

# *appendix A*

## *HTML5 and related specifications*

---

### ***This appendix covers***

- Development of the HTML5 specification
- Popular W3C-accepted HTML5 specifications (non-drafts)
- Related, popular specifications (non-HTML5)

It would be odd if you hadn't heard buzzwords such as HTML5, CSS3, and Node.js used inaccurately, or even incorrectly, at some point. In particular, HTML5 has become a catchall word for emerging web technologies. For example, one of the authors once met a marketer who said, "I can create an SEO-optimized video game with HTML5." At the least, it's important to know what an HTML5 specification is, and what it isn't, to keep you from making a fool of yourself. For appendix A, we'll cover what's officially HTML5 and what isn't.

### **A.1 *The origins of HTML5***

You might be surprised to learn that the Worldwide Web Consortium (W3C) didn't advocate HTML5 in the beginning. W3C considered HTML to be dead after HTML4 and was working on XHTML2, continuing the trend of web markup based on an XML syntax. If you thought XHTML1 was strict, the second version promised to take

things further. As a result, many members in the W3C felt a need for a change of direction, and the WHATWG (Web Hypertext Application Technology Working Group) was formed to begin work on HTML5.

HTML5 started off as Web Apps 1.0 and Web Forms 2.0, then later merged into a single specification: HTML5. Before long, W3C began to realize that there was merit to the case for HTML and began working on version 5 of HTML (not quite the same as HTML5, it should be noted), taking the work of WHATWG as the starting point for the new standard. For a time, this only added to the confusion. Not only was WHATWG continuing to work on HTML5, but W3C was also working on version 5 of HTML, derived from an earlier version of the HTML5 specification, while it was also continuing work on XHTML2. Confused? We certainly were.

Since that time, XHTML2 finally died, and developers at both WHATWG and W3C worked on the HTML5 specification, with each maintaining a separate version, albeit both overseen by the same editor. Why the need for two separate groups? Politics. For various reasons, some stakeholders in the process can't join WHATWG and others can't join W3C. As a result, both groups continue to work concurrently.

### **A.1.1 WHATWG vs. W3C**

The goal of WHATWG is to continually update the “HTML Living Standard” based on feedback from all stakeholders to maintain a position slightly ahead of current implementations. WHATWG has given up on version numbers and sees the standard as an evolving document. It aims to stay just ahead of the functionality in browsers, providing a forum for everyone to agree on the details of any new feature and documentation of the final implementations.

W3C is sticking with the traditional version-based approach. We can expect HTML5 to be followed by HTML6 and HTML7, all using a snapshot of the WHATWG document as a basis. As a result, W3C has split what exists as one specification at the WHATWG into (currently) eight different specifications so that features can develop at their own pace without holding up the release of standards. You can find a list of the individual specifications at WHATWG's FAQ page: <http://mng.bz/dWRb>.

Another key difference between the groups is decision making. In WHATWG, the editor has complete control when it comes to making decisions regarding the HTML5 specification. W3C has an HTML Working Group with its own escalation process for making decisions on disputed issues.

W3C has a large number of specifications outside of HTML, and one goal is that all the specs should be compatible. W3C has been focused on XML-based technologies for a number of years, and WHATWG was formed in opposition to the pure XML approach, so this has been the underlying source of the disagreements so far. But despite some heated discussions, the two specs are yet to diverge.

To help you keep the key differences straight, refer to the summary in table A.1.

**Table A.1 WHATWG and W3C compared**

Topic	W3C	WHATWG
<b>Membership</b>	Mostly paid members with corporate sponsors.	Anyone can join the mailing list.
<b>Editorial process</b>	Editor is subject to strictures of W3C's feedback and review processes.	Editor is "benevolent dictator."
<b>HTML-related specifications managed</b>	8 (derived from WHATWG's 1 spec).	1.
<b>Non-HTML specifications managed</b>	Lots (e.g., CSS, DOM, SVG, XML, RDF).	None.
<b>Release process</b>	Versioned snapshots.	Rolling release, constantly updated.

The real-life interactions of thousands of smart people are, of course, more complex than can be described in a simple table, especially when you remember that many of these people are in both W3C and WHATWG. But this section has given you some useful context if you ever have to dive into a debate on the WHATWG mailing list or the W3C bug tracker over some detail of one of the specs when you're just trying to figure out which browser is "doing the right thing."

### **A.1.2 So ... what is HTML5 anyway?**

We consider a technology an official part of HTML5 if it's part of the WHATWG Living Standard or it's one of W3C's specifications derived from that standard. But many of the technologies, such as CSS3, Geolocation, and the Storage APIs, that partake of the buzzword *HTML5* aren't part of this official definition. In the next section, you'll have a quick review of the HTML5 technologies that are officially HTML5, and in the following section, those that are not.

#### **Does it really matter what is or isn't HTML5?**

The short answer is no! When you're building web apps, you need to pick and choose technologies in the modern web platform based not on which spec they appear in but on whether they do something you need and they work in browsers. Although you may end up in some heated social network debates, you'll receive no explicit punishment for claiming things like Geolocation as a key part of your HTML5 app. As you'll see, even the authors of this book have stretched the definition of HTML5 to include several "unofficial" technologies.

## **A.2 Popular HTML5 specifications**

In this section, we'll discuss the technologies that are part of WHATWG's HTML Living Standard and the HTML5 family of specifications at W3C. Although the WHATWG spec hasn't always been called the HTML Living Standard, we'll use that term to differentiate it from the HTML5 spec at W3C. Each section will mention which specification at the W3C applies and the relevant chapter or chapters in this book.

### **A.2.1 Semantic markup, forms**

HTML5 introduces HTML elements that change how people structure website markup and use form elements. It also gives programmers more control over their markup through attributes such as `data`. These attributes can hold important metadata inside an HTML element. This is all core HTML stuff and so is in the W3C HTML5 specification.

You can learn about semantic markup and forms in chapters 1 and 2.

### **A.2.2 Video and sound (multimedia)**

In the past, web developers have primarily relied on Flash or another plug-in to provide audio and video support. The HTML5 audio and video elements allow a browser to run both, without any additional configuration. Both use the Media Element API, which means their event systems for toggling playback, sound, stopping, and so on are similar. This is also in the core W3C HTML5 specification.

Audio and video are covered in chapter 8; also check out appendix I for some of the more cutting-edge video technologies.

### **A.2.3 Canvas and SVG (interactive media)**

The Canvas API and SVG give you the ability to create interactive media via JavaScript programming. The first and most popular Canvas API was originally an Apple product from Mac OS X. Developers can create raster-based graphics on the fly inside a `<canvas>` element with it. Although the `<canvas>` element itself is covered in the core HTML5 spec, the 2D context (the JavaScript API that lets you draw stuff) is in a separate specification called "HTML Canvas 2D Context." Note that although WebGL allows Canvas to display 3D graphics, the 3D context is not officially part of HTML5 (see section A.3 for details).

SVG is an XML-based language that's been around since 2001. All HTML5 adds is the ability to inject SVG elements into HTML pages (it has always been allowed to inject SVG into XHTML pages), nothing more. It's important to understand that SVG is a piece of HTML5 but not a specification created by it.

Canvas, the 2D context, and SVG are covered in chapters 6 and 7; Canvas is also used in chapter 8 to manipulate live video and in chapter 9 along with the 3D context.

### **A.2.4 Storage**

HTML5 is associated with several storage-based APIs; the ones that are part of the HTML5 specifications are Web Storage and Offline Applications.

At W3C, offline apps are covered in the core HTML5 spec, and session and local storage are covered by the Web Storage spec. Both are discussed in chapter 5.

### **A.2.5 Messaging**

Web Messaging (cross-document and channel messaging), Server-Sent Events, and WebSockets are all core HTML5 technologies. At W3C they are covered by three specs: “HTML5 Web Messaging,” “Server-Sent Events,” and “WebSockets API.” Note that the WebSockets Protocol, which describes the format of the transmitted data, is defined by a specification at the Internet Engineering Task Force (IETF). Messaging is covered in chapter 4 and appendix F.

### **A.2.6 The XML HTTP Request object**

This API has existed in IE since the late 1990s and has been heavily used in web applications since Firefox implemented its version between 2000 and 2002, giving birth to AJAX (Asynchronous JavaScript And XML). But XHR had never been documented in any specification until WHATWG added it to its specifications in 2004. Currently, the XML HTTP Request (XHR) object has a specification all to itself at the W3C. XHR and AJAX are well known and well used, so even though XHR is, strictly speaking, HTML5, we don’t cover it specifically in this book.

## **A.3 Popular non-HTML5 technologies**

Some popular specifications and technologies are commonly mistaken for HTML5 because of their intriguing features. Although these new technologies began to emerge around the same time that HTML5 was becoming established and frequently featured in HTML5 Showcase sites and HTML5 books (including this one), they’re not HTML5 by the definition given earlier. One good way to describe this group of web development technologies, suggested by Bruce Lawson, is “HTML5 and friends.”

### **A.3.1 CSS3**

CSS3 brings several amazing features to web development, such as transitions and 3D transforms. But it’s an entirely separate specification from HTML5. There is no specific CSS3 coverage in this book, but CSS will be used to support all of this book’s apps.

For a gentle introduction to CSS3, see *Hello! HTML5 & CSS3* by Rob Crowther (Manning, 2012). There’s also good information on tools for CSS3 in *Sass and Compass in Action* by Wynn Netherland, Nathan Weizenbaum, Chris Eppstein, and Brandon Mathis (Manning, 2013).

### A.3.2 Geolocation

A lot of early HTML5 demos featured the Geolocation API. But this API has never been a part of the HTML Living Standard or the HTML5 family of specifications at W3C.

The Geolocation API has its own specification at the W3C; it's covered briefly in chapter 3.

### A.3.3 Storage

We mentioned storage in the previous sections. There are two key storage technologies that aren't part of the HTML5 spec: IndexedDB and the File System API. These are in the Indexed Database API, File API, File API: Directories and System, and File API: Writer specs at the W3C.

Check out chapter 5 for more on IndexedDB and chapter 3 for the File API.

### A.3.4 WebGL

The WebGL technology is based on OpenGL. The Khronos Group has taken OpenGL and adapted it for use in web browsers; the result is WebGL. All desktop browsers have support for WebGL. Even Microsoft, after initially being opposed to the technology, has implemented WebGL in IE11.

### A.3.5 Node.js

Many people have mistaken the new software platform Node.js (often simply called *Node*) for an HTML5 API. Although it makes use of emerging web-standard technologies and improves the use of many HTML5 APIs, it's not part of any web standard. It runs on Google's V8 JavaScript engine and is primarily sponsored by Joyent. This book covers basic Node usage; for more, check out *Node.js in Action* by Mike Cantelon, TJ Holowaychuk, and Nathan Rajlich (Manning, 2013). Node.js is also covered in chapter 4 and appendix E.

### A.3.6 jQuery and other JavaScript libraries

JavaScript libraries followed along after the last "buzzword fad" on the web: AJAX. The main problem they initially solved was to provide a compatibility layer over the differing browser implementations of the XMLHttpRequest object that underlies AJAX, but each also added its own features. The popular Prototype.js added features and encouraged a style of programming inspired by the Ruby programming language; Dojo did a similar thing except in the style of Python. For many years, the ultimate solution in cross-browser compatibility has been the jQuery library. HTML5 doesn't replace libraries like jQuery, but it should help make them more performant. The extensive effort to standardize browser behavior through the process of building the HTML5 spec will also make the compatibility provided by these libraries less important. Some common JS library features that are replaced by HTML5 are shown in table A.2.

**Table A.2 JS Library functionality and modern web platform equivalents**

Feature	JS libraries	HTML5 (or related) feature
Selecting elements by class	Nearly all	The <code>getElementsByClassName()</code> method was originally introduced in the HTML Living Standard; it's currently in the DOM CORE spec at W3C. The <code>querySelector()</code> and <code>querySelectorAll()</code> methods are defined in the Selectors API Level 1 spec at the W3C.
Drag and drop	Scriptaculous, jQuery-UI, ExtJS, Dojo, YUI	Added to the HTML Living Standard as a reverse engineering of the IE feature.
Advanced form controls (date pickers, sliders, spinboxes, etc.)	jQuery-UI, ExtJS, Dojo, YUI	New form controls are part of the core HTML5 specification.
Storing arbitrary data on elements	Jquery, Dojo	HTML5 has <code>data-*</code> attributes for storing data for scripting.

## A.4 Keeping up with the specs

The best way to keep up with the main HTML specification is to follow The WHATWG Blog (<http://blog.whatwg.org/>). Reading the specification in its raw form can be tedious, to say the least. We find it much easier to read the spec using the edition for web authors, which is available at <http://developers.whatwg.org/>. This edition doesn't include the technical information targeted at browser vendors and is far easier to read.

For the rest of the specifications there's no central source. Each individual W3C working group has its own blog and/or mailing list. One approach is to keep an eye on the development blogs for the major browsers to find out what new features they're experimenting with:

- *Mozilla Hacks*: <https://hacks.mozilla.org/>
- *Google Chrome Blog*: <http://chrome.blogspot.co.uk/>
- *IEBlog*: <http://blogs.msdn.com/b/ie/>
- *Surfin' Safari*: <https://www.webkit.org/blog/>
- *Opera Desktop Team*: <http://my.opera.com/desktopteam/blog/>
- *Opera Mobile*: <http://my.opera.com/mobile/blog/>

# *appendix B*

## *HTML5 API reference*

---

In this appendix, you'll find numerous references that give you a quick overview of various HTML5 and related APIs. We've compiled lists of methods, attributes, and events that should make it easy for you to look up how to use API information when you need it.

The material is broken down into three sections:

- The HTML5 APIs
- Other APIs and specifications, which cover Geolocation and IndexedDB
- The File System API

We begin with the HTML5 APIs.

### **B.1 HTML5 APIs**

In this section, we cover what you need to know for the

- Constraint Validation API
- API for offline web applications
- Editing API
- Drag and Drop API
- Microdata API
- APIs for Web Storage
- Media Element API

#### **B.1.1 Constraint Validation API**

The Constraint Validation API defines a series of new attributes and methods, outlined in table B.1, that you can use to detect and modify the validity of a given form element.



**Table B.1 Constraint Validation API**

Attribute/method	Description
<code>willValidate</code>	Checks if the element validates when the form is submitted.
<code>validationMessage</code>	Holds the error message the user will see if the element is checked for validity.
<code>validity</code>	<p>An object that contains attributes representing the validity states of the element. Each attribute defines a validation error condition. When “getting” an attribute, a value of <code>true</code> is returned if the error condition is true, otherwise <code>false</code>.</p> <p><code>validity</code> contains the following boolean attributes:</p> <ul style="list-style-type: none"> <li>▪ <code>valueMissing</code> (required field but has no value)</li> <li>▪ <code>typeMismatch</code> (incorrect data type)</li> <li>▪ <code>patternMismatch</code> (doesn't match required pattern)</li> <li>▪ <code>tooLong</code> (longer than <code>maxLength</code> content attribute value)</li> <li>▪ <code>rangeUnderflow</code> (lower than <code>min</code> content attribute value)</li> <li>▪ <code>rangeOverflow</code> (higher than <code>max</code> content attribute value)</li> <li>▪ <code>stepMismatch</code> (not a multiple of <code>step</code> content attribute)</li> <li>▪ <code>customError</code> (has a custom error)</li> <li>▪ <code>valid</code> (field is valid)</li> </ul>
<code>checkValidity()</code>	Checks if the element is valid.
<code>setCustomValidity(message)</code>	Sets a custom error message on the element.

### B.1.2 API for offline web applications

The API for offline web applications consists of a collection of events and a number of DOM attributes and methods. Table B.2 lists the events.

**Table B.2 Application cache events**

Event name	Description
<code>checking</code>	Fires when checking for an update or trying to download the cache manifest for the first time.
<code>noupdate</code>	Fires when manifest has not been modified.
<code>downloading</code>	Fires when the browser is downloading items in the manifest for the first time. Also fires when the browser is downloading items after detecting a manifest update.
<code>progress</code>	Fires once per file as the browser downloads each file listed in the manifest. The event object's <code>total</code> attribute returns the total number of files to be downloaded. The event object's <code>loaded</code> attribute returns the number of files processed so far.
<code>cached</code>	The application is cached and the download is complete.
<code>updateready</code>	Resources have been downloaded and an update is available. The application can use the <code>swapCache</code> method to switch to the new resources.

**Table B.2 Application cache events (continued)**

Event name	Description
<code>obsolete</code>	The manifest was not found and the cache is being removed.
<code>error</code>	The manifest or one of the resources in it was not found, or the manifest changed while the update was in progress, or some other error has occurred, so caching has been canceled.

Table B.3 lists the DOM attributes and methods for offline applications. All apply to the application cache object itself, apart from the ones where an explicit root object is listed.

**Table B.3 Application cache API**

Attribute/method	Description
<code>window.applicationCache</code>	Returns an application cache object for the active document.
<code>self.applicationCache</code>	Returns an application cache object for a shared worker.
<code>status</code>	Gets the current status of the cache: <ul style="list-style-type: none"> <li>■ <code>UNCACHED</code> (numeric value: 0)</li> <li>■ <code>IDLE</code> (1)</li> <li>■ <code>CHECKING</code> (2)</li> <li>■ <code>DOWNLOADING</code> (3)</li> <li>■ <code>UPDATEREADY</code> (4)</li> <li>■ <code>OBSOLETE</code> (5)</li> </ul>
<code>update()</code>	Starts downloading resources into a new application cache.
<code>abort()</code>	Cancels downloading of resources.
<code>swapCache()</code>	Switches to the newest application cache, if a newer one is available.

The Browser State API is covered in table B.4, though this is less useful than you might think. Deciding whether the browser is online isn't the same thing as being able to connect to the internet or your application. It's merely a reflection of the browser's online mode.

**Table B.4 Browser State attributes and events**

Attribute/method, event name	Description
<code>window.navigator.onLine</code>	Checks if the browser mode is online (returns <code>true</code> ) or offline (returns <code>false</code> ).
<code>online</code>	The browser's online status has changed to online.
<code>offline</code>	The browser's online status has changed to offline.

### B.1.3 Editing API

The Editing API allows you to implement direct editing of HTML pages loaded in the browser. This is commonly referred to as rich-text editing; it enables the web application to use all the formatting options available to HTML. This ability distinguishes rich-text editing from plain-text editing that can be achieved in `textarea` elements and other form inputs.

The Editing API was created by reverse engineering the behavior of IE. The documentation had always been incomplete, so there are many parts of it that exist simply because IE has them rather than because there's a rational explanation for their existence.

All the methods in table B.5 are on the document object; in most cases they will apply to any selected block of text within a `contenteditable` section of the current document.

**Table B.5** Editing API

Method	Description
<code>execCommand(command, showUI, value)</code>	Executes the command described in the first argument. The <code>command</code> argument is a string value. The <code>showUI</code> argument is a Boolean value to determine whether or not to show the default UI associated with <code>command</code> . The <code>value</code> argument is passed to <code>command</code> . Not all commands need a <code>value</code> argument.
<code>queryCommandEnabled(command)</code>	Checks if <code>command</code> is supported and enabled.
<code>queryCommandIndeterm(command)</code>	Checks if <code>command</code> is indeterminate (if the selected text is part active and part inactive).
<code>queryCommandState(command)</code>	Returns a Boolean value indicating whether <code>command</code> is currently applied to the selected text.
<code>queryCommandSupported(command)</code>	Checks if <code>command</code> is supported.
<code>queryCommandValue(command)</code>	Returns <code>command</code> 's value, if it has one.

