators of the boilerplate point out, it is delete-key friendly – so if you don't want to include anything that comes as part of the boilerplate, you can simply remove it. The latest version of the project also supports custom builds, allowing you to include only those parts that you really need. For further information and to download the HTML5 Boilerplate, visit the project's website at <http://html5boilerplate.com>.

Now, having covered the various aspects of HTML5 that you can add to your existing Web pages and applications, let's explore the various tools that every HTML5 developer should have installed on their system.

### ***1.3 An HTML5 developer's toolbox***

When developing HTML5 applications, your development environment will primarily consist of a text editor or integrated development environment (IDE), and a Web browser. Every Web developer should at least have a copy of the latest versions of the major browsers:

- Apple Safari
- Google Chrome
- Microsoft Internet Explorer
- Mozilla Firefox
- Opera

Ideally, you'll have a virtualization environment available to you that will allow you to test several versions of each browser on the same machine. For example, you can use applications such as Oracle VirtualBox, VMware Fusion/Player/Workstation or Parallels Desktop to create multiple virtual machines with different versions of browsers installed in various different operating system environments. This is extremely convenient for testing how your application will work in a wide variety of different browser and system configurations.

In addition to installing the major Web browsers, you should also ensure that you have the relevant tools available to make your life easier. All of the major browsers now include a suite of Web developer tools, either as standard or separately via a browser plug-in. These tools are vital when it comes to testing, debugging and analyzing the performance of your Web pages and applications. The features provided by these tools include:

- Console output
- JavaScript debugging
- Element and property inspection
- Network activity and traffic analysis
- JavaScript performance profiling
- On-the-fly element styling and manipulation

In this section, you will learn about the developer tools available in each of the major Web browsers. You will discover how to access Safari and Chrome's Developer Tools, how to download and install Firebug and Web Developer plug-ins for Firefox, how to start the Dragonfly tool in the Opera browser, and finally where you can launch Developer Tools in the various versions of IE.

### **1.3.1 Safari and Chrome Developer Tools**

Both Safari and Chrome include a set of developer tools out-of-the-box that provide all of the aforementioned features. In Chrome, you can launch these tools from the main menu, under Tools > Developer Tools. In Safari, the tools are hidden by default. From Safari's Preferences dialog, check the "Show Develop menu in menu bar" option in the Advanced section. You will then see an additional menu in Safari called "Develop". The "Show Web Inspector" menu item will open Safari's Developer Tools. See figure 1.5 for an example of Chrome's Developer Tools in action.

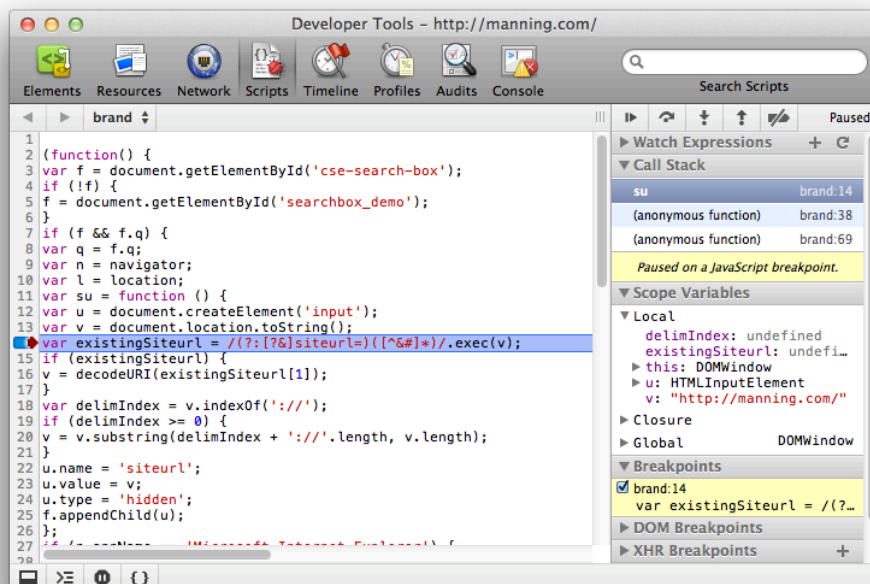


Figure 1.5. Google Chrome Developer Tools showing how you can set breakpoints, watch expressions and change variable values to perform JavaScript debugging directly in the browser

The Developer Tools included with Safari and Chrome are rich in features, including DOM live editing, full JavaScript debugging, network and JavaScript analysis tools, and the ability to view cookie and other data stored on the client-side. The tools can be docked to the bottom of the browser window or opened separately in their own application window. In addition, Safari's Develop menu makes it easy to enable and disable support for the likes of images, JavaScript, style sheets and cookies, allowing you to quickly test how your application works under those constraints.