As you can see, the API isn't much use without a value to enter for `command`. Tables B.6–B.8 list categories of available commands. Pass the `command` as a string to the methods in table B.6, for example: `execCommand('bold', false, '')`. For more information on these commands, see <http://mng.bz/4216>.

Table B.6 lists commands for formatting inline elements.

**Table B.6** Inline formatting commands

<code>backColor</code>	<code>bold</code>	<code>createLink</code>
<code>fontName</code>	<code>fontSize</code>	<code>foreColor</code>
<code>hiliteColor</code>	<code>italic</code>	<code>removeFormat</code>
<code>strikethroug</code>	<code>subscript</code>	<code>superscript</code>
<code>underline</code>	<code>unlink</code>	

Table B.7 lists commands for formatting block elements.

**Table B.7 Block formatting commands**

<code>delete</code>	<code>formatBlock</code>	<code>forwardDelete</code>
<code>indent</code>	<code>insertHorizontalRule</code>	<code>insertHTML</code>
<code>insertImage</code>	<code>insertLineBreak</code>	<code>insertOrderedList</code>
<code>insertParagraph</code>	<code>insertText</code>	<code>insertUnorderedList</code>
<code>justifyCenter</code>	<code>justifyFull</code>	<code>justifyLeft</code>
<code>justifyRight</code>	<code>outdent</code>	

Table B.8 lists commands for other formatting and editing issues.

**Table B.8 Miscellaneous commands**

<code>copy</code>	<code>cut</code>	<code>defaultParagraphSeparator</code>
<code>paste</code>	<code>redo</code>	<code>selectAll</code>
<code>styleWithCSS</code>	<code>undo</code>	<code>useCSS</code>

#### B.1.4 Drag and Drop API

The Drag and Drop API is another API that's reverse engineered from the IE implementation. The API has three main parts: the `dataTransfer` object, the `dataTransfer` item, and a collection of events. These are covered in tables B.9, B.10, and B.11, respectively. A drag operation will create a `dataTransfer` object; this will contain one or more `dataTransfer` items in the `items` attribute, and you can gain access to both by listening to the events.

**Table B.9 dataTransfer object**

Attribute/method	Description
<code>dropEffect</code>	This is the type of operation taking place (copy, link, move, none).
<code>effectAllowed</code>	Contains the type of operations allowed (copy, copyLink, copyMove, link, linkMove, move, all, uninitialized, none).
<code>items</code>	Returns a list of <code>dataTransfer</code> items with the drag data (see table B.12).
<code>setDragImage(element, x, y)</code>	Updates the drag feedback image with the given element and coordinates.
<code>addElement(element)</code>	Adds an element to the list of elements used to render drag feedback.

**Table B.9** `dataTransfer` object (*continued*)

Attribute/method	Description
<code>types</code>	List of data formats set in the <code>dragstart</code> event.
<code>getData(format)</code>	Returns the data being dragged.
<code>setData(format, data)</code>	Sets the data being dragged.
<code>clearData([format])</code>	Removes data of the specified format (or all formats if omitted).
<code>files</code>	Returns a list of files being dragged, if any.

Table B.10 lists the attributes and methods of the `dataTransfer` item. The `dataTransfer` item defines an object being dragged to the drop zone.

**Table B.10** `dataTransfer` item

Attribute/method	Description
<code>kind</code>	This is the kind of item being dragged (string or file).
<code>type</code>	This is the data item type string.
<code>getAsString(callback)</code>	If the data kind is string, this invokes the callback with the string data as an argument.
<code>getAsFile()</code>	If the data kind is file, this returns a <code>file</code> object.

Table B.11 lists the drag-and-drop events. When the application listens for these events, it can use the event object to gain access to the `dataTransfer` object or `dataTransfer` items. To access the `dataTransfer` object, use `e.dataTransfer`, where `e` is an event object. To access `dataTransfer` items, use `e.dataTransfer.items`, where `items` is a list of `dataTransfer` items.

**Table B.11** Drag-and-drop events

Event name	Description
<code>dragstart</code>	Fires on the source element when the user starts to drag the source element.
<code>drag</code>	Fires on the source element as the user is dragging the source element.
<code>dragenter</code>	Fires on the target element when the user drags the source element into it.
<code>dragleave</code>	Fires on the target element when the user drags the source element out of it.
<code>dragover</code>	Fires on the target element as the user is dragging the source element over it.
<code>drop</code>	Fires on the target element when the user drops the source element on it.
<code>dragend</code>	Fires on the source element when the user stops dragging the source element.

### B.1.5 Microdata API

The Microdata API (table B.12) has one method on the document object and a couple of DOM attributes on elements that have Microdata content attributes (`itemscope` and `itemprop`).

**Table B.12 Microdata API**

Attribute/method	Description
<code>document.getItems([type])</code>	Returns a list of top-level Microdata items. If you're looking for a particular type of Microdata item, such as event items, you can select all event items by specifying <code>'http://microformats.org/profile/hcalendar#event'</code> as the type parameter. Multiple types can be specified in a space-separated list.
<code>element.properties</code>	Gets the element's attributes (only if it has an <code>itemscope</code> attribute).
<code>element.itemValue</code>	Gets or sets the element's Microdata item value (only if it has an <code>itemprop</code> attribute).

### B.1.6 APIs for Web Storage

Web Storage defines APIs on two objects, `window.localStorage` and `window.sessionStorage`; see table B.13. The APIs for both of these objects are identical.

**Table B.13 localStorage and sessionStorage API**

Attribute/method	Description
<code>length</code>	Number of items (key/value pairs) currently stored in the storage area.
<code>key(index)</code>	Gets the name of the key at the given index.
<code>getItem(key)</code>	Gets the value of the item at the given key.
<code>setItem(key, value)</code>	Sets the value of the item at the given key to the value provided.
<code>removeItem(key)</code>	Removes the item at the given key.
<code>clear()</code>	Removes all items in the storage area.

Web Storage also defines an event, `storage`, that fires when the storage area changes. This event returns a storage event object, which contains attributes to determine what changed; see table B.14.

**Table B.14 Storage event object**

Attribute/method	Description
<code>key</code>	The key of the item that was modified
<code>oldValue</code>	The previous value of the modified item

**Table B.14 Storage event object (continued)**

Attribute/method	Description
<code>newValue</code>	The new value of the modified item
<code>url</code>	The address of the document that contains the item
<code>storageArea</code>	The storage object in which the change was made

The methods in table B.15 apply to `localStorage`, `sessionStorage`, and to cookies created using the `document.cookie` API. It's available on the `window.navigator` object.

**Table B.15 Another storage method**

Attribute/method	Description
<code>yieldForStorageUpdates()</code>	Allows scripts to access storage areas, even if other scripts are currently blocking those areas.

### B.1.7 Media Element API

The Media Element API, shown in table B.16, is implemented by both the `<audio>` and `<video>` elements.

**Table B.16 Media Element API**

Attribute/method	Description
<code>autoplay</code>	Corresponding DOM attribute to the <code>autoplay</code> content attribute.
<code>buffered</code>	Returns a <code>TimeRanges</code> object (an array of start and end times) that represents the ranges of the media resource that the browser has buffered.
<code>canPlayType(type)</code>	Accepts a MIME type, for example, <code>video/webm</code> , and returns a value indicating whether or not the browser thinks it will be able to play media of that type. The possible return values, in decreasing order of certainty, are <code>'probably'</code> , <code>'maybe'</code> , and an empty string.
<code>controller</code>	The <code>MediaController</code> object associated with the element's <code>mediagroup</code> .
<code>controls</code>	Corresponding DOM attribute to the <code>controls</code> content attribute.
<code>crossOrigin</code>	Reflects the value of the <code>crossorigin</code> content attribute. This setting is for Cross Origin Resource Sharing (CORS). The value can be either <code>anonymous</code> or <code>use-credentials</code> , depending on whether the <code>omit credentials</code> flag should be set or unset in the CORS headers.
<code>currentSrc</code>	The address of the currently playing media.
<code>currentTime</code>	The offset, in seconds, from the start of the media to the point currently playing.

Table B.16 Media Element API (continued)

Attribute/method	Description
defaultMuted	Corresponding DOM attribute to the muted content attribute.
defaultPlaybackRate	The default playback rate of the media; if this differs from the playbackRate, then the user is using fast forward or slow motion.
duration	The playing time, in seconds, of the media (if available).
ended	Boolean attribute that returns true if the media has reached the end of playback.
error	If any error has occurred, this attribute will be set to a MediaError object, which can be examined for the details.
load()	Resets the media element, clearing any currently playing media and rerunning the media-selection algorithm as if the page had just been loaded.
loop	Corresponding DOM attribute to the loop content attribute.
mediaGroup	Corresponding DOM attribute to the mediagroup content attribute. Allows the grouping of multiple media elements for synchronized playback.
muted	Boolean value indicating whether or not the current media is muted.
networkState	The state of any interaction between the media element and the network. Returns an integer value from 0 to 3, which corresponds to the constants NETWORK_EMPTY, NETWORK_IDLE, NETWORK_LOADING, and NETWORK_NO_SOURCE, respectively.
pause()	Sets the paused attribute to true, loading the media resource if necessary.
paused	Boolean value indicating whether or not the media is paused.
play()	Sets the paused attribute to false, loading the media and beginning playback if necessary. If the playback had ended, will restart it from the beginning.
playbackRate	The current effective playback rate; 1.0 is normal speed.
played	Returns a TimeRanges object (an array of start and end times) that represents the ranges of the media resource that the browser has played.
preload	Corresponds to the value of the preload content attribute; can have the value none, metadata, or auto.
readyState	The readiness of the element to play media. Returns an integer value from 0 to 4, which corresponds to the constants HAVE_NOTHING, HAVE_METADATA, HAVE_CURRENT_DATA, HAVE_FUTURE_DATA, and HAVE_ENOUGH_DATA, respectively.
seekable	Returns a TimeRanges object (an array of start and end times) that represents the ranges of the media resource that the browser is able to seek to (if any).



**Table B.16 Media Element API (continued)**

Attribute/method	Description
<code>seeking</code>	Boolean value indicating whether or not the browser is seeking (i.e., loading new data) because the playback position has been skipped forward.
<code>src</code>	Corresponds to the value of the <code>src</code> content attribute.
<code>startDate</code>	If the media has an embedded explicit time (for example, timestamped CCTV footage), this attribute will return the start date. This attribute was previously called <code>startOffsetTime</code> .
<code>volume</code>	Returns the current playback volume as a value between 0.0 and 1.0, inclusive.

## B.2 Other APIs and specifications

In this section, we cover the Geolocation API and the IndexedDB specification.

### B.2.1 Geolocation API

The Geolocation API methods are defined on the `window.navigator.geolocation` object. The `options` argument in the two position-retrieval API methods in table B.17 is a `Position Options` object and can have any of the attributes defined in table B.18.

**Table B.17 Geolocation API**

Attribute/method	Description
<code>getCurrentPosition(successCallback, errorCallback, [options])</code>	Gets the current position of the device, invoking the relevant success callback function when it has been located. If a problem is encountered, the error callback function will be called.
<code>watchPosition(successCallback, errorCallback, [options])</code>	Monitors the position of the device and invokes the relevant success callback provided as the location of the device is updated or the error callback if there's a problem. Calling this function returns a watch ID, which can be passed to <code>clearWatch</code> to cancel a watch.
<code>clearWatch(watchId)</code>	Clears an existing geolocation watch.

**Table B.18 Position Options object**

Attribute/method	Description
<code>enableHighAccuracy</code>	Informs the browser that the application would like to receive the maximum possible results. The browser can use this to determine whether it should use a more accurate sensor such as a Global Positioning System (GPS) sensor.
<code>timeout</code>	The maximum length of time (in milliseconds) allowed to pass before the relevant callback function is invoked.

**Table B.18** Position Options object (continued)

Attribute/method	Description
maximumAge	Typically, a device will store position information for a period of time to avoid wasting battery by having the position-detection hardware running constantly. If you're willing to accept slightly out-of-date position data, you can specify an acceptable maximum age in milliseconds in this parameter. If the value is 0 or omitted, the browser must fetch a new position, even if a cached position is available.

When one of the Geolocation API methods invokes a success callback function, it passes a Position object to that function; see table B.19.

**Table B.19** Position object

Attribute/method	Description
coords	A Coordinates object including the geographic coordinates of the user's location and the estimated accuracy. Further details are shown in table B.20.
timestamp	The time when the user's position was acquired.

coords, an attribute of the position object, lists the device's coordinates, the estimated accuracy of those coordinates, the device's direction of travel, and its speed.

**Table B.20** Coordinates object

Attribute/method	Description
latitude	Geographic latitude coordinate, in degrees
longitude	Geographic longitude coordinate, in degrees
altitude	The height, in meters, above (approximately) sea level
accuracy	The accuracy of the latitude and longitude values, in meters
altitudeAccuracy	The accuracy of the altitude value, in meters
heading	The direction the device is traveling in, specified in degrees
speed	The device's current velocity in meters per second

### B.2.2 IndexedDB specification

IndexedDB is a very large specification, approximately 105 printed pages, so there's not room in this appendix to discuss every single attribute, method, and the like. Instead, this section lists only the most important components used in this book. These components have been grouped under their respective IndexedDB interfaces, and presented in a table format. Summaries for each component have been prepared by Joe Lennon and Greg Wanish and are derived from IndexedDB content (<http://mng.bz/1M6o>) by

Mozilla contributors at the Mozilla Developer Network (MDN) and used under Creative Commons CC-BY-SA (<http://creativecommons.org/licenses/by-sa/2.5/>). These tables of IndexedDB interfaces are licensed under Creative Commons CC-BY-SA (<http://creativecommons.org/licenses/by-sa/2.5/>) by Joe Lennon and Greg Wanish. See <http://mng.bz/1M6o> for a more complete explanation of the IndexedDB specification.

The object `window.indexedDB` implements the `IDBFactory` interface and enables applications to create, access, and delete an indexed database. Table B.21 lists the methods and attributes for the asynchronous version of the `IDBFactory` interface. The asynchronous version works with or without web workers; no browser at this time supports the synchronous version.

**Table B.21** `IDBFactory` interface

Attribute/method	Description
<code>open(name, [version])</code>	Requests a connection to a database with given name and version number. If no database with name exists, create a database with given name and version number.
<code>deleteDatabase(name)</code>	Requests deletion of a database with given name.
<code>cmp(first, second)</code>	Compares two keys to determine equality and ordering for IndexedDB operations, such as ordering. Returns a -1, if first key is less than second key; 0, if first key is equal to second key; 1, if first key is greater than second key.

Table B.22 lists the attributes and methods of the `IDBCursor` object. The cursor iterates over object stores and indexes within an indexed database.

**Table B.22** `IDBCursor` interface

Attribute/method	Description
<code>source</code>	On getting, returns the <code>IDBObjectStore</code> or <code>IDBIndex</code> that the cursor is iterating over.
<code>direction</code>	On getting, returns the cursor's current direction of traversal.
<code>key</code>	On getting, returns the key for the record at the cursor's position. If the cursor is outside its range, this is <code>undefined</code> .
<code>primaryKey</code>	On getting, returns the cursor's current effective key. If the cursor is currently being iterated or has iterated outside its range, returns <code>undefined</code> .
<code>update(value)</code>	Returns an <code>IDBRequest</code> object. In a separate thread, uses <code>value</code> to update the value at the current position of the cursor in the object store. If the cursor points to a record that has just been deleted, a new record is created with the given value.
<code>continue(key)</code>	Continues along the cursor's current direction of movement, and finds the next item with a key matching the optional <code>key</code> parameter. If no key is specified, goes to the immediate next position, based on the cursor's current direction of movement.

**Table B.22** IDBCursor interface (continued)

Attribute/method	Description
<code>delete()</code>	Returns an IDBRequest object. In a separate thread, deletes the record at the cursor's position without moving the cursor. Afterward, the cursor's value is set to null.

Table B.23 lists the methods and attributes of the IDBDatabase object. The IDBDatabase serves primarily as a container for indexes and object stores. The IDBDatabase object is the only way to get a transaction on the database.

**Table B.23** IDBDatabase interface

Attribute/method	Description
<code>createObjectStore(name, [parameters])</code>	Creates and returns a new object store or index with a given name. <code>parameters</code> is an optional object with the following properties: <ul style="list-style-type: none"> <li><code>keyPath</code> Specifies a field in the object as a key. Each object must have a unique key.</li> <li><code>autoIncrement</code> If true, the object store creates keys automatically via a key generator.</li> </ul>
<code>setversion (deprecated)</code>	Updates the version of the database. Upon invocation, returns immediately, and, on a separate thread, runs a <code>versionchange</code> transaction on the connected database.
<code>transaction(storeNames, [mode])</code>	Immediately returns an IDBTransaction object and, on a separate thread, starts a transaction. The parameter <code>storeNames</code> , an array of strings, identifies the object stores and indexes that are to be accessible to the new transaction. The <code>mode</code> parameter defines the new transaction's type of access: 'readonly' or 'readwrite'. The default is 'readonly'.
<code>version</code>	The version of the connected database. When a database is first created, this attribute is the empty string.

Table B.24 defines the interface for the IDBEnvironment object; it has only a single attribute, `indexedDB`. The IDBEnvironment provides access to a client-side database.

**Table B.24** IDBEnvironment interface

Attribute/method	Description
<code>indexedDB</code>	Provides a mechanism for applications to asynchronously access the capabilities of indexed databases.

Table B.25 lists the IDBIndex object method, `openCursor`. This method is useful for filtering through an index. The IDBIndex provides methods to access an index of a database.

**Table B.25** IDBIndex interface

Attribute/method	Description
<code>openCursor([range], [direction])</code>	Immediately returns an <code>IDBRequest</code> object, then, on a separate thread, creates a cursor over the specified key range. The optional parameter <code>range</code> specifies the key range of the cursor. The other optional parameter, <code>direction</code> , specifies the cursor's direction of movement through the index.

Table B.26 lists some of the methods for creating indexes and working with the object store. These methods belong to the `IDBObjectStore` object.

**Table B.26** IDBObjectStore interface

Attribute/method	Description
<code>createIndex(name, keypath, [parameters])</code>	Creates and returns a new <code>IDBIndex</code> object with given name and keypath. This method can only be called from a <code>VersionChange</code> transaction mode callback. The optional <code>parameters</code> object has the following properties: <ul style="list-style-type: none"> <li>▪ <code>unique</code></li> <li>▪ <code>multiEntry</code></li> </ul>
<code>index(name)</code>	Returns the name index in the object store.
<code>openCursor([range], [direction])</code>	Immediately returns an <code>IDBRequest</code> object, then, on a separate thread, creates a cursor over the records in the object store. The <code>range</code> parameter specifies the key range of the cursor. If the <code>range</code> is not specified, it defaults to all records in the object store. The <code>direction</code> parameter defines the cursor's direction of movement.
<code>put(value, [key])</code>	Immediately returns an <code>IDBRequest</code> object, then, on a separate thread, creates a clone of the value and stores it in the object store. The <code>value</code> parameter defines the value to be stored. The parameter <code>key</code> identifies the record. If not defined, it defaults to <code>null</code> .