### 1.3.2 Firebug and Web Developer plug-ins for Firefox

Although Firefox doesn't include a suite of developer tools by default other than a basic error log window, there are several plug-ins available that will add such features to Firefox. The primary developer plug-in for Firefox, Firebug, is actually maintained by Mozilla themselves, and includes most of the features you can find in Safari/Chrome Developer Tools, including console logging and output, live DOM, JavaScript and style editing, JavaScript debugging with support for breakpoints and watch expressions and network analysis tools. You can download Firebug at <http://getfirebug.com/>. Firebug can be seen in action in figure 1.6.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

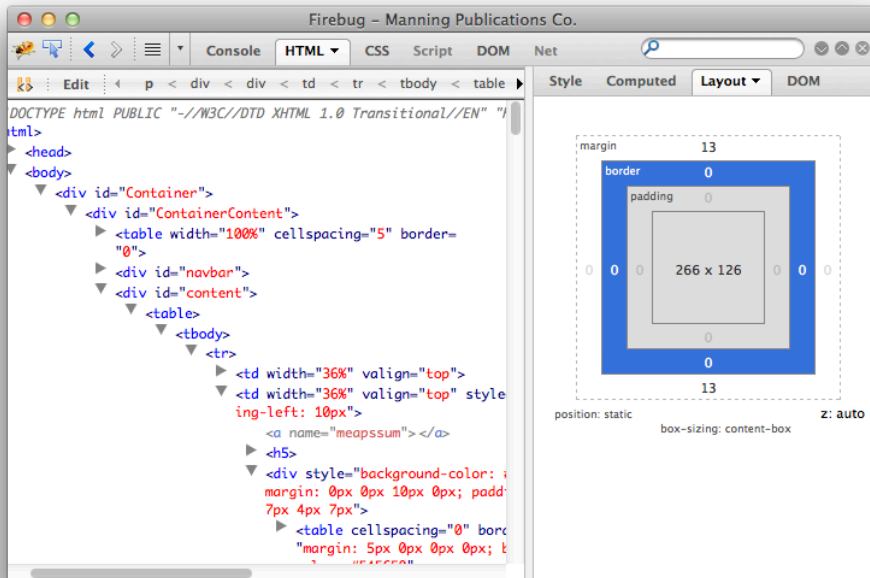


Figure 1.6. Firebug plugin for Mozilla Firefox, illustrating how you can view the HTML source and styling properties for a Web page, which is dynamically updated when the source is modified in JavaScript

Firebug does not include all the tools you might need, for example, it does not include features to view detailed information about cookies, or indeed the ability to easily switch on and off things like images, cookies and JavaScript. Fortunately, all of these features, and more besides, can be added with the Web Developer plug-in. You can download and find out more about the Web Developer plug-in on the Firefox Add-ons website at <https://addons.mozilla.org/en-US/firefox/addon/web-developer/>.

### 1.3.3 Opera Dragonfly

Like Chrome and Safari, the Opera browser also natively includes a suite of Web developer tools, called Dragonfly. You can launch Dragonfly in Opera from the "Tools > Advanced > Opera Dragonfly" menu. Dragonfly can be seen in action in figure 1.7.