Table B.27 lists the `onupgradeneeded` event handler used in the My Tasks application. `onupgradeneeded` is an event in the `IDBOpenDBRequest` interface that provides access to results of requests to open a database using event handler attributes.

**Table B.27** IDBOpenDBRequest interface

Attribute/method	Description
<code>onupgradeneeded</code>	The event handler attribute for the upgrade needed event. This event handler is executed when a database's version number has increased.

Table B.28 lists the `onsuccess` event handler used in the My Tasks application to access the results of an asynchronous request. `onsuccess` is an event in the `IDBRequest` interface that provides access to results of asynchronous requests to databases and database objects using event handler attributes. Reading and writing operations on a database are executed with a request.

**Table B.28** IDBRequest interface

Attribute/method	Description
<code>onsuccess</code>	The event handler attribute for the <code>success</code> event.

Table B.29 shows the method of the `IDBKeyRange` object used to search for keys within the index created for the My Tasks database. The `IDBKeyRange` interface defines a range of keys.

**Table B.29** IDBKeyRange interface

Attribute/method	Description
<code>bound(lower, upper, [lowerOpen], [upperOpen])</code>	Creates and returns a key range with <code>upper</code> and <code>lower</code> bounds. If optional parameter <code>lowerOpen</code> is <code>false</code> (the default value), then the range includes the lower bound of the key range. If optional parameter <code>upperOpen</code> is <code>false</code> (the default value), then the range includes the upper bound value of the key range.

### B.3 File System API

The File System API is massive; it will change how people think about managing web application data. In this section, we cover directory-based APIs within the File System API, as well as Blob data APIs. The following tables will give you some references and shortcuts for better managing your file data.

Table B.30 lists attributes associated with a `File` object.

**Table B.30** File API

Attribute/method	Description
<code>name</code>	The name of the file
<code>size</code>	The size of the file in bytes
<code>type</code>	The MIME type of the file

Table B.31 lists attributes and methods associated with the `FileList` object. A `FileList` object is returned by the `files` property of the HTML `<input>` element.

**Table B.31** FileList API

Attribute/method	Description
length	Number of files in the list.
item(index)	Gets the file at the given index (zero-based).

Table B.32 lists attributes and methods associated with the `FileReader` object. A `FileReader` object lets web applications asynchronously read the contents of files (or raw data buffers) stored on the user's computer, using `File` or `Blob` objects to specify the file or data to read.

**Table B.32** FileReader API

Attribute/method	Description
<code>abort()</code>	Aborts reading the file
<code>readAsArrayBuffer(blob)</code>	Reads the contents of the blob (which is either a <code>File</code> or a <code>Blob</code> object) into an array buffer.
<code>readAsDataURL(blob)</code>	Reads the contents of a <code>Blob</code> or <code>File</code> object and returns a <code>data:</code> URL to it.
<code>readAsText(blob, [encoding])</code>	Reads the contents of a <code>Blob</code> or <code>File</code> object into a text string if the optional encoding parameter is specified (e.g., 'ISO-8859-1' or 'UTF-8'); then the string will be encoded using that character set.
<code>error</code>	If an error occurs, it will be loaded into this property.
<code>readyState</code>	The state of the file read operation (0 = EMPTY, 1 = LOADING, 2 = DONE).
<code>result</code>	This will be populated with the file's contents when a read operation has been completed. The format of the result will depend on the method used to read the file.

Table B.33 lists events associated with the `FileReader` object.

**Table B.33** FileReader events

Event name	Description
<code>abort</code>	Fires when the read operation is aborted.
<code>error</code>	Fires when an error occurs while reading the file.
<code>load</code>	Fires when the read operation has successfully completed.
<code>loadend</code>	Fires after <code>onload</code> or <code>onerror</code> , regardless of whether the operation was successful.

**Table B.33** FileReader events (*continued*)

Event name	Description
loadstart	Fires when the read operation is about to start.
progress	Fires periodically during the read operation.

Table B.34 lists methods associated with the FileWriter object. A FileWriter object can perform multiple write actions, rather than just saving a single Blob.

**Table B.34** FileWriter API

Attribute/method	Description
seek(offset)	Sets a specific file location at which the next write will occur.
truncate(size)	Alters the length of the file to the size passed in bytes.
write(data)	Writes the input data to a Blob object.

Table B.35 lists the methods associated with the FileSaver object which has methods to write a Blob object to a file.

**Table B.35** FileSaver API

Constructor/attribute/method	Description
FileSaver(data)	Creates a FileSaver object with Blob data.
abort()	Terminates file saving.

Table B.36 lists the events associated with the FileSaver object which has events to monitor the progress of writing a Blob to a file.

**Table B.36** FileSaver events

Event name	Description
writestart	Fires when starting a writing event.
progress	Fires repeatedly while file is being written.
write	Fires when a file is being written to.
abort	Fires when file writing is canceled.
error	Fires in response to an error or an abort.
writeend	Fires when writing to a file has ended.

Table B.37 lists the methods associated with the FileEntry object. A FileEntry object has methods to write and inspect the state of a file.



**Table B.37 FileEntry API**

Constructor/attribute/method	Description
<code>createWriter(success, error)</code>	Creates a new <code>FileWriter</code> associated with the file that <code>FileEntry</code> represents. If successful, calls function <code>success</code> ; otherwise calls function <code>error</code> .
<code>file(success, error)</code>	Returns a file that represents the current state of the file that the <code>FileEntry</code> represents. If successful, calls function <code>success</code> ; otherwise calls function <code>error</code> .

### B.3.1 Directory-based APIs within the File System API

The File System API contains APIs to read directory entries in a directory. It also contains APIs to create, read, look up, and recursively remove files in a directory. Directory entries are objects that describe either a file or subdirectory. A directory entry contains attributes defining the entry's status as a file or subdirectory, the pathname to the entry, and the filesystem containing the entry.

Table B.38 lists the methods for the directory entry object. This object represents a directory entry in a filesystem. It includes methods for creating, reading, looking up, and recursively removing files and subdirectories in a directory.

**Table B.38 DirectoryEntry API**

Constructor/attribute/method	Description
<code>createReader()</code>	Creates a new <code>DirectoryReader</code> object to read the directory.
<code>getDirectory(path, [options], [success], [error])</code>	Creates or looks up a directory depending on set options. Successful creation or location is handled by the <code>success</code> callback; any errors will cause the <code>error</code> callback to be executed.
<code>getFile(path, [options], [success], [error])</code>	Creates or looks up a file depending on set options. Successful creation or location is handled by the <code>success</code> callback; any errors will cause the <code>error</code> callback to be executed.
<code>removeRecursively(success, [error])</code>	Deletes a directory and all contents; may only partially delete a directory if an error occurs. Successful creation or location is handled by the <code>success</code> callback; any errors will cause the <code>error</code> callback to be executed.

Table B.39 lists the only method for the `DirectoryReader` object.

**Table B.39 DirectoryReader API**

Attribute/method	Description
<code>readEntries(success, error)</code>	Allows you to read the next block of entries from the current directory, with a successful read being handled by the <code>success</code> callback and errors being handled by the <code>error</code> callback.

### B.3.2 Blob data APIs

A Blob is an object of immutable data. Blobs are usually used to store the contents of a file. Part of the File API is inherited from the Blob API. Table B.40 lists the methods and attributes of a Blob.

**Table B.40** Blob interface

Constructor/attribute/method	Description
<code>blob([array], [attributes])</code>	Creates a Blob object without <code>BlobBuilder</code> . The array can be any number of <code>ArrayBuffer</code> , <code>ArrayBufferView</code> (typed array), <code>Blob</code> , or <code>DOMString</code> objects, in any order. <code>attributes</code> is an object that can specify the media type and line endings in the <code>type</code> and <code>ending</code> properties, respectively.
<code>size</code>	Size in bytes of Blob's data; read only.
<code>type</code>	MIME type of the Blob's data.
<code>slice([start], [end], [type])</code>	Returns a specific chunk of Blob data, from offset <code>start</code> to offset <code>end</code> with MIME type <code>type</code> .

Table B.41 lists the methods for the `BlobBuilder` object. The `BlobBuilder` provides a way to construct Blob objects by calling one or more `append` methods on the `BlobBuilder` object. This API has been deprecated.

**Table B.41** BlobBuilder API

Attribute/method	Description
<code>append(ArrayBuffer)</code>	Appends the <code>ArrayBuffer</code> to the Blob.
<code>append(Blob)</code>	Appends the <code>Blob</code> parameter to the Blob.
<code>append(data, [endings])</code>	Appends the string data to the Blob. The <code>endings</code> parameter specifies how strings containing <code>\n</code> are to be written out. This can be <code>'transparent'</code> (endings unchanged) or <code>'native'</code> (endings changed to match host system convention).
<code>getBlob([contentType])</code>	Returns the Blob object that's the result of all the <code>append</code> operations. If specified, the content type will be set on the returned Blob. This operation will also empty the <code>BlobBuilder</code> of all data.
<code>getFile(name, [contentType])</code>	Returns a file object with an optional content type.

# appendix C

## Installing PHP and MySQL

---

To make the SSE Chat application from chapter 4 work, you'll need to set up a web server with PHP and MySQL. This combination is available free from various online providers, but setting up your own local install will allow you to experiment more freely. In this appendix we'll walk you through setting up PHP and then MySQL on Windows 7 and Mac OS X Mountain Lion.

### **C.1 Installing PHP on Windows 7**

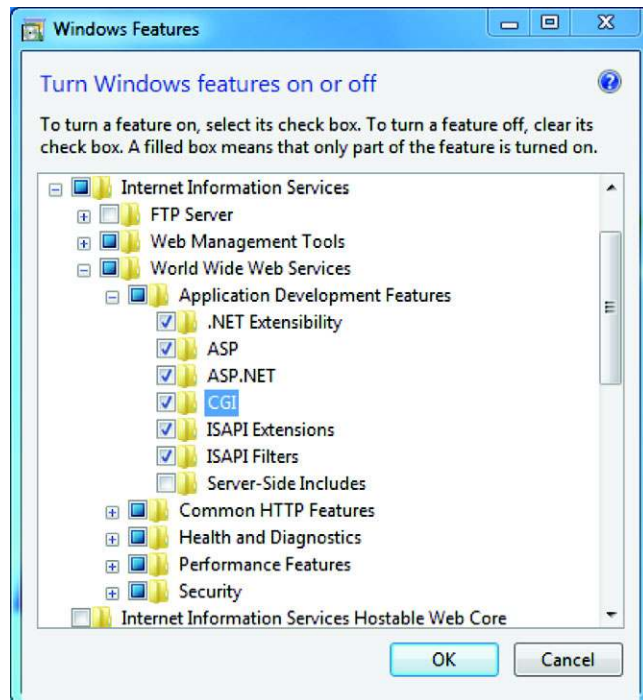
In this section you're going to download and install PHP and get it working with Windows's built-in web server component, Internet Information Services (IIS).

#### **C.1.1 Configuring Windows 7 IIS**

IIS is not installed by default in Windows 7 but can be added through the Control Panel option Turn Windows Features On and Off. Follow three steps to install IIS:

- 1 Open Control Panel and use the search feature to locate the Turn Windows Features On and Off option. Double-click it, and you'll see a dialog box like the one shown in figure C.1.

In figure C.1 the functionality is divided into a tree of options. A check mark shows that the feature and all its subfeatures are installed. A blue square indicates that the feature, but only some of the subfeatures, are installed. Selecting a feature with subfeatures will select the default set of subfeatures; this isn't necessarily *all* of the subfeatures. In the figure you can see that Application Development Features is selected, but only six of the seven subfeatures are selected.



**Figure C.1** Adding the IIS components to Windows 7

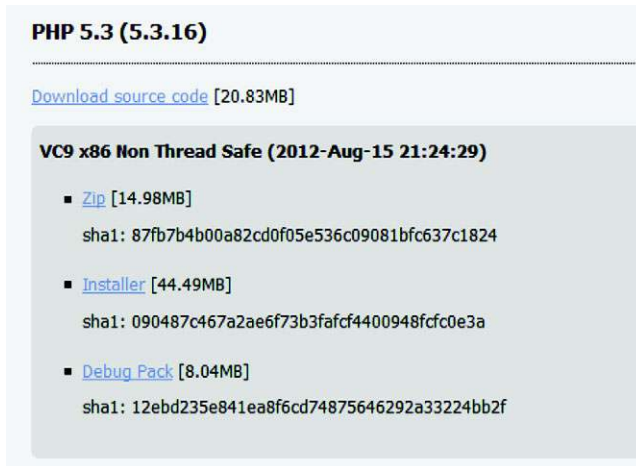
- 2 Ensure that the options for IIS, World Wide Web Services, and, under the Application Development Features section, CGI are all selected. Selecting IIS will automatically select World Wide Web Services but not the CGI feature. Make sure you expand the tree and select the CGI feature explicitly.
- 3 After you make all your changes, click OK. There will be a short delay while the new features are installed.

### **C.1.2** Downloading PHP

PHP installers for Windows are available from <http://windows.php.net/download/>; look for the links that say “Installer.” To follow along with us, use the latest 5.3 version (5.3.16 at the time of writing, see figure C.2), which has an Installer option. The installer will do a lot of automatic setup for you, so it’s the better option even if it’s not the most recent version on the page.

One other feature you should notice on the download page is that the Windows binaries are available in Thread Safe and Non Thread Safe varieties. The difference is only relevant if you want to integrate PHP with Apache; for installing PHP with IIS, you want the Non Thread Safe version, so download that now.

After clicking the link, you should have a file called php-5.3.16-nts-Win32-VC9-x86.msi (or a similar name with a larger version number) to use in the next step.

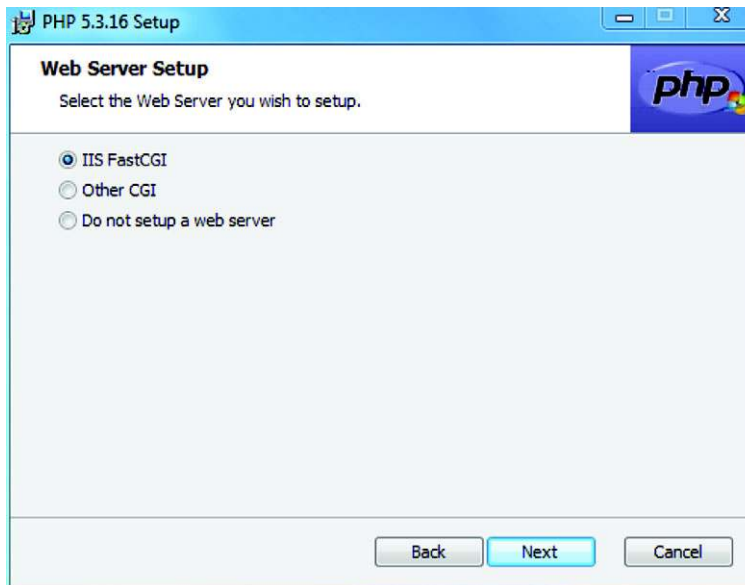


**Figure C.2** The download page at php.net; use the latest 5.3 version to follow along as you read.

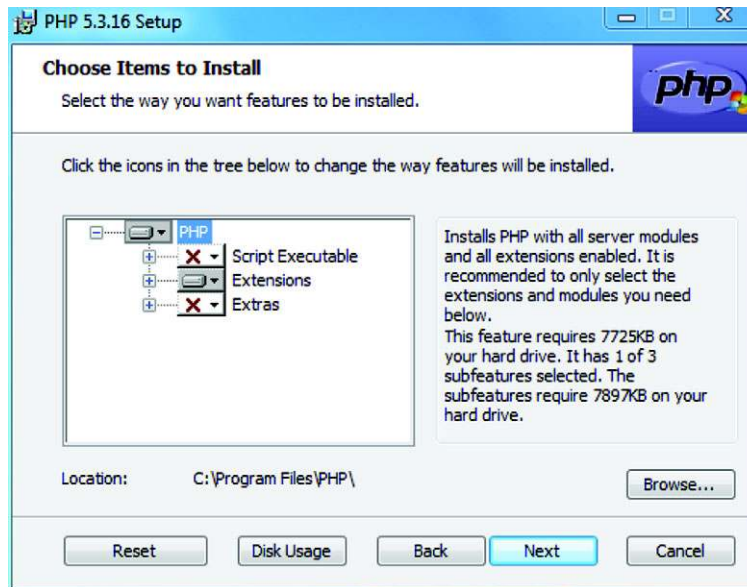
### C.1.3 *Installing PHP*

Now that you have the installation files downloaded you're ready to install PHP by following these steps:

- 1 The MSI file you downloaded in C.1.2 will do most of the work for you. There are only two steps, which we'll walk you through, where you have to make decisions. Double-click the file to start, and accept the license agreement and the default file location.
- 2 For IIS configuration, select the option IIS FastCGI, which appears in the first decision screen, as shown in figure C.3. Note that this is why we had you take special care to select the CGI option earlier.



**Figure C.3** Selecting the web server configuration in the PHP setup



**Figure C.4** Select the PHP components to install

- 3 When you see the next decision screen (figure C.4), accepting the defaults should be fine, but just in case, you want both PHP and Extensions selected.

Continue to the end of the installer, and you'll have a working PHP installation. As a final step, let's check that everything is working.

### C.1.4 Confirm PHP is installed

IIS by default will serve files from the directory `C:\inetpub\WWWRoot\`.

- 1 Create a file in that directory called `index.php`. Add the following code to it:

```
<?php phpinfo(); ?>
```

- 2 Load the URL <http://localhost/index.php> in your web browser. You should see a page like the one shown in figure C.5.

When it comes time to run chapter 4's SSE Chat application, you can make this work in a similar way; copy the entire working folder into `C:\inetpub\WWWRoot\`, then browse to <http://localhost/sse-chat/index.php> (substitute `sse-chat` for whatever name you gave your working directory). Now that you have PHP installed, it's time to move on to setting up MySQL.

## C.2 Installing MySQL on Windows 7

MySQL also has a convenient MSI-based installation process, which will take care of everything for you. In this section you'll walk through downloading and installing the database and client tools and then creating a database for use with the sample application in the book.

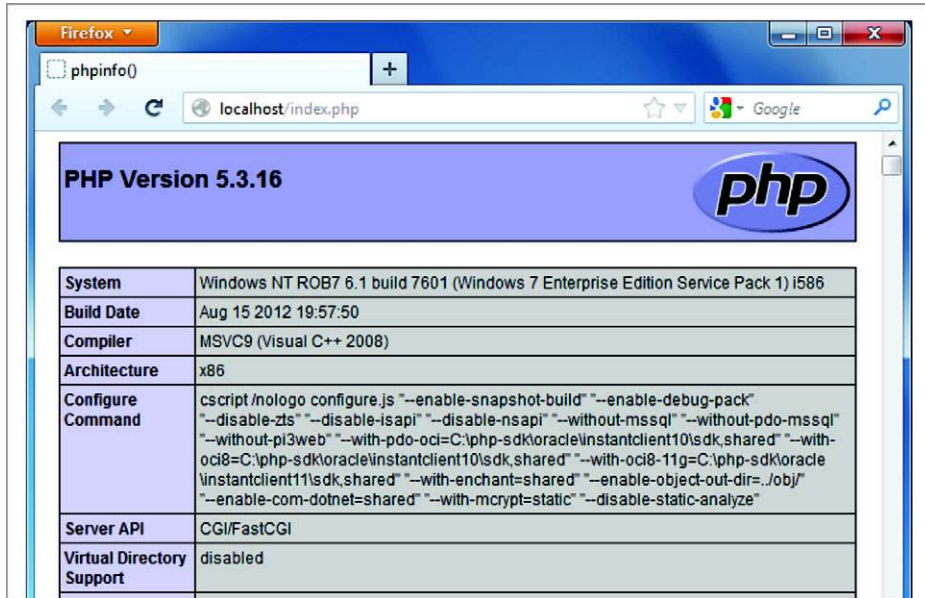


Figure C.5 PHP is successfully installed.

### C.2.1 Downloading MySQL

MySQL can be downloaded from <http://dev.mysql.com/downloads/>. The Download button is hard to miss because it's prominently displayed in the middle of the page, as you can see from figure C.6.



Figure C.6 The Download button is very prominent on the MySQL website.

**Begin Your Download - mysql**

Please take the time to let us know about you.

If this is the first time you have downloaded from us, we will ask you to log into all of the MySQL web sites, including MySQL.com.

If you already have a MySQL.com account, save time by logging in.

**Returning Users**

Save time by logging in

Email:

Password:

[Forgot your password?](#)

[» No thanks, just start my download!](#)

**Figure C.7** Click the “No thanks” link at the bottom of the page to download without registering.

- 1 Click the button to download.
- 2 On the next page, you’ll be presented with an option to create an account (figure C.7). You don’t have to do this, although you can if you want to. To start the download, just click the link at the bottom that says, “No thanks, just start my download!”

## C.2.2 Installing MySQL

In this section, you’ll install the MySQL server:

- 1 You should now have an MSI file called `mysql-installer-community-5.5.27.3.msi`, except you’ll have a more recent version number; double-click it to start.
- 2 Although you should be able to accept the defaults at every step to get a working installation, the next few steps highlight a few of the screens involved to help you stay on track. When you get to the Setup Type screen, shown in figure C.8, make sure the option Developer Default is selected.

Selecting the Developer Default option will install all the necessary tools to run and manage a local database instance.

- 3 When you get to the Configuration page, shown in figure C.9, you don’t have to change the defaults, but you should consider whether you really want your MySQL Server available to anyone on your local network. If you want only local connections allowed, deselect the option Enable TCP/IP Networking. If you



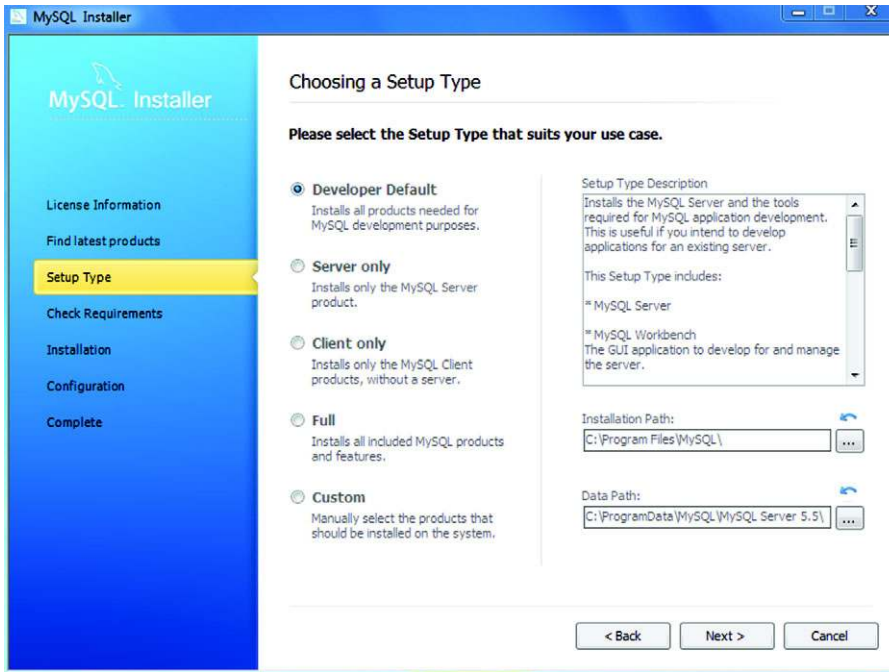


Figure C.8 Choosing the MySQL setup type

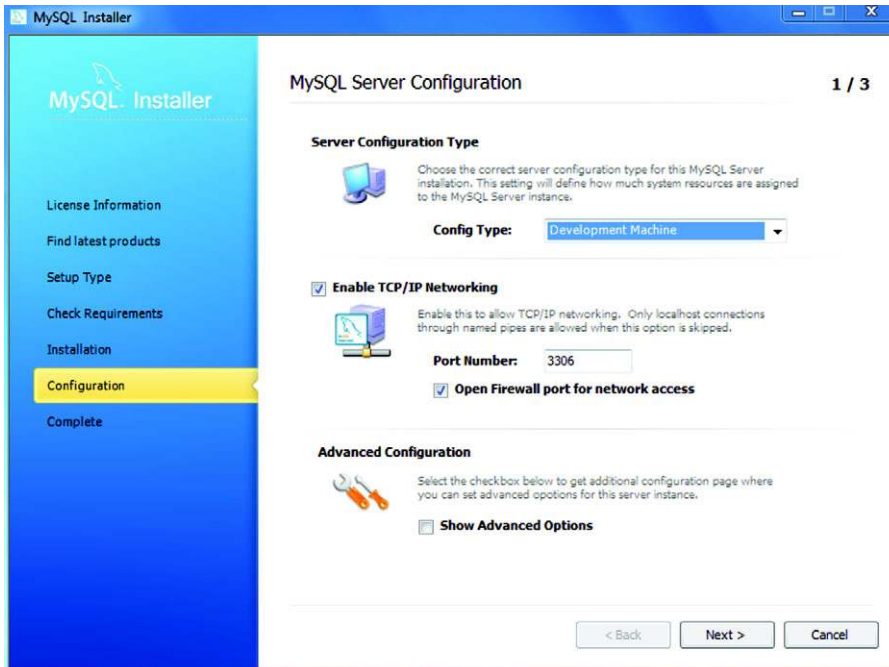
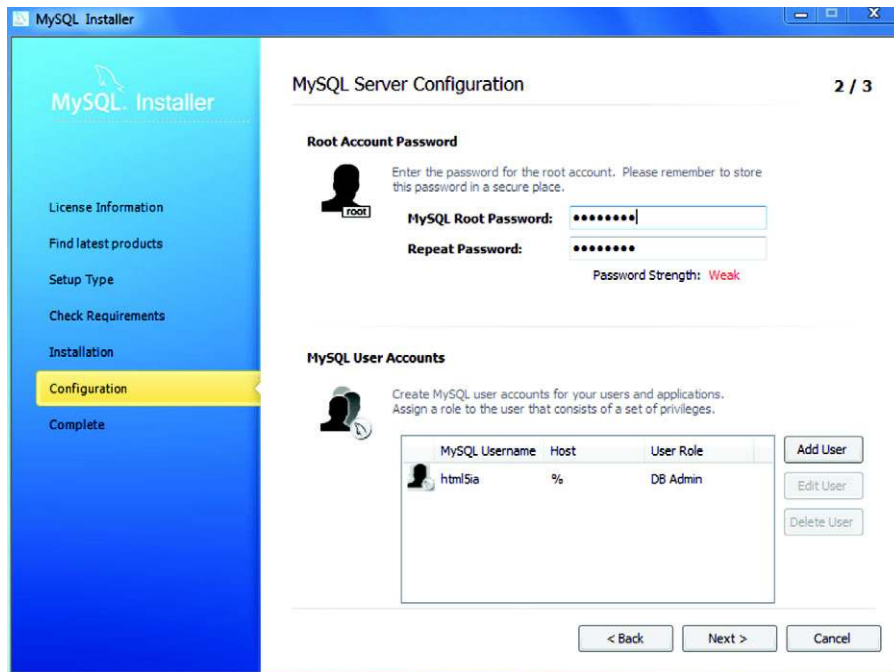


Figure C.9 The MySQL installer Configuration page



**Figure C.10** Setting the root password

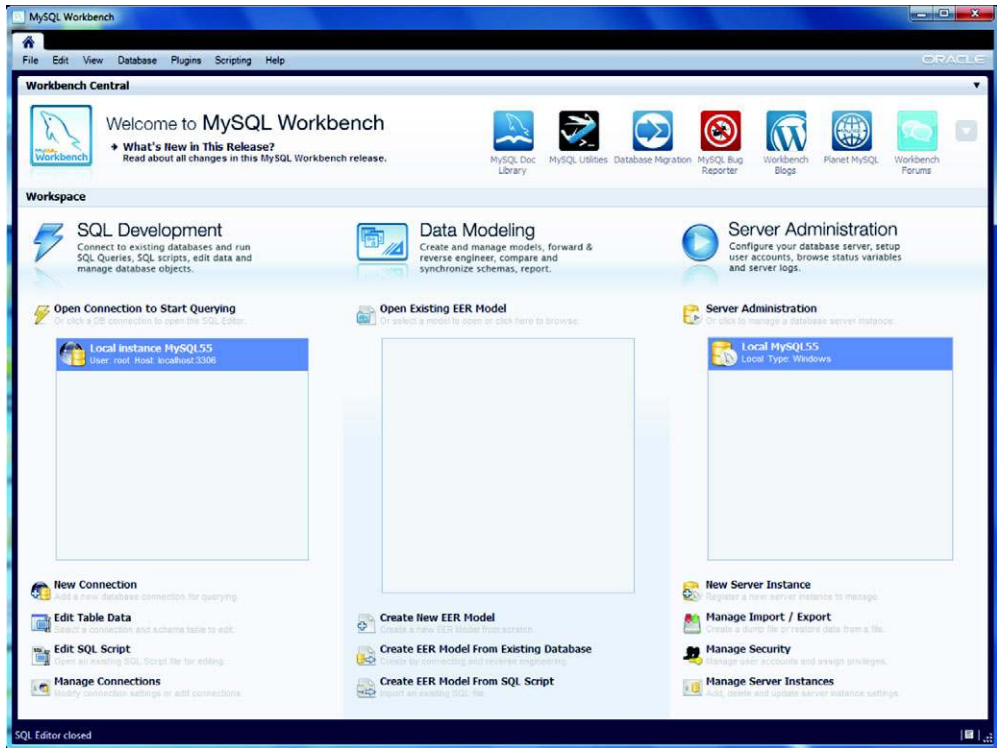
spend a lot of time connected to public Wi-Fi networks, you should definitely deselect this option.

Figure C.10 shows the next key configuration step, setting the root password. Although it's important to set a strong password, it's also important to set a memorable one. If you forget this password, you won't be able to access the database server. If you deselected the option to allow network access in the previous screen, then the password strength is less of an issue. On this screen you can also create other user accounts and assign them to various administrative roles within the database server. This isn't necessary to get anything in this book working but shouldn't break anything if you'd like to add some.

- 4 Enter a password, and click Next until the installer has finished.
- 5 At the end of the process, the installer will ask if you want to launch MySQL Workbench now; click Yes before proceeding to the next section. This is a tool for managing databases and running scripts; in the next section you'll use it to create a database you can use for the SSE Chat application.

### **C.2.3** *Creating a database and running scripts*

Having a database server available is only half the battle; you also need to create a database on that server for your app to use. In this section you'll create a database and



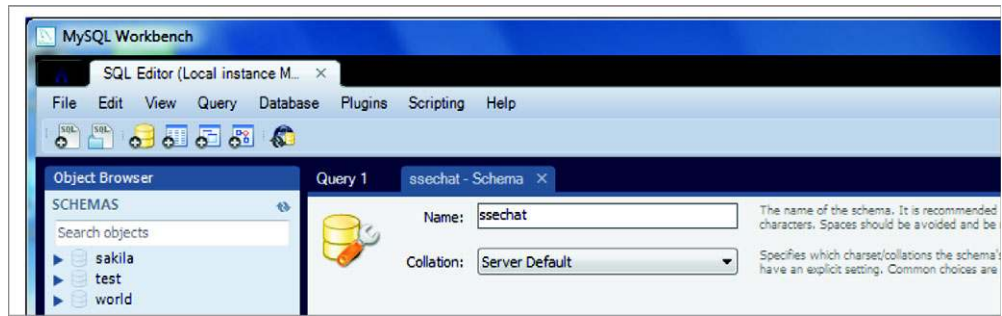
**Figure C.11** The home page of MySQL Workbench

add the required structures for the chapter 4 SSE Chat app by running the chat.sql script provided in the code download for that chapter. Before you start, make sure you can see the MySQL Workbench welcome screen shown in figure C.11.

The first task is to connect to your new database server. Double-click the local instance in the leftmost box on the Welcome screen, under the heading Open Connection to Start Querying. You'll then be asked to enter your root password, as shown in figure C.12.



**Figure C.12** The enter root password dialog box

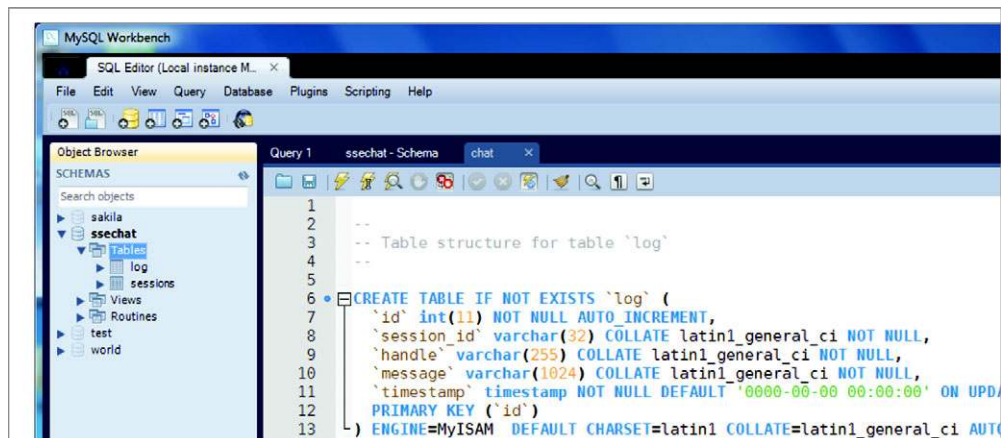


**Figure C.13** Creating a database

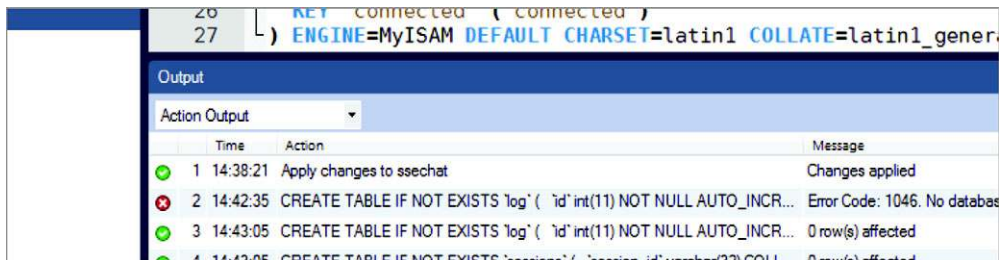
Type in the password you set earlier and click OK. You'll be taken to the SQL Editor screen. In the left pane you'll see a list of databases (MySQL Workbench calls them Schemas), and on the right is a text editor to use to enter queries.

The second task is to create a database. On the toolbar you'll see an icon of a yellow cylindrical object with a plus sign in front of it; it's the third icon from the left.

- 1 Click that third icon, and you should see the create database dialog box shown on the right side of figure C.13.
- 2 Enter a suitable name like ssechat. Click the Apply button toward the bottom of the screen. Confirm that the script is being run, as shown in figure C.14, which will create the database for you.
- 3 Open the chat.sql file from the chapter 4 code download; the File menu has all the usual options for this sort of thing.
- 4 Run the script on the database you've just created by selecting the Execute (All or Selection) option from the Query menu. This will set up the tables required for the app.



**Figure C.14** The chat.sql file open in the MySQL workbench



**Figure C.15** If you see an error 1046, it's because you've not selected a database for running the query.

Note that if you see an error 1046 like the one shown in figure C.15, this is because the database isn't selected.

If you get that error, double-click the database name in the left pane and run the script again.

You now have PHP and MySQL set up and working on your Windows 7 machine.

### C.3 *Installing PHP and MySQL on Mac OS X Mountain Lion*

All recent versions of Mac OS X come equipped with Apache and PHP. By default, Apache is not running, nor is it configured to load PHP when it runs. To get everything running, you'll need to follow along with a few steps.

#### C.3.1 *Configuring Apache and PHP*

To get PHP running on your computer you must first edit a couple of Apache configuration files. By default, these files are hidden from the Finder, so the easiest way to access them is through the Terminal application. Don't worry if you're not familiar with the Terminal and command line in OS X; just follow along and you should be okay.

**NOTE** For brevity, when we display the command line we'll simply use \$ to represent the prompt. Any bold text is text that you'll type in, and any non-bold text is what will appear in the Terminal.

#### USING THE TERMINAL

The Terminal app can be found in the Applications/Utilities folder on your system.

Open the Terminal and you'll be presented with a greeting message followed by a prompt that looks similar to this:

```
MacBook:~ scott$
```

The first part of the prompt, MacBook, is the hostname of your computer; this will most likely be different on your computer. (The hostname can be set to whatever you'd like in the Computer Name: text field of the Sharing System Preference pane.) After the colon (:) is your current path. The path represents what folder you are currently in. Most likely, when you start the terminal you're in the Home directory (/Users/YourUsername); Terminal abbreviates a user's Home directory with the ~ symbol. After that the

prompt shows you your username (which won't be scott unless that's actually your username) followed by the \$ prompt and a cursor awaiting your input.

Our first Terminal command will take you to the location where the Apache configuration files are stored:

```
$ cd /etc/apache2/
```

This will take you to the apache2 folder where the configuration files are kept. (Note that after you type this command the ~ in the prompt changes to apache2.) It does this with the cd (change directory) command, which tells the terminal to go to a specific directory.

Next, let's look at the files in this directory:

```
$ ls -FG
```

You should get a response showing the following:

```
extra/          magic          original/     users/
httpd.conf     mime.types    other/
```

The ls (list directory) command lists the contents of a directory. The -FG part is flags that add features to the basic ls command. In this case the -F adds symbols to special files (in this case the trailing / for subdirectories) and the -G adds color to special files. These two are slightly redundant, but they make the listing prettier.

At this point, your first step is to edit the httpd.conf file. This is the master Apache configuration file.