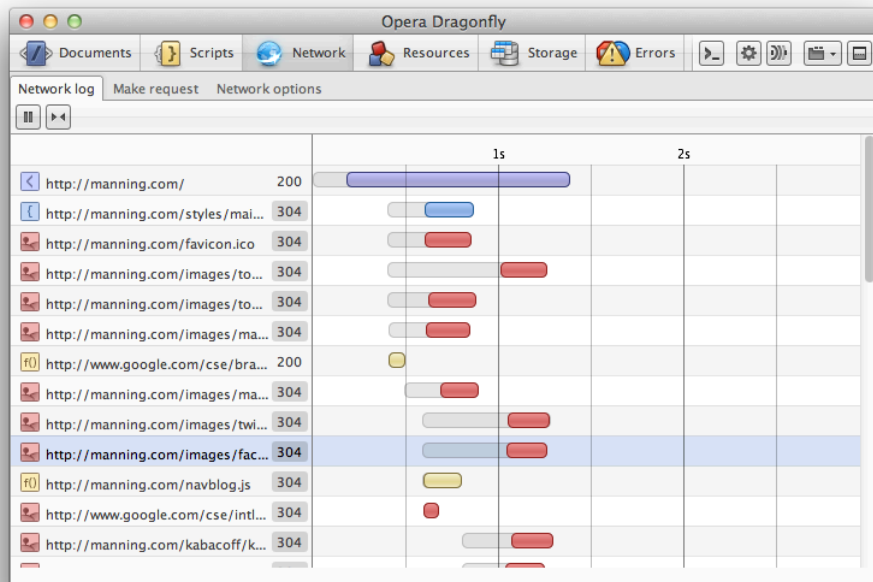


Figure 1.7. Opera Dragonfly displaying a visual representation of the time each resource on a page takes to load. Using this graph, you can easily identify the parts of your application that take the longest to load.

Dragonfly includes most of the features you'll find in Safari/Chrome Developer Tools, but with a few neat additions like the ability to take screenshots and pick colors from Web pages, without requiring additional plug-ins or tools to be installed.

### 1.3.4 Internet Explorer Developer Tools

As of version 8, Internet Explorer includes a set of web developer tools known simply as "Developer Tools". Unfortunately, these tools are quite basic and until IE9, did not include several features that can be found in the tools for competing browsers, such as network analysis and console logging. These missing features can be supplanted through the use of additional external tools, but you are probably just best to use an alternative browser as your standard development browser. If you want to get Developer Tools for IE6 or IE7, you can download it in the form of the "Developer Toolbar". Figure 1.8 shows a screenshot of IE8's Developer Tools.

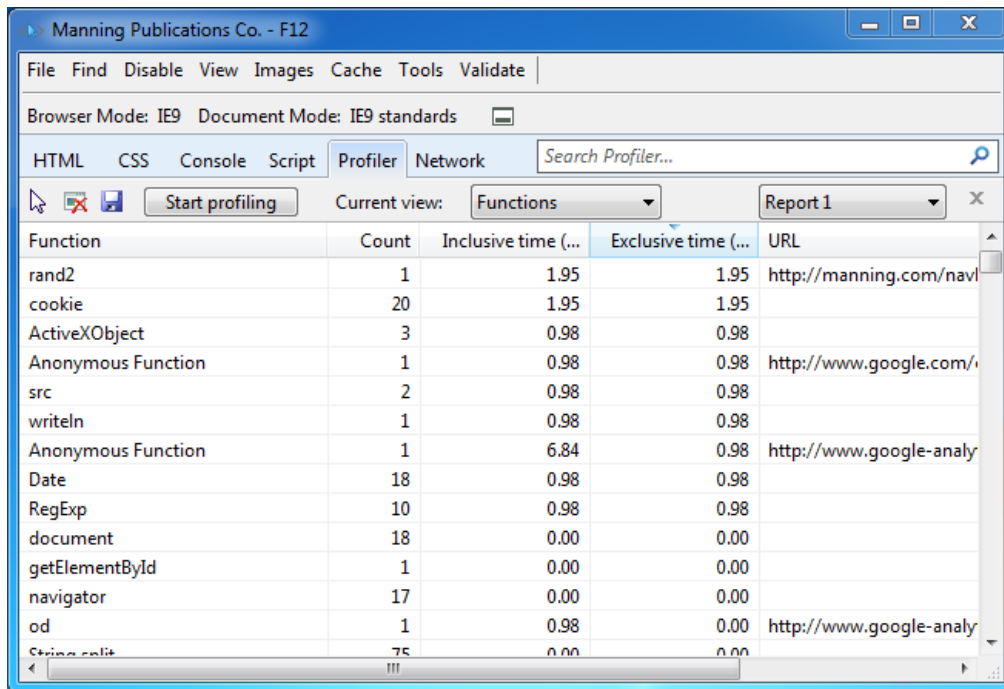


Figure 1.8. Microsoft Internet Explorer 8 Developer Tools showing how JavaScript profiling allows you to measure the time functions and blocks of code take to execute, allowing you to optimize your code for optimum performance.

Additional tools are available for Internet Explorer such as DebugBar, which adds some missing tools to older versions of IE, as well as some nice extras such as HTML validation, taking screenshots and a color picker. DebugBar can be downloaded from <http://www.debugbar.com/>.

### 1.3.5 jsFiddle

Sometimes you may want to try out some HTML, CSS and JavaScript code quickly and store it as a snippet that you can return to at some point in the future. To do this on your own machine you would need to open a text editor, create one or more text files (if you want to separate the HTML, CSS and JavaScript elements), save the files and open them in a browser. If you want to share the snippet, you would need to upload the files to a Web server, and the person you are sharing with must use their browser's "View Source" feature to see the code behind it. This is, quite frankly, a bit of a pain. Wouldn't it be great if there was an integrated solution that allows you to enter HTML, CSS and JavaScript code and view the results in a single window? How awesome would it be to be able to save that snippet so that when you share it the recipient sees the exact same view as you?

There is a nifty little web application named jsFiddle that provides all of this functionality. Not only that, but it also gives you a real simple way to include various JavaScript libraries, tidy up your markup, check the validity of your JavaScript code with JSLint, test Ajax requests and much more besides. A screenshot of jsFiddle in action is shown in figure 1.9.

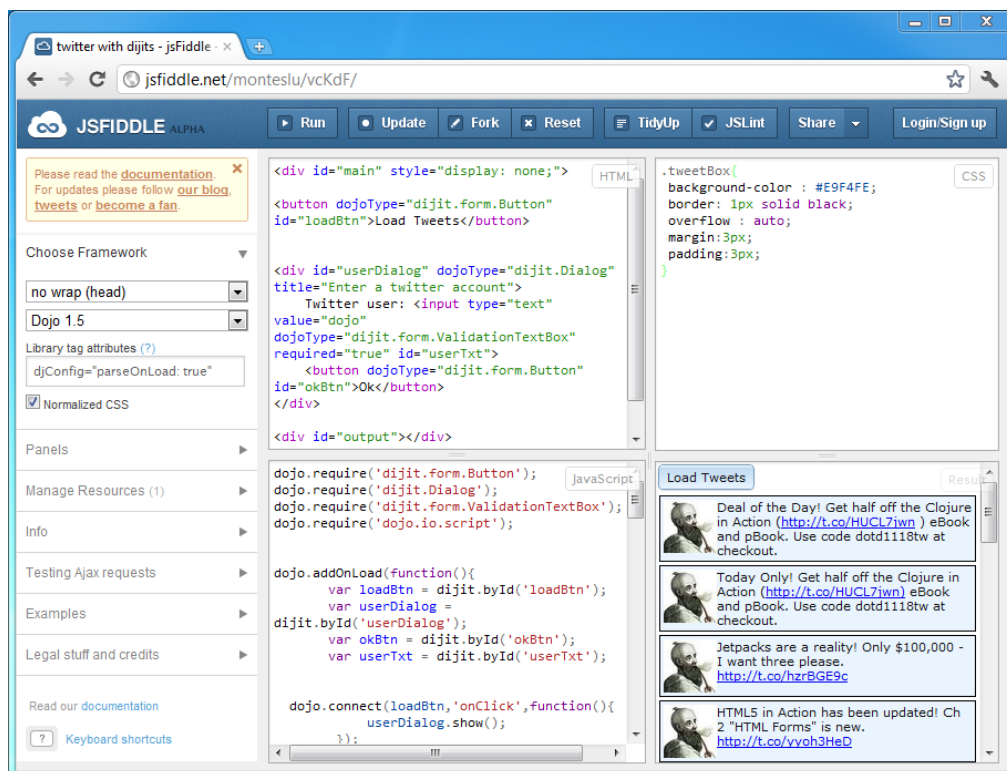


Figure 1.9. The jsFiddle web app allows you to quickly write HTML, CSS and JavaScript code and see a preview of the result, all in a single browser window.

To use jsFiddle, simply visit <http://jsfiddle.net> – you don't even need an account. Check out the examples to get an idea of some of the neat things you can do with this excellent tool.

Now that you're equipped with the tools you need to start writing HTML Web applications, we'll help you put those tools and your new knowledge to use with the new elements introduced in HTML5.