#### EDITING APACHE CONFIGURATION FILES

Editing the httpd.conf file involves a few tricks. By default, only a superuser (aka root) can edit this file; for this reason most graphical text editors (including any downloaded through the App Store) will refuse to save any changes to this file (see figures C.16 and C.17)

**NOTE** Some graphical text editors will allow you to unlock and edit files like httpd.conf, but such capabilities aren't allowed in applications found in the App Store. For example, BBEdit, which is available in the App Store, will allow you to edit httpd.conf, but if you purchase the App Store version you'll need to download an additional file from the BareBones website to enable this feature.



**Figure C.16** When you try to edit httpd.conf in most text editors, you'll first get a warning saying the file is locked.



**Figure C.17** If you try to unlock `httpd.conf`, in most text editors you'll get another warning saying that it can't be unlocked here.

So, if there are many roadblocks to editing the `httpd.conf` file, how do you go about it? It's not too difficult from the Terminal app using `sudo` along with a command-line text editor.

**NOTE** The `sudo` (switch user and do) command is available to any user on Mac OS X with Admin rights. Most users have Admin rights to their Mac. But if a business, school, parent, or untrusting spouse provided your computer for you, you may not have Admin rights. If this is the case, you can't continue on your own; rather you should bug the person who provided you your computer incessantly until they either give you Admin rights to your system or set all of this up for you.

To begin, though, we'll test out `sudo` and create a backup copy of `httpd.conf` just in case, all at the same time with the following:

```
$ sudo cp httpd.conf httpd.conf.orig
Password:
```

After typing this command you'll be prompted to enter your system password to complete the command. Also, if this is the first time you've used `sudo`, you'll be given a warning about the dangers of using `sudo` inappropriately. Upon successfully typing in your password, you can run the `ls` command, and you should see a new `httpd.conf.orig` file listed. If not, something went wrong (check the previous note about being Admin).

Assuming you were able to create a copy of `httpd.conf`, you should be ready to go, assured that even if you do something horribly wrong, you can recover using your backup file. So begin the editing with

```
$ sudo nano httpd.conf
```

Now, because you recently ran the `sudo` command to create your backup, you may not be prompted again for your password. `sudo` will remember you for short periods of time between `sudo` commands, so you don't need to enter your password every time you run the command.

This command will open the `httpd.conf` file in the nano text editor with superuser permissions, allowing you to edit and save the file. As a result, nano will take over your terminal screen, which should now look something like this:

```

GNU nano 2.0.6                               File: httpd.conf
#
# This is the main Apache HTTP server configuration file.  It contains the
# configuration directives that give the server its instructions.
# See <URL:http://httpd.apache.org/docs/2.2> for detailed information.
# In particular, see
# <URL:http://httpd.apache.org/docs/2.2/mod/directives.html>
# for a discussion of each configuration directive.
#
# Do NOT simply read the instructions in here without understanding
# what they do.  They're here only as hints or reminders.  If you are unsure
# consult the online docs.  You have been warned.
#
# Configuration and logfile names: If the filenames you specify for many
# of the server's control files begin with "/" (or "drive:/" for Win32), the
# server will use that explicit path.  If the filenames do *not* begin
# with "/", the value of ServerRoot is prepended -- so "log/foo_log"
# with ServerRoot set to "/usr" will be interpreted by the
# server as "/usr/log/foo_log".

[ Read 500 lines ]
^G Get Help   ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text    ^C Cur Pos
^X Exit       ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell

```

Now your primary goal in this file is to set up and enable PHP. To do this you need to scroll down to the directive that loads the PHP module. By default this should be on line 117. You can use the Ctrl+Shift+\_ keyboard shortcut to invoke the Enter line number, column number: command in nano and enter 117 to go directly to line 117. Alternatively, just scroll down using the down-arrow key until you reach the part of the file that looks like this:

```

LoadModule alias_module libexec/apache2/mod_alias.so
LoadModule rewrite_module libexec/apache2/mod_rewrite.so
#LoadModule perl_module libexec/apache2/mod_perl.so
#LoadModule php5_module libexec/apache2/libphp5.so
#LoadModule hfs_apple_module libexec/apache2/mod_hfs_apple.so

<IfModule !mpm_netware_module>

```

The line that reads #LoadModule php5\_module libexec/apache2/libphp5.so is the line you're interested in. Once you're there, place the cursor in front of the # at the beginning of the line and delete it (with the Delete key). That's it. Now hit Ctrl+X to exit. Upon exiting you'll be asked if you want to save the buffer (geek speak for "save the file"). Press Y for yes, then Enter to accept httpd.conf as the name you want to save it as. Finished!

**NOTE** nano is one of the command-line options for text editors available to Mac OS X users. You could also choose to use vi (or vim) or emacs, both of which are significantly more powerful than nano, but both also present a much steeper learning curve, one that isn't appropriate for this discussion. If you already know and wish to use one of these other text editors, it'll work just fine.



Now, to make sure everything works right, start Apache (or restart it) with the following command:

```
$ sudo apachectl graceful
```

**NOTE** Prior to Mountain Lion you could control Apache by selecting the Web Sharing option in the Sharing System Preference pane. This, to much criticism, was removed for Mountain Lion. Apple feels that if you really must run a web server, you'd be better served by loading OS X Server (\$19.99) from the App Store.

If you inadvertently created any errors in your `httpd.conf` file, you may receive an error here. If so, compare your `httpd.conf` file to your `httpd.conf.orig` backup and see if there are any changes other than removing the `#` from the PHP `LoadModule` line.

If you see nothing, you're probably in good shape. Try opening <http://localhost> in a web browser. If you get a web page that by default says "It Works!" you're in good shape; Apache is running.

#### **SERVING WEB FILES FROM YOUR OWN SITES DIRECTORY**

There's one more configuration step for files so you can easily create and serve web pages from a Sites folder in your Home folder. The first thing is to go to your Home folder in the Finder (once you're in the Finder, the `Command+Shift+H` keyboard shortcut will take you directly to your Home folder) and create a new folder called Sites. This is where you'll create your web files.

**NOTE** The editing of the `httpd-userdir.conf` file isn't necessary on OS X prior to Mountain Lion.

Now upon restarting Apache (with the `apachectl graceful` command), Apache will immediately recognize your folder, but if you try to access it through a web browser, you'll get an error. The reason for this is Apache has very restrictive default directory settings as a security precaution. To override this for user directories you need to edit the `httpd-userdir.conf` file. To open the file for editing, use this command:

```
$ sudo nano /etc/apache2/extra/httpd-userdir.conf
```

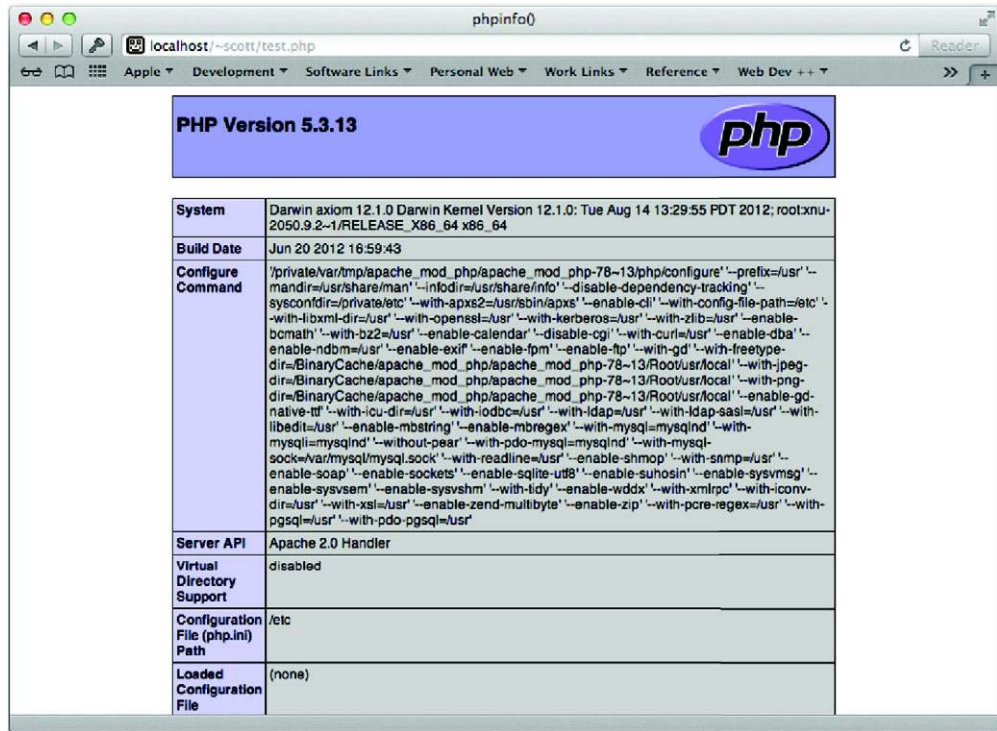
You may or may not be prompted for your password depending on when you last used `sudo`.

Once the `httpd.userdir.conf` file is open, scroll to the bottom and add the following:

```
<Directory "/Users/*/Sites/">
    Options Indexes FollowSymLinks MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Exit nano as you did before saving the revised `httpd.userdir.conf` file.

In short, this bit of code tells Apache that it has permission to look and serve content from any user's Sites folder.



**Figure C.18** If the PHP info page shows up, Apache is configured and running properly.

To test everything and make sure it all works, type the following at the terminal prompt:

```
$ echo "<? phpinfo() ?>" > ~/Sites/test.php
$ sudo apachectl graceful
```

Then point your web browser to <http://localhost/~user/test.php> (where *user* is replaced by your username). The resulting web page should look like figure C.18.

### C.3.2 Installing MySQL on Mac OS X

The easiest way to get MySQL up and running on your Mac is as follows:

- 1 Go to the MySQL website and download the latest version of MySQL community edition (<http://www.mysql.com/downloads/mysql>). If you're running Mountain Lion (which is a 64-bit OS), then the appropriate version to download is the X86, 64-bit version of MySQL in DMG format. Once the disk image is downloaded, open it and right-click the MySQL installer package and select Open. This will install MySQL into your `/usr/local/` directory. For convenience, right-click the `MySQL.prefPane` item on the disk image, and select Open. This will install a preference pane, allowing you to control MySQL from the System Preferences (see figure C.19).
- 2 Start MySQL (click the Start MySQL Server button in the Control Panel).



**Figure C.19** The MySQL preference pane will allow you to start and stop MySQL as needed.

- 3 Set a root password for MySQL by issuing the following command at the terminal prompt:

```
$ /usr/local/mysql/bin/mysqladmin -u root password "newpassword"
```

This will set the root password for MySQL to *newpassword* or whatever you put in the quotes (remember it!).

That completes the basic configuration of MySQL (easy!). Now you can invoke the MySQL client from the command prompt using