## 1.4 Using HTML5's new elements

When any new version of HTML is released, it typically defines a series of new elements that developers can use in their Web pages. HTML5 is no different, and defines quite a large

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

number of new elements, and deprecates or removes some others. In this section, you will discover the new semantic elements that are included in HTML5, and how to use them appropriately in your own markup. Next, you will learn about how using ARIA roles can enable assistive devices to better cope with modern, sophisticated Web applications. Following this, you will see how you can use microdata to further enhance the semantics of your HTML documents, allowing search engines to make more sense of different aspects of your Web pages. Finally, you will find out what elements and attributes have been removed from the HTML language altogether in HTML5. Let's get started!

### 1.4.1 *The new semantic elements*

If you have read about HTML5 before you picked up this book, chances are you've heard plenty about the new semantic elements. Although they are relatively straightforward, there is a tendency for people to get overexcited about this new set of tags. They are quite important, particularly if you want search engines and assistive devices such as screen readers to understand your pages better, but from a functional perspective the new elements are really simple.

If you're expecting these new elements to do something magical in terms of how they look on your page, you're in for quite some disappointment. Using these new elements on your page is functionally equivalent to using a series of `<div>` elements; they behave as block elements by default and can be styled as required using CSS. Their importance comes from the standard semantic meaning they have. If you think of a typical blog post, the Web page probably contains a series of sections. First, you have the site heading and navigation, maybe some side bar navigation, a main content area, and a footer area with further navigation links and perhaps some copyright and legal links. The following listing demonstrates how such a blog post might have been marked up in HTML 4 or XHTML.

#### Listing 1.2. HTML 4 markup for a blog post

```
<div class="header">
  <h1>My Site Name</h1>
  <h2>My Site Slogan</h2>
  <div class="nav">
    <ul><!-- Main Site Nav here --></ul>
  </div>
</div>

<div class="sidebar">
  <h3>Links Heading</h3>
  <ul><!-- Sidebar links --></ul>
</div>

<div class="main">
  <h4>Blog Post Title</h4>
  <div class="meta">
    Published by Joe on 01 May 2011 @ 12:30pm
  </div>
  <div class="post">
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

```

        <!-- Actual blog post -->
    </div>
</div>

<div class="footer">
    <ul><!-- Footer links --></ul>
    <!-- Copyright info -->
</div>

```

First off, there is nothing wrong with this code. In fact, it is perfectly valid to use it in HTML5 and you can absolutely continue to use `<div>` elements with semantic class names if you wish. However, from a semantic point of view, there are a number of problems with the approach. First off, we are separating the various areas of the blog post using named classes. This is fine, but it is completely up to the author how the classes are named. My “header” might be your “heading”; I call the main section “main”; you might call it “body” or “article”. In addition, some people may prefer to use IDs instead of classes. In short, there is no way for a search engine or other computer-controlled application to consistently determine what each section actually represents.

This is where the new semantic elements come into play. Rather than using classes and IDs for sections like headings, navigation, footers, and so on, you now use a standard HTML element. Let’s look at these new elements now.

- **<header>** This element contains the title of a page or section. You can use this multiple times on the same page—that is, once for the page heading (logo, slogan) and again for the post heading (blog post title, permalink, meta information).
- **<hgroup>** You can use the `<hgroup>` element to group numbered heading (`<h1>` to `<h6>`) elements where you have more than one. In HTML5, the numbered heading elements are relevant to the section – you can have an `<h1>` in your site’s main heading, and then a separate `<h1>` in your actual blog post. The heading structure doesn’t apply to the entire page like it did in HTML 4.
- **<footer>** This usually appears at the bottom of a page or section, typically used for things like related posts or links, copyright information, meta data and more.
- **<nav>** This element represents a section of links that allow the user to navigate to other pages or parts of the same page. The specification dictates that only “major navigation blocks” should use the `<nav>` element – main site navigation, table of contents or side navigation bars are all usually considered “major navigation blocks”.
- **<section>** This is used to define a section of a document or application – in a wiki article this might be a major section of the article, in an application it might be a popup form that allows you to add data. Sections can have their own headers, navigation and footers if required.
- **<article>** An article is deemed to be a self-contained publishable composition that can be redistributed on its own, for example as an entry in an RSS feed. Articles are typically items like blog posts, comments, forum posts, news entries, and so on. Like

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

sections, articles can themselves contain headers, navigation and footer elements. A good example of a potential usage for <article> in a Web application is a “tweet” in a list of Twitter updates.

- **<aside>** The <aside> element is used to define a section of a page that is separate to the content area it is defined in. In a book or magazine, this might be represented as a sidebar – it contains information on the same topic but doesn’t quite fit into the main article itself. For example, if you had a blog, you may have adverts displaying alongside posts – these could be placed in an <aside> element. In a Web application, you might use <aside> for a popup or floating window that appears over the main part of the application itself.
- **<time>** Dates and times can be represented in a wide range of formats, languages and abbreviated styles. This makes it difficult to parse dates consistently. The <time> element allows you to continue to use the date/time format of your choice, meanwhile wrapping it in a standardized format that is easier for a computer to understand. The <time> element also offers a pubdate attribute which can be used to denote the publish date of an article such as a blog post.
- **<mark>** You can use the <mark> element to represent a part of text in your document that should be marked or highlighted. A common use for this would be to highlight search terms within a document.

The following listing illustrates how we might re-write the earlier HTML 4 example using HTML5 and some of the new elements.

### Listing 1.3. HTML5 markup for a blog post

```
<header>
  <hgroup>
    <h1>My Site Name</h1>
    <h2>My Site Slogan</h2>
  </hgroup>
  <nav>
    <ul><!-- Main Site Nav Here --></ul>
  </nav>
</header>

<nav>
  <h1>Links Heading</h1>
  <ul><!-- Sidebar links --></ul>
</nav>

<section>
  <article>
    <header>
      <h1>Blog Post Title</h1>
      <div class="meta">
        Published by Joe on
        <time datetime="2011-05-01T12:30+00:00" pubdate>
```

```

                01 May 2011 @ 12:30pm
            </time>
        </div>
    </header>
    <section>
        <!-- Actual blog post -->
    </section>
</article>
</section>

<footer>
    <ul><!-- Footer Links --></ul>
    <!-- Copyright info -->
</footer>

```

Next, you'll learn how you can improve the semantics of your page even further using ARIA roles, which greatly enhance the accessibility of your Web pages.

### 1.4.2 Enhanced accessibility with ARIA roles

When building Web applications, you must take care to ensure that your application is accessible to all users, including those who require assistive devices such as screen readers. Ensuring that your documents are accessible requires careful consideration when it comes to the semantic meaning of your markup. Using simple HTML markup this is relatively straightforward, and HTML5's new elements improve the semantics even further. However, when you move towards the area of Web applications, it becomes much more difficult to cater to assistive technologies. Because of the large amount of JavaScript code used to dynamically modify Web pages in modern Web apps, it may seem almost impossible to keep your application working in an accessible way. This is where the Web Accessibility Initiative (WAI) and Accessible Rich Internet Applications (ARIA) roles come into play.

The WAI-ARIA specification intends to improve Web applications by expanding on the accessibility information provided by the author of an HTML document. It requires that applications provide full keyboard support that can be implemented regardless of the device, making it easier to use on a smart phone, e-Book reader or Internet-ready television. Most importantly, perhaps, the specification focuses on improving the accessibility of dynamic content generated by scripts and providing interoperability for assistive technologies.

ARIA roles, relationships, states and properties allow you to define exactly how your Web application works in a way that an assistive device such as a screen reader can understand. The HTML5 specification explicitly states that you may use the ARIA role and `aria-*` attributes on HTML elements as described in the relevant ARIA specification. HTML5 also defines a set of default ARIA roles that apply to certain HTML elements – for example it is implied that a checkbox `<input>` element has an ARIA role of “checkbox”, and you should not explicitly use the role or `aria-*` attributes in these cases.

There are also various HTML elements where the native semantics can be modified so that they behave differently. For example, you might have an `<a>` element which behaves like a button, perhaps submitting a form after performing some validation routine. The

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

HTML5 specification defines a list of valid semantics for these elements, for example, when you use the <a> element to create a hyperlink, it assumes the “link” role by default, and if this is modified its role can only be changed to one of the following: “link”, “button”, “checkbox”, “menuitem”, “menuitemcheckbox”, “menuitemradio”, “tab” or “treeitem”.

For a full list of both the default implied ARIA semantics and the restrictions on how you can modify the semantics of certain elements, see the WAI-ARIA section of the HTML5 specification at <http://dev.w3.org/html5/spec/Overview.html#wai-aria>.