```
$ /usr/local/mysql/bin/mysql -u root -p
```

and entering your password when prompted.

### C.3.3 *Getting MySQL and PHP to play nice together*

There's one frustrating issue with getting PHP and MySQL to play nice together in Mac OS X: the location of `mysql.sock`. `mysql.sock` is a Unix socket file that allows bidirectional communication between MySQL and any other local application. In the case of our sample application, we want PHP and MySQL to talk to each other, but if you look at the PHP info (using your `test.php` web page from before), you'll see that PHP is looking for `mysql.sock` in `/var/mysql`, whereas the actual `mysql.sock` file is by default in `/tmp/`. What to do?

There are four ways to fix this. Read through the options and decide which is the sanest approach for you.

- Create the `/var/mysql` directory (`sudo mkdir /var/mysql`) and create a symbolic link from `/tmp/mysql.sock` to `/var/mysql/mysql.sock` (with: `sudo ln -s /tmp/mysql.sock /var/mysql/mysql.sock`). This method is easiest but a bit of a hack.

- Edit the `/etc/php.ini` file (it may not exist, in which case just `sudo cp /etc/php.ini.default /etc/php.ini`) so that `pdo_mysql.default_socket=/tmp/mysql.sock` (by default line 1065), `mysql.default_socket = /tmp/mysql.sock` (by default line 1219), and `mysqli.default_socket = /tmp/mysql.sock` (by default line 1278). This method is the easiest *real* way.
- Edit (or create) `/etc/my.cnf`, adding the following lines: `[mysqld] socket=/var/mysql/mysql.sock` `[client] socket=/var/mysql/mysql.sock`. This will tell MySQL to create its socket where Mac OS X's default PHP is looking for it. Next, you also need to create the `/var/mysql` directory *and* `sudo chown mysql /var/mysql` it, or MySQL won't start because it won't be able to create the socket. (Various sample `my.cnf` files can be found in `/usr/local/mysql/support-files`.) This method isn't too bad, but it could cause issues with other MySQL clients that look for the socket in `/tmp/mysql.sock`.
- Recompile `php` for your version of MySQL. This method, although a pain, isn't a terrible idea; it's actually the best fix although clearly neither fast nor easy.

Pick the way that works for you and do it. When you've finished, your computer will be ready to serve up MySQL-driven, PHP-based web apps—including the sample app in chapter 4.

# appendix D

## Computer networking primer

---

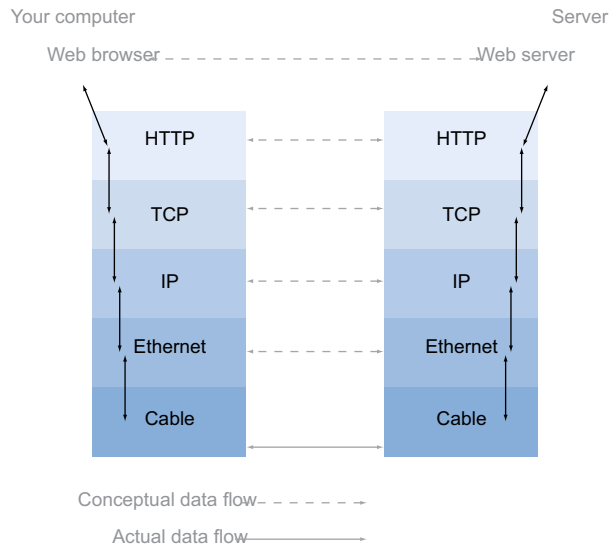
The client-server model is the foundation of the web: Your browser is the client, servers sit out in the internet cloud, and computer networking is how they talk to each other. JavaScript can only do so much by itself; most web applications are still built around the communication back to the web server. The fundamentals of computer networking—and terminology like headers, latency, throughput, and polling—are covered in most undergraduate computer science programs, but because web development attracts people from a broad range of backgrounds, this appendix assumes you’ve not been through a program like that. Here, we’ll introduce you to the following concepts:

- The basics of computer networking
- The overhead of headers
- Two important network performance metrics: latency and throughput
- Polling versus event-driven communications
- Server-side choices for event-driven web applications
- The WebSocket protocol

Along the way, you’ll also briefly review the hacks used in HTML4 to avoid the particular performance trade-offs inherent in the fundamental web protocol, HTTP. For starters, if you’re not sure what *real-time web development* even means, this appendix will provide some context.

### **D.1 The basics of computer networking**

Computer networks have both hardware and software components. Physically, they’re wires, fiber optics, or radio waves, but in software they’re defined by what’s



**Figure D.1 A network stack. Conceptually, each layer communicates directly with its counterpart on another computer. In reality, the data flow is down the stack, across the physical wires, and back up the other stack.**

called a *protocol*. The physical wires transmit pure bits of data, zeros and ones; it's the protocols that give those bits wider meaning.

To keep life simple, the software protocols are divided into layers. At the “bottom” of the stack are things like Ethernet, which is a protocol for pushing bits along wires by dividing them into packets. Above that sit protocols like the Internet Protocol (IP), which can deal with routing messages across several Ethernet connections. On top of this are protocols such as the Transmission Control Protocol (TCP), which deals with keeping track of which messages have been sent, which have been received, when to consider a message lost and repeat it, and what order they should all be in when they arrive. It's only once you get above TCP that you hit protocols like HTTP, which was designed specifically for passing web pages around.

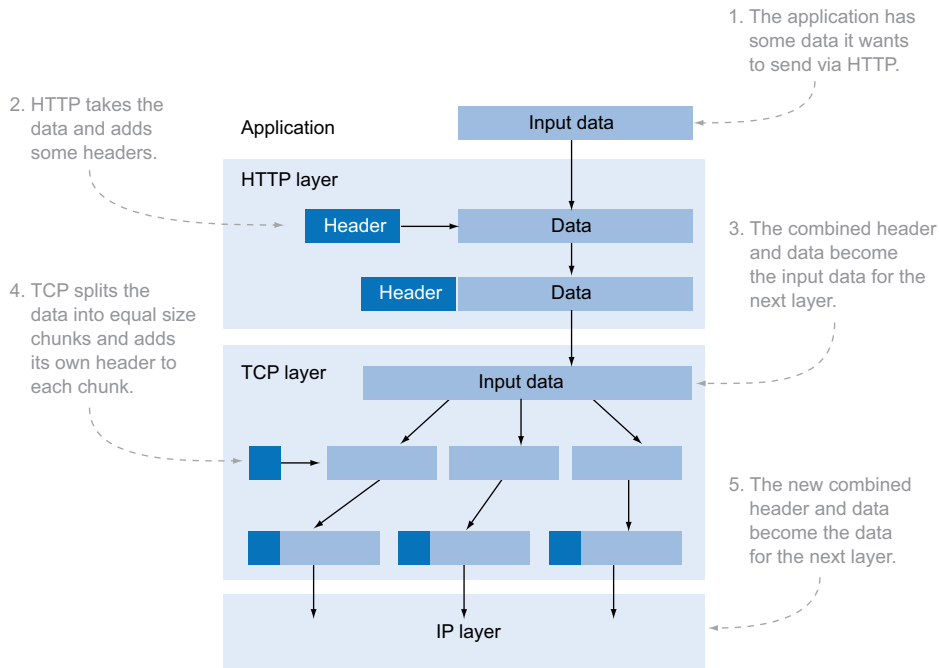
When writing a web server it's not necessary to consider how to communicate with different types of network hardware. It doesn't need different methods for sending messages across Ethernet or Wi-Fi. All it needs to know is how to describe HTTP requests to the TCP layer of the local network stack.

Figure D.1 shows this arrangement in pictorial form.

This arrangement allows communication to be conceptually simple. Applications that want to talk HTTP only need to know about HTTP and not all the other layers. But this simplicity doesn't come without cost. Each layer needs to add some information to what's being transmitted—this information generally can be referred to as *headers*, and you'll learn more about them in the next section.

## D.2 The overhead of headers

Figure D.2 focuses on exactly what's going on at the interchange between the layers, when data needs to be passed from an application over HTTP and down the network



**Figure D.2** Each layer adds header information and passes the data down the stack.

stack. At each stage, a small amount of information is added to allow the receiving layer to understand what the data is and what it's for.

The HTTP, TCP, and later IP layers each add a different type of header information. The TCP and IP header information is binary data. In binary data each of the headers can be represented by the minimum number of bits—if there are only four possible values, then only 2 bits need to be used. HTTP headers are plain text, which makes them easy to read but more verbose. The smallest possible theoretical header is a single-character label, a colon, and a single-character value. In ASCII encoding this adds up to 24 bits. Most labels and values are made up of several letters, and each HTTP request has several headers attached, with the result that most HTTP requests attach between 0.7 and 2 kilobytes of headers. This is one of the disadvantages of HTTP for data communication. If a single chat message needs to be sent, and the message is only 20 or 30 bytes, it needs to be sent with all this extra data.

In network performance terms we talk about *throughput* (or bandwidth): the amount of data that the server can send per second. If the server is limited to a throughput of 10 kilobytes per second, then it can deliver around 10 HTTP responses per second. If it only had to send the chat data, it would be able to send about 330 chat messages. From a slightly different point of view, an application based on thousands of users receiving small, real-time updates will need 33 times as many servers if you send that data over HTTP than if you're just sending the chat data.

Throughput is only one measure of network performance. In the next section you'll consider the other key factor, latency.

### **D.3 Network performance metrics: latency and throughput**

Throughput, the amount of raw data that can be transferred in a given time period, is only one aspect of networking performance. The other key factor is *latency*: the time it takes for a single bit of data to travel between two computers. Latency is important when you expect to have a lot of requests, and those requests depend on one or more of the previous requests completing.

In the previous section you learned that all the extra headers used by HTTP impact the throughput. You have every right to wonder, then, why bother with them. One reason is to improve latency. All of those headers include information about caching. This allows a browser to only download a resource, such as an image or a style sheet, a single time and then reuse the cached version for every other page that uses it. For any users who visit more than one page on your site, this means fewer network requests and therefore lower latency.

For transferring small and largely independent portions of data, all of these extra headers are a waste. The data is unique; otherwise, there's no point sending it, which means you've nothing to gain from caching.

That's not the only problem with using HTTP for data transfer. What if the client only wants to check to see if there's new data available, a process known as *polling*? Each poll will come with all the baggage of those HTTP headers. Polling can be inefficient to start with, which makes it a poor choice for real-time applications. The next section will examine this issue in more detail.

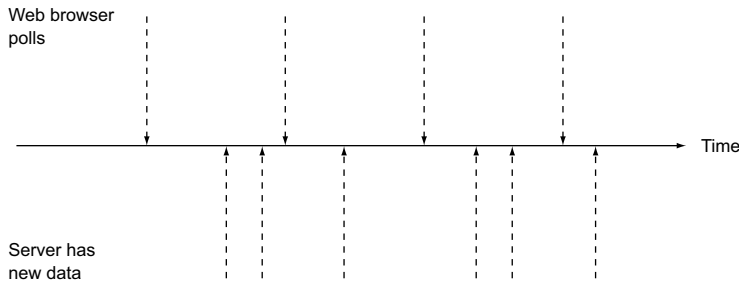
### **D.4 Polling vs. event-driven**

The phrase “real-time web” has become fashionable in recent years. Although it's based on a number of trends, the real-time web embodies a shift from the traditional client polling approach in web applications to a more *event-driven* approach. Instead of clients deciding when to ask the server if there's new information, the server sends new information to the client when it's ready.

Event-driven approaches are far more efficient than polling. This section will demonstrate that point with a series of timeline diagrams. Figure D.3 illustrates an optimum case for polling.

Even with the optimum polling solution you'll still have polls when there's no data, and for other polls data will be available for nearly the full length of time between polls. And the optimum polling solution is hard to achieve. The average chat room will have busy periods and quiet periods, and when those occur depends on the confluence of schedules of people living thousands of miles apart. It's more likely the application will spend more time in the degenerate cases (see figure D.4).

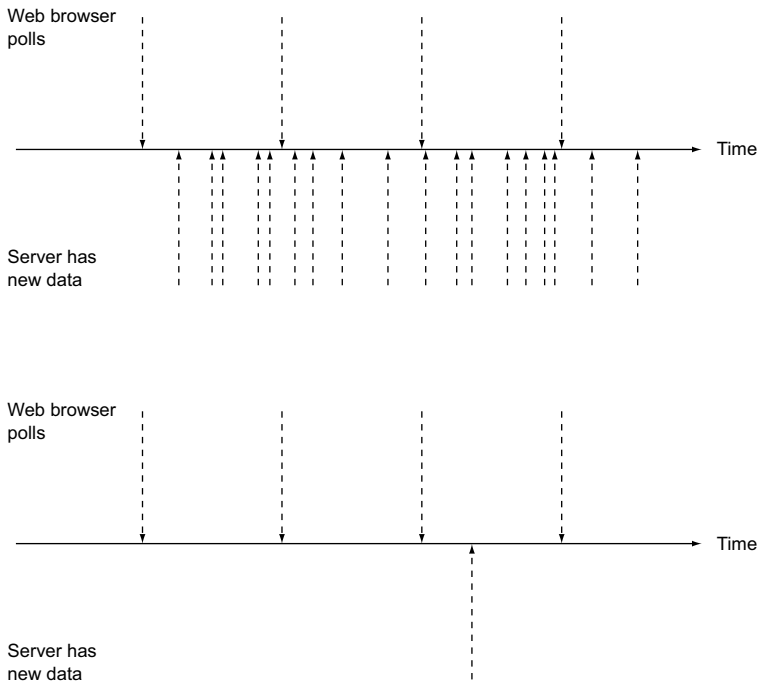




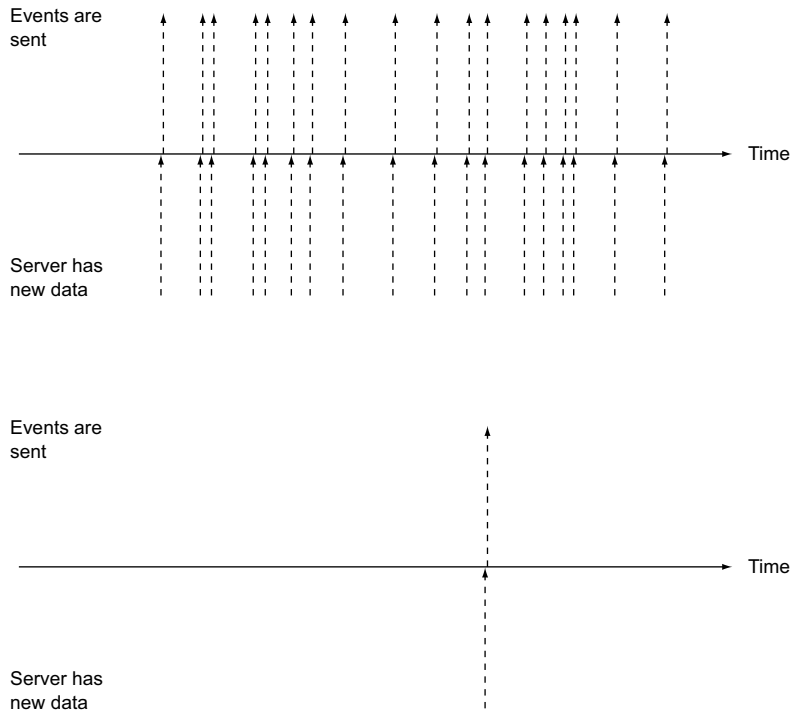
**Figure D.3** The optimum case for polling: new data is available regularly, and the frequency of the new data being available is similar to the number of polls.

The solution is to switch from polling to event-driven communication, as illustrated in figure D.5. Then the server, which knows when the information is available, is in charge of when information is delivered.

Event-driven communication is clearly more efficient because it exactly matches the frequency of communication with the frequency of the availability of new data. With no built-in support for event-driven messaging, web developers who wanted to avoid the use of plug-ins have resorted to two HTML/JavaScript hacks to simulate it: long polling and the forever frame.



**Figure D.4** The worst cases for polling: top—when new data is available far more frequently than it's polled for; bottom—when polling happens far more frequently than there's data available.



**Figure D.5** Having event-driven communication means data is sent exactly as often and exactly when it becomes available. Data is received without any wasted requests or delays.

*Long polling* allows for an increased chance of instantaneous updates by being purposefully slow in responding to a request. Instead of responding to a request immediately, the server holds the connection open and waits until there's new data. As soon as the browser receives the new data, another long poll is initiated.

The *forever frame* is a way of loading a web page slowly. The web page is loaded into a hidden iframe element. Instead of delivering all the content as quickly as possible, the server sends a chunk at a time, as updates become available. In the main page, the iframe is repeatedly scanned for new content.

Long polling approximates event-driven communication, but each request still requires a full set of HTTP headers. The forever frame approach requires the headers to be sent only once, but it still requires a lot of messing around in client code to check the contents of the frame to see if they've been updated.

Server-sent events (SSE) work along the same lines as forever frames, except the browser has a convenient API that's similar to the cross-document and channel-messaging APIs you've already seen.

## **D.5 Server-side choices for event-driven web applications**

The two new event-driven, client-server APIs in HTML5 are SSEs and WebSocket. Event-driven, client-server approaches are ideal for applications that need to send small amounts of data quickly to many clients; for example, stock-trading applications, where a few milliseconds' delay in updating can have measurable financial impact, or network gaming where delays (or lag) can make the game unplayable.

On a traditional web server, each connection is allocated a dedicated thread or process (a flow of execution within a program), which suits the model where each connection is data-intensive but short lived, such as when a web page and its linked resources are being downloaded. Event-driven communication expects the connections to be long lived but with relatively little activity. When each thread is assigned a connection, the maximum limit is soon reached and the server becomes unable to respond to new requests.

This can be a problem for traditional web servers like Apache, which allocate a process or thread per connection. The number of processes or threads that can be created is limited, even if, as is usually the case, all of those processes or threads spend most of their time doing nothing. Servers such as Lighttpd and nginx share the processes between the connections to allow them to handle a far larger number; these servers have risen in popularity along with event-driven, real-time web applications.

## **D.6 Understanding the WebSocket protocol**

The WebSocket protocol allows bare-bones networking between clients and servers with little overhead—certainly far less overhead than the previously more common approach of attempting to tunnel other protocols through HTTP. With WebSockets it's possible to package your data using the appropriate protocol, the eXtensible Messaging and Presence Protocol (XMPP) for chat, for example, but benefit from the strengths of HTTP, which, like MasterCard, is accepted nearly everywhere.

### **D.6.1 WebSocket protocol vs. WebSocket API**

The specifications for WebSockets are split into two parts. The WebSocket protocol describes what browser vendors and servers have to implement behind the scenes; it's the protocol used at the network layer to establish and maintain socket connections and pass data through them. The WebSocket API describes the interface that needs to be available in the DOM so that WebSockets can be used from JavaScript.

The Internet Engineering Task Force (IETF) maintains the specification for the WebSocket protocol. This is the same organization that manages the specifications for HTTP, TCP, and IP.

WHATWG maintains the specification for the WebSocket API in concert with W3C, the same as for the HTML5 specification itself.

## D.6.2 The WebSocket protocol

Like parts of the HTML5 specification, the WebSocket protocol spent many months under heavy development, but unlike HTML5, the versions that the client and server are using need to match for everything to work.

The WebSocket protocol describes, in detail, the exact steps a client and server take to establish a WebSocket connection, exchange messages, and ultimately close the WebSocket. To make a node, or any web server, accept WebSocket connections, you need to implement the WebSocket protocol. In this section you'll get an overview of how that protocol works. The following listing is a set of HTTP headers that the browser will send to the server in order to initiate a WebSocket connection.

### Listing D.1 The WebSocket handshake

This header is a base64-encoded string (decoded, this one reads “the sample nonce”). The decoded string must be 16 bytes long. It will be transformed by the server and returned to the browser for security verification in listing D.2.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat.example.com, chatplus.example.com
Sec-WebSocket-Version: 13
```

The format is intentionally modeled after HTTP requests; to web servers, routers, proxies, and other web infrastructure this request should look like a normal HTTP request.

These headers indicate to the server that you're expecting an upgrade from HTTP to WebSocket.

The WebSocket protocol version the web browser is expecting; for hybi-17 the version is 13 because hybi-13 was the last version where a noncompatible change was made.

The list of subprotocols the browser understands; the protocol the application will be using across the Web Socket—these are application-specific and the whole header is optional.

This tells the server where the script making the WebSocket request originated, allowing cross-domain requests to be blocked if necessary.

A typical server response is shown in the next listing.

### Listing D.2 The server response

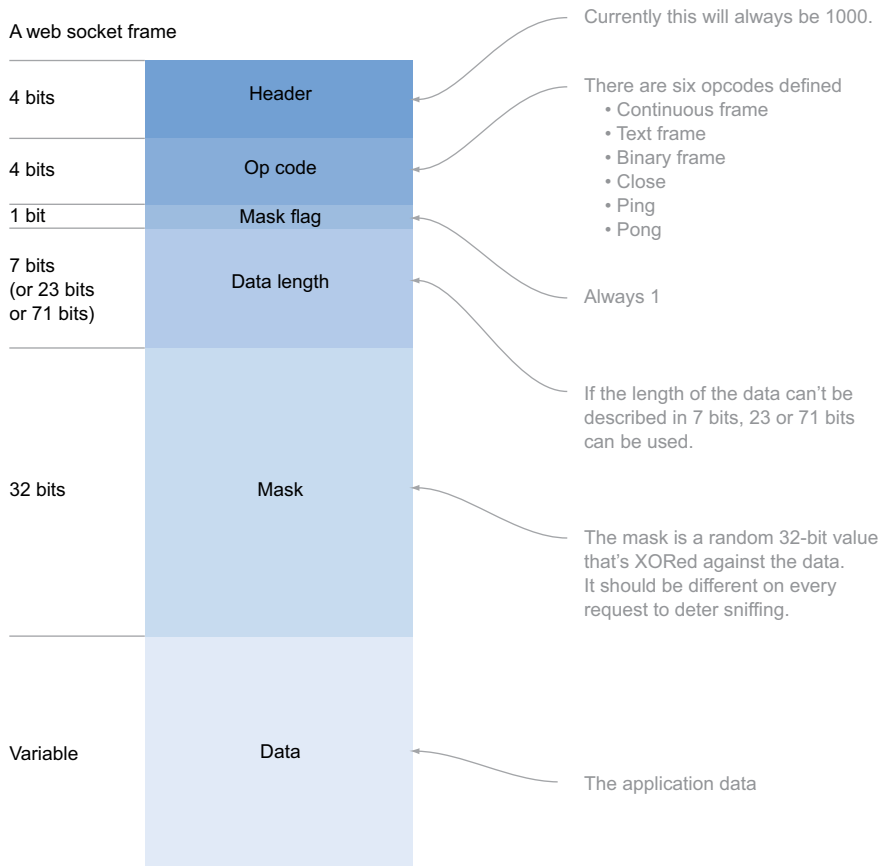
```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat.example.com
```

The web server announces that it has accepted the upgrade request.

The upgrade headers are echoed back.

Of the subprotocols listed by the client, this is the one the server understands.

Response to the Sec-WebSocket-Key header in listing D.1. The string 258EAFAS-E914-47DA-95CA-C5AB0DC85B11 is appended to the value; it's hashed with SHA-1 and then base64-encoded again and placed in this field.



**Figure D.6** Diagram of a WebSocket frame—the non-data parts take up only 48 bits, equivalent to six characters.






Once the handshake is complete, data exchange can begin. The messages in WebSockets are sent in what are referred to as frames. Figure D.6 shows the structure of a frame. Frames are a lightweight container for the data with minimal binary headers. The overhead per message is only 6 bytes.

### D.6.3 **WebSocket browser support**

Although the protocol is well-defined, let's review further complications. The WebSocket protocol was only finalized in spring 2012. Before that, seven different versions of it had seen some browser support. Table D.1 shows the different versions and the browsers that support each of them.

Table D.1 lists only the versions where noncompatible changes were made. But you can see from the numbers in the table that there have been many versions of the specification, up to version 76 when the specification was still maintained by WHATWG (hixie-76), which then became the initial version of the IETF-maintained specification.

**Table D.1** WebSocket protocol versions and browser support

WebSocket protocol version					
hixie-75	4				5.0.0
hixie-76 / hybi-00	6	4 (disabled)		11 (disabled)	5.0.1 (and iOS5)
hybi-06			8/9 (add-on)		
hybi-07		6			
hybi-09			8/9 (add-on)		
hybi-10	14	7	10 (DP1)		
hybi-17 / RFC 6455	16	11	10	12.50	6.0, iOS6

It has since seen 17 further revisions before finally being released as RFC 6455. The client passes the version it understands in the initial request (listing D.1). On the server side you can decide whether or not to support the version the client understands. Obviously, the more versions you choose to support, the more work you have to do on the server.

# *appendix E*

## *Setting up Node.js*

---

This appendix is provided for readers who need to set up Node.js for the chapter 4 application. You might be wondering why we chose Node.js. There are several alternative web servers that are much better suited to WebSockets than the traditional choices of Apache or IIS (IIS8 will have built-in WebSocket support). These servers share connections between threads, taking advantage of the mostly idle nature of event-driven connections. In chapter 4, you'll be using Node.js for two reasons:

- It uses JavaScript, which you're already familiar with.
- It has an easy-to-use library implementing the WebSocket protocol.

This appendix will walk you through installing and setting up Node.js for the chapter 4 application. You'll also learn how to build basic web applications with Node.js and how to use the Node Package Manager (NPM). NPM lets you easily install modules to extend the functionality of Node. You'll also create a simple application to confirm that the modules are installed correctly.

### **E.1 Setting up Node.js to serve web content**

Node.js is an event-driven web server based on the V8 JavaScript engine, which is part of the Google Chrome browser. The basic process for installing Node is to download the source code and compile it. For Linux and Unix users, this isn't an unfamiliar approach, but this may come as a bit of a shock to Windows users. For Windows, a prebuilt binary is available from the installation page: <https://github.com/joyent/node/wiki/Installation>.

Even if you're using the prebuilt binaries, the prerequisites mentioned on the installation page are still required because they're used in the installation of modules (which you'll look at in E.2). Unfortunately, the installation page doesn't do a very good job of explicitly stating the requirements for each platform; table E.1 summarizes the prerequisites for all the major platforms.

**Table E.1** Node.js prerequisites by platform

Platform	Prerequisites
Linux	GCC 4.x.x; GNU make 3.81 or newer; Python 2.6 or 2.7
Unix/BSD	GCC 4.x.x; GNU make 3.81 or newer; Python 2.6 or 2.7; libexecinfo
Mac	Xcode 4.5; GNU make 3.81 or newer; Python 2.6 or 2.7
Windows	Visual Studio 2010 or Visual C++ 2010 Express; Python 2.6 or 2.7

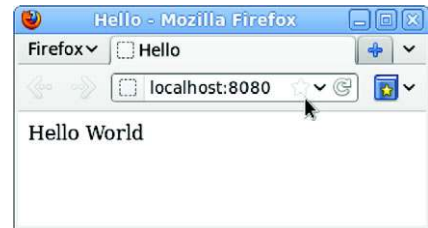
Once you've installed your prerequisites correctly you can get started. This appendix will walk you through a few simple example Node applications, which will confirm that your installation is correct and let you see how common web application scenarios are handled in Node.

### E.1.1 Create a Node Hello World application

In this section you'll build, in two steps, the traditional Hello World application shown in figure E.1. You'll generate a page entirely dynamically using JavaScript.

#### STEP 1: CREATE A NODE APPLICATION

The first listing is a simple Hello World application for Node, as shown in figure E.1. Create a file called `app.js` in your working directory and place this code into it.