In the next section, we’ll look at another concept that is closely tied into HTML5, albeit not strictly defined in the specification itself: microdata.

### 1.4.3 More semantics with Microdata

When most people discuss HTML5, they are usually referring not just to the HTML5 specification itself, but also to a number of related specifications. The HTML microdata specification allows additional semantic information to be added to a Web page, allowing the likes of search engines and Web browsers to provide additional functionality to the user based on this data.

To use microdata, you need a vocabulary, which defines the semantics that are to be used. You can define your own vocabularies, but more likely you’ll want to use generic versions, such as those provided by Google at <http://www.data-vocabulary.org/>, including Event, Organization, Person, Product, Review, Review-aggregate, Breadcrumb, Offer and Offer-aggregate. By using a generic vocabulary where possible, your microdata will be easier to digest for search engines and other applications, as they will also use this vocabulary. In a moment, we’ll look at an example of defining an event using microdata. First, we need to look at the various HTML properties that allow us to use microdata on our pages.

- **itemscope** The itemscope attribute tells the parser that this element and everything contained inside it describes the entity being referenced. The value of this attribute is Boolean, and is usually omitted. When an element has the itemscope attribute, it also defines the itemtype attribute.
- **itemtype** This attribute defines the URL at which the vocabulary for the item being specified is found. For example, if you are using the Person microdata vocabulary provided by Google, the value of this attribute would be “http://data-vocabulary.org/Person”. This attribute is defined on the same element as the itemscope attribute.
- **itemid** This attribute is a global unique identifier for the item. It is optional, but if used, it must be a valid URL.
- **itemprop** This attribute indicates that the content of the element contains the value of the specified property. For example, the following code `<span itemprop="name">Joe Lennon</span>` denotes that the name property of this item is “Joe Lennon”. The itemprop attribute is the one that you will likely use the most when working with microdata.

- **itemref** If you want to specify microdata attributes on an element that is not contained inside an element that defines the itemscope attribute, you can use the itemref property to refer to that item instead. The value of this attribute must be an unordered set of unique space-separated tokens that contain the IDs of elements on the same HTML page. The itemref property must be put on the same element you place itemscope on.

Listing 1.4 illustrates microdata in action using an event item that adheres to Google's Event microdata vocabulary at <http://data-vocabulary.org/Event>. This code creates a snippet of HTML code for an event, with microdata properties defined that will allow a search engine to provide better interpretation of the event, perhaps by showing the event date in a calendar, or location on a map.

#### Listing 1.4. Microdata in action

```

<div itemscope itemtype="http://data-vocabulary.org/Event">
  <a href="http://example.com/event/1" itemprop="url">
    <span itemprop="summary">John's 40th Birthday Party</span>
  </a>
  <span itemprop="description">To celebrate John turning 40,
we're throwing a BBQ party in his honour this Friday evening
at close of business. Please come and bring your friends and
family!</span>

  Location:
  <span itemprop="location" itemscope itemtype="http://data-
vocabulary.org/Address">
    <span itemprop="street-address">500 Market Way</span>
    <span itemprop="locality">Ballincollig</span>
    <span itemprop="region">Cork</span>
  </span>