**Figure E.1** Node says "Hello World."**Listing E.1** Node Hello World

```

var http = require('http');
http.createServer(function(request, response) {
  response.writeHead(200);
  response.write("<!DOCTYPE html>");
  response.write("<html>");
  response.write("<head>");
  response.write("<title>Hello</title>");
  response.write("</head>");
  response.write("<body>");
  response.write("Hello World");
  response.write("</body>");
  response.write("</html>");
  response.end();
}).listen(8080);

```

Most functionality in Node is implemented through a system of modules; here the built-in `http` module is loaded.

The `http` module has a `createServer` method, which is passed a handler function that will be called when requests are made.

The response itself is a simple web page; each line is explicitly written into the response.

After the server is created, it's set to listen on port 8080; any requests to <http://localhost:8080> will now be passed to the handler function.

The `end()` method indicates that the content is complete.

#### STEP 2: RUN A NODE APPLICATION

After you've created your `app.js` file you should be able to start Node. Issue a command like the following from your shell or command prompt, making sure you're in



your working directory (you may need to log off and back on for the Node folder to be added to your path):

```
node app.js
```

This command will start Node running in the current directory using the file `app.js` to determine behavior. Start the Node server with the command shown previously. Once the Node server is running, point your browser at <http://localhost:8080/> and re-create figure E.1.

## E.1.2 **Serving static files with Node**

Node is a bare-bones web server, which means many things you might take for granted with more traditional web servers won't happen in Node, unless you write code to make them happen. For instance, Node won't transparently transfer any static files that happen to be sitting in the execution directory. If you want a file called `index.html` to be sent to the browser in response to a request, it's up to you to detect the requested URL, locate the file, and then send it in response.

In this section you'll create, in four steps, a static file and serve it with Node:

- Step 1: Create a static file.
- Step 2: Load a file from a disk.
- Step 3: Send the file to the browser.
- Step 4: Run the application.

The end result will look identical to what you achieved in the previous section, but the architecture will be improved because your display rendering is separated from your application logic.

### **STEP 1: CREATE A STATIC FILE**

The next listing is a simple `index.html` file—place it in a new working directory.

#### **Listing E.2 A static `index.html` file**

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello</title>
</head>
<body>
Hello world
</body>
</html>
```

### **STEP 2: LOAD A FILE FROM A DISK**

Now that you have a static HTML file, you need to load it from a disk. Node has a built-in module for reading files from a disk called `fs`. You can use the `fs.readFile()` method to load a file. Create a new `app.js` file in your working directory and add the code from the following listing.

Listing E.3 An app.js file that reads a file from a disk

```

var http = require('http');
var fs = require('fs');

http.createServer(function(request, response) {
  fs.readFile('./index.html', function(error, content) {
    //Code to handle the file goes here
  });
}).listen(8080);

```

← To read from the filesystem, the fs module is needed.

← Two parameters are required for readFile(), a path to a file and a function that will be called after the file has been read.

← In the next step you'll fill in this code.

**STEP 3: SEND THE FILE TO THE BROWSER**

After the `readFile()` function has completed, your callback function will execute; you need to check that the file was read successfully and send it to the browser. You can use the same `http` module methods from section E.1.1 to do this. Replace the comment in listing E.3 with the code in the following listing.

Listing E.4 Sending the file to the browser in app.js

```

if (error) {
  response.writeHead(500);
  response.end();
} else {
  response.writeHead(200,
    { 'Content-Type': 'text/html' });
  response.end(content, 'utf-8');
}

```

← If there's an error reading the file, it will be handled gracefully here...

← ...otherwise, send the file to the user.

**STEP 4: RUN THE APPLICATION**

Just as in the last example, start your application from the command line:

```
node app.js
```

Point your browser at <http://localhost:8080/> and check that you can see the Hello World page.

Serving static files isn't interesting in and of itself, but serving purely dynamic files, as in section E.1.1, isn't practical either; in a real application you don't want your web designers attempting to edit HTML and CSS inside your JavaScript application logic. You need to be able to mix static content with the results of the application logic, and you'll look at that in the next section.

**E.1.3 Serving mixed static and dynamic content with Node**

In this section you're going to create an HTML template file with placeholders that will be replaced with dynamic values when the page is requested.

**STEP 1: CREATE A STATIC TEMPLATE WITH PLACEHOLDERS**

The `index.html` template is shown in the next listing. It's identical to the previous `index.html` file, apart from the added placeholder for the dynamic variable. Create a new working directory and place this `index.html` file into it.

## Listing E.5 A simple template index.html

```

<!DOCTYPE html>
<html>
<head>
  <title>Hello</title>
</head>
<body>
Hello world {0}
</body>
</html>

```

**{0}** is a placeholder for your dynamic content.

**STEP 2: MIX DYNAMIC CONTENT INTO YOUR TEMPLATE**

The following listing inserts dynamic values into a static template file using the standard JavaScript `String.Replace` function. Create an `app.js` file in your working directory and add this code to it.

## Listing E.6 Mixing static and dynamic content in app.js

```

var http = require("http");
var fs = require("fs");
var inc = 0;

http.createServer(function(request, response) {
  fs.readFile('./index.html', function(error, content) {
    if (error) {
      response.writeHead(500);
      response.end();
    } else {
      response.writeHead(200, { 'Content-Type': 'text/html' });
      response.end(
        content.toString().replace(/\{0\}/g, ++inc),
        'utf-8'
      );
    }
  });
}).listen(8080);

```

The file is loaded as before.

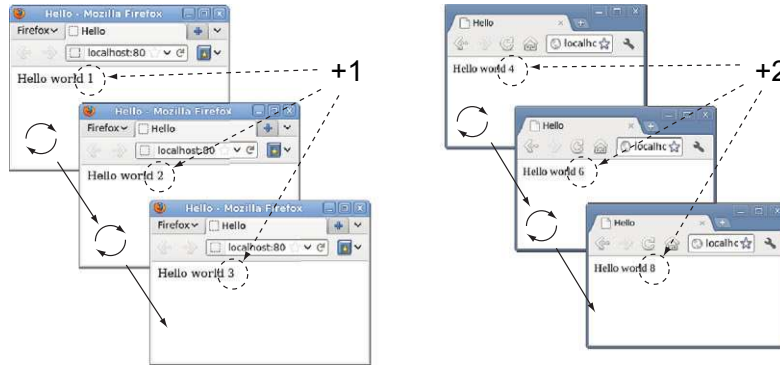
A simple dynamic page is implemented by replacing all instances of **{0}** with the value of a pre-incremented `inc`.

**STEP 3: TEST IN THE BROWSER**

Start Node with your new `app.js` file as you did in the previous examples, and load the page in the browser. Each time the page gets refreshed in the browser, the variable should get incremented once. You can expect the number on the page to increase by one with every page load. Let's try it in real time in a few browsers. Figure E.2 shows the results.

You may be scratching your head at this result—we certainly did. Chrome's Network tab in its developer tools shows only a single request, so why does the variable get incremented twice? The answer is that Chrome makes an additional request that it doesn't tell you about for `favicon.ico`. In case you're not familiar with it, `favicon.ico` is the standard name for the little icon that appears alongside the URL in the address bar. Because your Node server is configured to respond to every request with

Each time the page is loaded in Firefox, the variable increments once.



But each time the page is reloaded in Chrome, the variable increments by two. What's going on?

**Figure E.2** The results of reloading the simple dynamic page in Firefox and Chrome

the same HTML file, it sends that file in response to the request for `favicon.ico`, too. This results in the variable being incremented twice.

To stop this from happening, you need a way to route requests for different URLs to different server responses. This is called *routing* and will be the subject of the next section.

### E.1.4 Routing: serving different files for different URLs

Routing is the matching up of the URL requested by the browser with the appropriate response from the server. The following listing demonstrates a simple approach.

**Listing E.7** Simple routing `app.js` example

```
var http = require("http");
var fs = require("fs");
var inc = 0;

http.createServer(
function(request, response) {
  if (request.url === '/index.html' || request.url === '/') {
    fs.readFile('./index.html', function(error, content) {
      if (error) {
        response.writeHead(500);
        response.end();
      } else {
        response.writeHead(200, { 'Content-Type': 'text/html' });
        response.end(
```

Only respond with the file to requests for `index.html` or the default document.

```

        content.toString().replace(/\{0\}/g, ++inc),
        'utf-8'
    );
    }
  });
} else {
  response.writeHead(404);
  console.log(request.url + ' not found!');
}
}).listen(8080);

```

For any other request, return a 404 not found error.

The example increments the variable only once for every page reload in Chrome, because the `index.html` is sent only to explicit requests for it or for the root document.

In this section, we've provided a low-level understanding of how Node handles web applications. In real life, you don't want to spend hours figuring out how to do things such as sending each individual file to your users, or slicing up your data to fit into WebSocket frames, or keeping up with all the changes in the specification. To avoid repeatedly doing the boring stuff in Node.js, you need more modules. But the modules you'll need aren't included as standard with Node like `http` and `fs` that you've already used. In the next section, you'll learn how to go above and beyond the standard set of modules by downloading third-party modules using the Node Package Manager (NPM).

## E.2 Easy web apps with Node modules

In the previous section, you explored several functions that need to be performed in Node in order to create real web applications: placing dynamic content inside static files (or templating) and mapping requests at different URLs to appropriate handlers (or routing). But you also need to handle WebSocket requests. That was the point of installing Node in the first place. If you looked at the explanation of the WebSocket protocol in appendix D, you know that handling WebSockets involves a lot of slicing and dicing of binary data. This is hardly the use case for which JavaScript was designed. You don't want to spend all your time dealing with low-level stuff like that when you could be writing applications. All of this can be more easily accomplished in Node by taking advantage of third-party modules. In this section we'll set you up with the following modules:

- *Director*—for routing
- *Mustache*—for templating
- *WebSocket-Node*—for the WebSocket protocol

You can easily manage modules with the NPM script. Because modules have become fundamental to using Node, NPM, once a handy add-on, now comes as part of the main Node.js distribution.

It's easy to install the modules; Node looks for them in the `node_modules` directory of the current directory. This will be the same place where your `app.js` file is

Name	Size	Type
test-bed	2 items	folder
node_modules	3 items	folder
director	9 items	folder
websocket	12 items	folder
mustache	9 items	folder
app.js	1.0 kB	JavaScript program

**Figure E.3** Application file layout with modules installed

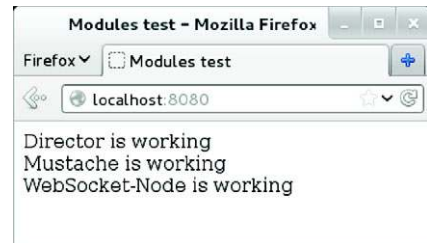
located, so make sure you're in that directory before installing modules. Then, run these commands:

```
npm install director
npm install mustache
npm install websocket
```

Modules are now installed local to your application, and you have a file structure like that shown in figure E.3.

Now you're going to create an application that's going to try loading all three of these modules and show a message if it is successful. This will tell you the modules have been installed correctly. The result is shown in figure E.4.

The following listing is the `app.js` file in figure E.3. Create a new working directory and add `app.js` to it. When you run it with Node, it'll attempt to load all the modules you've installed and create a simple page.



**Figure E.4** If you see this page, everything is working correctly.

#### Listing E.8 An `app.js` using Director, Mustache, and WebSocket-Node

```
var http = require("http");
var director = require("director");
var mustache = require("mustache");
var WebSocketServer = require('websocket').server;

var template = `<!DOCTYPE html>\n`
+ `<html>\n`
+ `<head>\n`
+ `<title>Modules test</title>\n`
+ `</head>\n`
+ `<body>\n`
+ `Director is {{director_status}}<br>\n`
+ `Mustache is {{mustache_status}}<br>\n`
+ `WebSocket-Node is {{socket_status}}\n`
+ `</body>\n`
+ `</html>`;
```

← When installed with NPM, add-on modules are referenced in the same way as built-in modules.

← Instead of emitting the markup directly, we'll use this variable to store a template for mustache. In later examples you'll load the template from disk.

```
var dict = {
  'director_status': director.http.Router ? 'working':'broken',
  'mustache_status': mustache.to_html ? 'working':'broken',
  'socket_status' : typeof WebSocketServer !== 'undefined'
                    ? 'working':'broken'
};

var html = mustache.to_html(template,dict);

http.createServer(
  function(request, response) {
    response.writeHead(200);
    response.write(html);
    response.end();
  }
).listen(8080);
```

←

**This object restores the results of a few simple tests, which confirm all the modules loaded correctly.**

←

**Mustache is the only module this example uses.**

If everything has gone according to plan, you should see a page similar to that shown in figure E.4 when you fire up Node and browse to port 8080.

Now that you know everything is installed correctly you're ready to build your first WebSocket application. If you came to this appendix from the build prerequisites (section 4.2) in chapter 4, you can head back to that chapter and continue with the build.

# appendix F

## Channel messaging

---

Channel messaging is similar to cross-document messaging (see chapter 4) except instead of one message channel per window, it allows multiple channels to be created. This is useful if you want to build the page out of a number of loosely coupled, event-driven, independent components. Rather than adapt them all to share a single cross-document messaging interface and ensure they don't clash with each other's internal API for message formats, each component can have its own set of private channels.



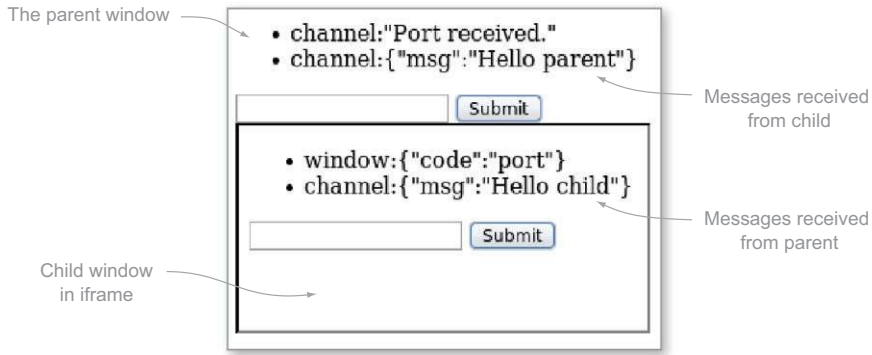
**Channel messaging** 4 ~ 10 11 5

In the next few pages you'll build a simple test bed by setting up a page that loads a document from a different domain. You can easily fake running multiple domains from your own computer, and in this section you'll walk through setting up two pages on your computer that run from different domains. One page will load the other in an `iframe`, and you'll use channel messaging to communicate between the two. Figure F.1 shows the test bed after channel messaging has occurred. You can use the textboxes to create the messages, and any message received will be added to the document.

You can build the example by following these five steps:

- Step 1: Install a local development web server.
- Step 2: Set up a cross-domain test environment.
- Step 3: Create the example pages.
- Step 4: Add JavaScript to the first page.
- Step 5: Add JavaScript to the second page.





**Figure F.1** A simple channel-messaging example, such as the one you'll build in chapter 4's listings 4.5 and 4.6

### STEP 1: INSTALL A LOCAL DEVELOPMENT WEB SERVER

You'll need to be able to serve web pages from your local machine. In other words, you need to have a web server. If you already have one, please skip ahead; otherwise, follow along, and we'll review some easy options:

<b>Windows</b>	Microsoft Visual Web Developer Express comes with a built-in web server. You can easily create a web application project and place the files you create in the next section within it. See the download pages at <a href="http://mng.bz/gu1b">http://mng.bz/gu1b</a> for further details.
<b>Mac</b>	Go into System Preferences > Sharing, and check the Web Sharing box. The default folder is <code>/Library/WebServer/Documents/</code> .
<b>Linux</b>	Most Linux distributions come with Python already installed, and Python includes the <code>SimpleHTTPServer</code> module. To start it, open a command prompt and set the current directory to the one containing your files; then issue the command <code>python -m SimpleHTTPServer 8000</code> to start a server listening on port 8000.

### STEP 2: SET UP A CROSS-DOMAIN TEST ENVIRONMENT

To experiment with messaging between scripts in different domains, you need to have multiple domains available. The easiest way to do this is to fake some domains by editing the hosts file on your computer. On Windows this file is usually found at `C:\Windows\System32\drivers\etc\hosts` (note the lack of a file extension; also note that this is a system file, so run your editor as administrator); on Mac OS X and Linux it'll be found at `/etc/hosts`. Opening that file in a simple text editor should reveal some lines like the following:

```
127.0.0.1 localhost
```

Add your fake domains to the end of the line starting 127.0.0.1:

```
127.0.0.1 localhost domain1.com domain2.com
```

Now you can browse to <http://domain1.com> and <http://domain2.com>, and the pages will be served from your local web server.

**STEP 3: CREATE THE EXAMPLE PAGES**

First, you'll need two pages in your working directory. Call them `example-1.html` and `example-2.html`, and add the markup shown in the following listing to the body section. This markup is even simpler than the cross-document messaging example in chapter 4 because JavaScript will add the `iframe`.

**Listing F.1** Channel messaging/`example-1.html` body content

```
<ul id="log"></ul>
<form id="msgform">
  <input type="text" id="msg">
  <input type="submit">
</form>
```

**STEP 4: ADD JAVASCRIPT TO THE FIRST PAGE**

Now you need to add code to initiate the messaging.

Channel messaging works by creating a pair of ports. A port is a generic object that allows messaging. It supports the `postMessage` method and `onmessage` event that you're familiar with from cross-document messaging. Anything sent to one port will appear as output from the other port; in HTML5 terms they're described as *entangled*. This is by analogy to *quantum entanglement*: two particles that, no matter what distance separates them, change simultaneously. One of the ports is then sent to another script context. This could be a script in another document or window or a web worker. Listing F.2 shows the details. The code from listing F.2 should go in a `<script>` block after the form in `example-1.html` (you could add it in the head element, but then you'd need to wrap it in a function and execute it on the load event). As you can see, the channel-messaging API is similar to the cross-document messaging API you looked at in chapter 4.

**Listing F.2** Channel messaging/`example-1.html` JavaScript

As with cross-document messaging, the second parameter is the domain the message is getting passed to.

```
var f = document.getElementById('msgform');
var m = document.getElementById('msg');
var l = document.getElementById('log');

var channel = new MessageChannel();

var w = document.createElement('iframe');
document.body.appendChild(w);
w.setAttribute('src', 'http://domain2.com/example-2.html');
var sendPort = function() {
  w.contentWindow.postMessage({"code": "port"},
    'http://domain2.com',
    [channel.port2]);
}
w.addEventListener('load', sendPort, false)
```

For convenience, several global variables are set up; if this was more than a single-page example, it would be better to wrap this whole listing in a reusable object.

The `MessageChannel` constructor returns a pair of entangled ports.

Create an `iframe` element and load a document into it.

Here's the familiar `postMessage` function; the first argument is a string. The value used here isn't necessary, but it'll allow for easy detection in the other page.

The new parameter for channel messaging is an array of port objects; the second port from the `MessageChannel` is passed.

There's no point in sending the port if the document isn't loaded, so wait for the load event before sending the port.