  Date and Time:
  <time itemprop="startDate" datetime="2011-05-06T18:00+00:00">
    Fri, May 6th @ 6pm
  </time>
</div>

```

Microdata is HTML5's answer to microformats, a set of open specifications for describing data of a particular format, such as calendar and event data (hCalendar), and people, companies, organizations and places (hCard). The primary difference between microdata and microformats is that the former is part of the HTML5 specification, whereas the latter comprises a separate set of standard specifications (see <http://microformats.org> for further information). One of the issues with using microformats is that they require developers to add semantics using the class attribute, which is also used for other purposes such as CSS styling and DOM manipulation. Because microdata in HTML5 uses new attributes specifically designed for specifying format, it allows developers to continue using the class attribute for its main purpose.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

There is a third way of annotating HTML syntax – the RDFa specification. RDFa has its roots in the XHTML standard, requiring the use of XML namespaces and schemas. Generally speaking RDFa is considered to be more complex than both microformats and microdata, due to its relationship with XML.

All three approaches (microdata, microformats and RDFa) can be used to improve your site's search listings in Google. For more information on using these approaches, see <http://www.google.com/support/webmasters/bin/topic.py?topic=21997>. Now, let's wrap up this chapter by looking at the parts of HTML that have been removed in HTML5.

#### 1.4.4 Out with the old...

A number of elements and attributes are absent from HTML5. They will typically continue to work in major browsers due to backward compatibility, but the specification requires that developers do not use them in their Web pages going forward. The list of removed elements appear in table 1.3.

**Table 1.3. Elements no longer supported in HTML5**

<acronym>	<applet>	<basefont>	<big>	<center>
<dir>	<font>	<frame>	<frameset>	<isindex>
<noframes>	<strike>	<tt>	<u>	

In addition, a series of attributes are no longer allowed. Some are purely presentational attributes, and we recommend that you use CSS instead of these attributes, which will no longer validate in HTML5. Others are merely deemed to no longer be required. Note that some of the attributes listed in table 1.4 may still be perfectly valid on other elements (for example, the type attribute on the <input> element). Next to each attribute, we list in parentheses the elements to which it no longer applies.

**Table 1.4. Attributes no longer supported in HTML5**

abbr (<td>, <th>)	align (all presentational elements)
alink (<body>)	archive (<object>)
axis (<td>, <th>)	background (<body>)
bgcolor (<body>, <table>, <tr>, <td>, <th>)	border (<table>, <object>)
cellpadding (<table>)	cellspacing (<table>)
char (<col>, <colgroup>, <tbody>, <td>, <tfoot>, <th>, <thead>, <tr>)	charoff (<col>, <colgroup>, <tbody>, <td>, <tfoot>, <th>, <thead>, <tr>)
charset (<a>, <link>)	classid (<object>)

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=792>

clear ( )	codebase (<object>)
codetype (<object>)	compact (<dl>, <menu>, <ol>, <ul>)
coords (<a>)	declare (<object>)
frame (<table>)	frameborder (<iframe>)
height (<td>, <th>)	hspace (<img>, <object>)
link (<body>)	longdesc (<img>, <iframe>)
marginheight (<iframe>)	marginwidth (<iframe>)
name (<img>)	nohref (<area>)
noshade (<hr>)	nowrap (<td>, <th>)
profile (<head>)	rev (<a>, <link>)
rules (<table>)	scheme (<meta>)
scope (<td>)	scrolling (<iframe>)
shape (<a>)	size (<hr>)
standby (<object>)	target (<link>)
text (<body>)	type (<li>, <ol>, <param>, <ul>)
valign (<col>, <colgroup>, <tbody>, <td>, <tfoot>, <th>, <thead>, <tr>)	valuetype (<param>)
version (<html>)	vlink (<body>)
vspace (<img>, <object>)	width (<col>, <colgroup>, <hr>, <pre>, <table>, <td>, <th>)

Alongside missing elements, there are also some elements whose behavior has changed in some shape or form between HTML 4 and HTML5. For detailed information on these changes, visit <http://www.w3.org/TR/html5-diff/#changed-elements>.

## 1.5 Summary

HTML5 is the most important revision of the HTML markup language since its inception in 1991. Although HTML started off as a relatively straightforward markup language, it has since become a platform for complex Web page design and Web application development, particularly when coupled with its close relations CSS and JavaScript. HTML5 is the first version of the language to acknowledge this and include a series of standard JavaScript APIs in the specification itself.

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=792>

You should now be able to create HTML5 ready pages of your own, with the new doctype, improved semantics and techniques to ensure backward-compatibility with older Web browsers and in particular, versions 6, 7 and 8 of Internet Explorer. At this point, you are probably as sick of hearing about old versions of browsers as we are, and you want to get right into developing some cool applications. Fear not! Starting with the next chapter, you will be developing a sample app in each and every chapter. By the time you've finished the book, you'll be able to say you've built some awesome games, mobile apps and much more besides. And no, that's not a typo, we did say games, plural!

But first, in the next chapter, you will go beyond introductory concepts and gain a deep insight into the vast improvements in Web Forms that HTML5 has to offer, including the new input types that allow entry of a much wider variety of data types, new attributes such as autofocus and placeholder, and the out-of-the-box features that reduce the reliance on JavaScript for client-side data validation.