```

var channel_message = function(e) {
    var li = document.createElement('li');
    li.appendChild(
        document.createTextNode(
            'channel:' + JSON.stringify(e.data)
        )
    );
    l.appendChild(li);
}
channel.port1.onmessage = channel_message;
var send_message = function(e) {
    var s = {};
    s.msg = m.value;
    channel.port1.postMessage(s);
    m.value = '';
    e.preventDefault();
}
f.addEventListener('submit', send_message, false);

```

The first port of your channel is now entangled with the port sent to the other document, so to receive the messages you need to listen to the onmessage event.

Reversing that, if you use postMessage on the first port, the message will appear on the second port in the other document.

The send\_message function is bound to the form's submit event to allow the user to send messages to the other document.

### STEP 5: ADD THE JAVASCRIPT TO THE SECOND PAGE

Now you need to set up the second page to allow it to receive the port and then send and receive messages through it. The code from the next listing should go in a <script> block after the form in example-2.html.

#### Listing F.3 Channel messaging/example-2.html JavaScript

```

var f = document.getElementById('msgform');
var m = document.getElementById('msg');
var l = document.getElementById('log');
var port;
var receive_port = function(e) {
    var d = typeof e.data === "string"
        ? JSON.parse(e.data)
        : e.data;
    if (d.code == "port") {
        port = e.ports[0];
        port.postMessage("Port received.");
        port.onmessage = function(e) {
            var d = typeof e.data === "string"
                ? JSON.parse(e.data)
                : e.data;

            var li = document.createElement('li');
            li.appendChild(
                document.createTextNode('channel:' + d)
            );
            l.appendChild(li);
        }
    }
    var send_message = function(e) {
        var s = {};
        s.msg = m.value;
        port.postMessage(s);
        m.value = '';
        e.preventDefault();
    }
}

```

Several global variables are set up, but this time there's no need to create a new MessagePort. Instead, a variable is created to store the port that will be sent to this window.

This function will be used when you're passed a message from another document.

The convention is that if there's a message code of "port," then you should grab the attached port and use it for messaging. You don't have to use "port"; anything else would work as well, but it's here that you would start to define an API for clients.

Any messages received will be logged.

Store a reference to the port in the global variable.

Let the caller know the port has been received.

Attach a handler to the port message event.

Set up the form so the user can send messages back through the port.

```
        f.addEventListener('submit', send_message, false);
    }
    var li = document.createElement('li');
    li.appendChild(document.createTextNode('window:' + d));
    l.appendChild(li);
}
window.addEventListener('message', receive_port, false);
```

The fact that a message event has been handled is logged for the benefit of your audience.

The receive\_port handler is bound to the window.

Now you're ready to try to re-create figure F.1. If you've been following along, the URL should be similar to <http://domain1.com:8000/example-1.html>. The port number (8000) might be different, depending on which local web server you used; refer to the documentation for details.

# *appendix G*

## *Tools and libraries*

---

It's important to know what tools and libraries are available for developers looking to leverage HTML5 APIs. These can save time and make your projects less buggy. In this appendix, you'll find out about several tools for mobile and HTML5 applications.

### **G.1 Tools for mobile web applications**

If you're an experienced web application developer, you'll probably be familiar with web and JavaScript frameworks that make your life easier when it comes to ensuring cross-browser compatibility with your code, or that reduce your workload by giving you access to UI widgets and components. If so, you may have been wondering if there's any such framework for mobile web applications. Fortunately, the answer is yes, and there's a decent choice of frameworks on offer, including:

- jQuery Mobile
- Sencha Touch
- Dojo Mobile
- Jo

These frameworks provide a means of building mobile web applications that leverage HTML5 to create a native app experience. They all feature a set of rich UI components that mimic native mobile UI features such as lists, navigation bars, toolbars, tab bars, form controls, carousels, and more. Each also provides a data abstraction layer that makes it easier to interact with HTML5's storage features, which are covered in chapter 5. All of these frameworks can be used in tandem with frameworks, such as PhoneGap, for deploying mobile web applications as native apps on various platforms.

## **G.2 Tools for HTML5 applications**

This section covers things that you'll find useful when developing HTML5 applications. It includes tools within browsers, development versions of browsers, and external tools and scripts to make your life easier.

### **G.2.1 Firebug, Chrome/Safari developer tools, Dragonfly, IE developer tools**

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 IE
- Mozilla Firefox
- Opera

In addition to installing the major web browsers, you should ensure that you have the available relevant tools to make your life easier. All of the major browsers now include a suite of web developer tools. 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 the following:




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

### **G.2.2 Browser development versions**

In addition to the release (or stable) versions of browsers your users have right now, you should also consider installing one or more of the development versions. Development versions of browsers are where the testing of new features happens, so they allow you to try out new standards as they're being finalized and also test your web applications in the next version of the browser.

Table G.1 lists the major browsers and where to get development versions of them.

**Table G.1 Browsers and their development versions**

Browser	Development Versions
	<ul style="list-style-type: none"> <li>■ Chrome Beta: <a href="https://www.google.com/landing/chrome/beta/">https://www.google.com/landing/chrome/beta/</a> (can't be installed side by side with stable or dev versions)</li> <li>■ Chrome Dev: <a href="http://mng.bz/XKev">http://mng.bz/XKev</a> (can't be installed side by side with stable or beta versions)</li> <li>■ Chrome Canary: <a href="https://tools.google.com/dlpage/chromesxs">https://tools.google.com/dlpage/chromesxs</a> (can be installed side by side with stable, beta, or dev versions)</li> </ul>
	<ul style="list-style-type: none"> <li>■ Firefox Beta: <a href="https://www.mozilla.org/en-US/firefox/beta/">https://www.mozilla.org/en-US/firefox/beta/</a></li> <li>■ Firefox Aurora: <a href="https://www.mozilla.org/en-US/firefox/aurora/">https://www.mozilla.org/en-US/firefox/aurora/</a></li> <li>■ Firefox Nightly: <a href="http://nightly.mozilla.org/">http://nightly.mozilla.org/</a></li> </ul>
	<p>Internet Explorer has a much slower release cycle than the other major browsers, so there isn't a regular snapshot available. Check <a href="http://ie.microsoft.com/testdrive/">http://ie.microsoft.com/testdrive/</a> for information on any beta versions or release candidates available.</p>
	<p>Opera: Beta and alpha versions are called Opera Next; get them here: <a href="http://www.opera.com/browser/next/">http://www.opera.com/browser/next/</a> Whether a particular Opera Next release is a beta or an alpha depends on how close to the next release they're getting; closer to release and they'll be betas.</p>
	<p>Safari: There are no beta releases of Safari as such, but you can download a nightly version of the WebKit rendering engine that powers Safari and use it within your existing Safari install: <a href="http://nightly.webkit.org/">http://nightly.webkit.org/</a></p>

The different browsers' development versions each use their own terminology. Table G.2 will help you to understand what to expect from each version.

**Table G.2 Development version terminology**

Term	Description
Beta/Release Candidate	Mostly stable and approaching release, updated once every week or two
Dev/Aurora/Alpha	Not guaranteed to be stable, updated once a week or more
Nightly/Canary	Cutting edge and unstable, updated every night

### G.2.3 **HTML5 Shiv**

HTML5 Shiv is a shim (a small, compatibility-focused library) for enabling support for HTML5's new elements in older versions of IE. That it's called HTML5 Shiv rather than HTML5 Shim is an accident of history, but it does serve to differentiate it from the large number of shims that have arisen around HTML5 in recent years. Download the latest version from <https://github.com/aFarkas/html5shiv> or use Modernizr (described in the next section), which includes the Shiv.

### G.2.4 **Modernizr**

One of the problems with using HTML5 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 Firefox4 but not in Firefox3.6. Safari4 didn't have support for WebSockets; these were introduced in Safari5. With the ever-expanding set of features in HTML5, and the ever-changing state of browser support among 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'd 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 previous approach. Let's take the `<canvas>` element as an example:

```
!!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'd 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 reusable, but writing functions for each and every HTML5 feature would be painstaking. Thankfully, there's 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:

```
<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'd 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>.

### G.2.5 **HTML5 Boilerplate**

If you're building an HTML5 application from scratch, there's 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:

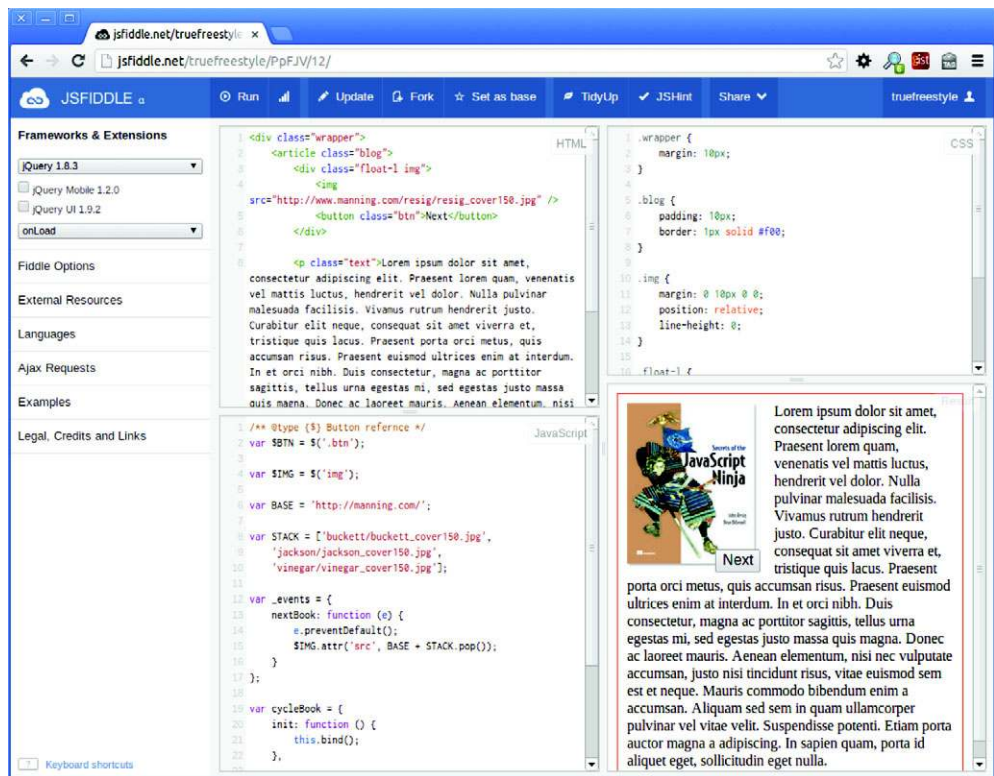
- Modernizr
- jQuery (hot-links to a Google-hosted file for performance, with a local fallback)
- Optimized code for including Google Analytics
- Conditional comments to allow for Internet Explorer-specific styling
- CSS reset, printer-friendly styles
- Google-friendly robots.txt file
- .htaccess file 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's Delete-key friendly, so if you don't want to include anything that comes as part of the Boilerplate, you can 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>.

## G.2.6 jsFiddle

Sometimes you may want to try out 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'd 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 wanted to share the snippet, you'd need to upload the files to a web server, and the person you're 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 were 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 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 really 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 G.1.



**Figure G.1** 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.

### **G.2.7** *Feature support websites*

For information on the current implementation status of new features, there are a couple of useful websites:

- <http://caniuse.com/>
- <http://html5test.com/>

# *appendix H*

## *Encoding with FFmpeg*

---

You can convert video between container formats and re-encode the audio and video streams within them using several different utilities. In this appendix you'll concentrate on FFmpeg, a command-line tool. Let's review several good reasons for using this tool:

- It's open source and freely downloadable.<sup>1</sup>
- It's available for all the major client and server platforms: Windows, OS X, and Linux.
- Command-line tools lend themselves to scripting, if you have to process many videos.
- It can be called from server-side code.

Let's also look at disadvantages:

- You may be unfamiliar with command-line tools if you've mainly used Windows OSes.
- The sheer flexibility of FFmpeg means it has a confusing plethora of options and configurations.

In this appendix, we'll do our best to walk you through using FFmpeg, but if you're planning to do serious video work, you'll need to get down to the nuts and bolts. If you're only interested in playing around with the video element itself, you can just stick with an easy-to-use tool such as Miro Video Converter (<http://www.mirovideoconverter.com/>).

---

<sup>1</sup> FFmpeg is free, but you may be required to pay a licensing fee to the MPEG-LA if you use it to encode h264 video.

## H.1 How to get FFmpeg

If you don't have FFmpeg, the first thing you need to do is to install it. FFmpeg is primarily distributed as source code. Fortunately, several helpful developers have produced binary versions of it for all the major platforms. Check the officially sanctioned downloads on the website for Windows binaries: <http://ffmpeg.org/download.html>. If you have a Mac, go here: <http://ffmpegmac.net>.

If you run Linux, or your server does, FFmpeg will almost certainly be available through one of your standard repositories (although possibly in the non-free section). Note that to do any encoding, you'll also need libraries to supply the codecs (like MP4 or OGG). On Linux, you'll download them using the same package manager you used to install the main binary, but Windows and Mac users may have to do a bit of extra work.

**NOTE** If you have problems with the examples in this section, we recommend using a virtual machine and installing one of the popular Linux distributions on it. The examples in this chapter have been tested with Fedora 17 using FFmpeg version 0.10.4 recompiled from the source RPM to enable FAAC.

## H.2 Finding out what codecs were used on source video

The first useful thing you can do with FFmpeg is investigate which codecs have been used on your source video. Following is an example command line:

```
ffmpeg -i VID_20120122_132134.mp4
```

This code will produce a whole load of output describing what options your FFmpeg binary was built with, but at the end you should see something like what's shown in the following listing.

### Listing H.1 Output from the `ffmpeg -i` command

```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'VID_20120122_132134.mp4':
  Metadata:
    major_brand      : isom
    minor_version    : 0
    compatible_brands: isom3gp4
  Duration: 00:00:11.59, start: 0.000000, bitrate: 3259 kb/s
  Stream #0.0(eng): Video: h264, yuv420p, 720x480, 3014 kb/s, PAR
65536:65536 DAR 3:2, 30.01 fps, 90k tbr, 90k tbn, 180k tbc
  Stream #0.1(eng): Audio: aac, 16000 Hz, mono, s16, 95 kb/s
```

**Summarizes the entire content—video and audio.**

**Describes the container format; in this case it includes a number of equivalent file extensions.**

**Stream #0.0 is the video stream, h264 encoded.**

**Stream #0.1 is the audio stream, AAC encoded.**

For comparison, the next listing shows the output from a file recorded on a “proper” HD video camera.

### Listing H.2 Output from the `ffmpeg -i` command

```
Input #0, mpegts, from '00003.MTS':
  Duration: 00:00:46.59, start: 0.332544, bitrate: 13305 kb/s
  Program 1
```

**This time the container format is mpegts.**

```
Stream #0.0[0x1011]: Video: h264, yuv420p, 1920x1080 [PAR 1:1 DAR 16:9], 50 fps, 50 tbr, 90k tbn, 50 tbc
Stream #0.1[0x1100]: Audio: ac3, 48000 Hz, 5.1, s16, 384 kb/s
```

→ This time the second stream is AC3 (Dolby Digital).

← The first stream is again an h264-encoded video, but this time it's 1080p HD.

### H.3 Determining container formats and supported codecs

A couple of other useful commands allow you to check what formats and codecs your version of FFmpeg has available. To see a list of available container formats, issue this command:

```
ffmpeg -formats
```

The hyphen indicates a parameter. What comes immediately after the parameter is the parameter name. In the previous command line the parameter is `formats`; in the command line at the start of the section the parameter is `i`, for input file, followed by the data for that parameter. Here it is again to remind you:

```
ffmpeg -i VID_20120122_132134.mp4
```

To see the list of supported codecs, issue this command:

```
ffmpeg -codecs
```

Now that you've learned the basics, the next few sections will cover converting to several key video formats. Note that all of the listings that follow show each `ffmpeg` option on a line by itself for clarity, but when you type the commands into the terminal, they should all be on a single line.

#### Recompiling FFmpeg to add codec support

As mentioned in the previous note, the examples in this chapter have been tested with Fedora 17 using FFmpeg version 0.10.4 recompiled from the source RPM to enable FAAC. This was necessary because the Fedora version of FFmpeg doesn't support AAC by default. Although this may sound scary, it's a straightforward process on Linux; check out this blog post for a description of the process we followed: <http://mng.bz/hF30>.

On Windows and Mac, you may have to search around for a build of FFmpeg that supports all the required codecs.

### H.4 Encoding to MP4/h264 with AAC

The videos you're using in this appendix are already MP4 containers with h264 video and AAC audio. This means for the application you don't need to know how to convert files to this format. In fact, because h264 is a lossy format, re-encoding the files to MP4 at the same resolution will lower the quality. But in real applications it's likely your source video is in a different format or is at least an HD recording (for example, 1080p

is 1920px by 1080px resolution), and one of your key targets will be iOS devices, where all those extra pixels will be wasted. Fortunately, FFmpeg allows you to re-encode a video at a lower resolution in the same way you would re-encode to a different format. A command line for encoding to MP4 with h264 and AAC is shown in the following listing. Again, although it's shown across multiple lines for clarity, you should type it into your command prompt in a single line.

**Listing H.3** `ffmpeg` command line

The `libfaac` encoder (AAC) provides the audio codec.

The `libx264` encoder (h264) provides the video codec.

```
ffmpeg -i VID_20120122_133702.mp4
  -acodec libfaac
  -b:a 96k
  -vcodec libx264
  -preset slower -preset main
  -level 21
  -refs 2
  -b:v 3000k
  -s 720x480
  VID_20120122_133702_2.mp4
```

The audio bit rate will be 96 kilobits per second.

The video bit rate will be approximately 3,000 kilobits per second; lower this number to make a smaller, lower-quality version of the file.

The output video resolution. Adjust this to make a smaller version of the video. Although you could also use this to make a higher-resolution video, doing that will reduce the quality.

This example is re-encoding a video at the same bit rate and the same resolution; in real life there's no need to do this.

FFmpeg includes several presets, which means you don't have to repeatedly enter large numbers of command-line options; these two specify using the slower (higher-quality) encoding method and the main h264 profile.

The input video can be any format that FFmpeg supports. If you have a non-MP4 video, replace the filename (after the `-i`) with your video.

## H.5 *Encoding to MP4/h264 with MP3*

Converting from AAC audio to MP3 isn't a common requirement for web development, but it does allow us to demonstrate another useful feature of FFmpeg. As you already know, h264 is a lossy codec, so re-encoding h264 at the same resolution will reduce quality. But sometimes you may want to re-encode the audio—how can you do that without reducing the quality of the video? Our next listing has the answer—again, type it into your command prompt on a single line.

**Listing H.4** `ffmpeg` command line

```
ffmpeg -i VID_20120122_132134.mp4
  -acodec libmp3lame
  -b:a 96k
  -vcodec copy
  VID_20120122_132134_3.mp4
```

Re-encode the audio to MP3 using the `libmp3lame` codec.

Use the `copy` codec so the video stream is copied across unchanged.

You can use the `copy` codec for several tricks like this. It's also useful if you want to convert between container formats without re-encoding either the audio or video streams.

## H.6 Encoding to WebM/VP8

It's likely that WebM will be your second most required format after MP4. With MP4 and WebM format videos, you'll have more than 80% of desktop browser users supported and significant mobile platforms, including iOS and Android. The next listing shows the command line for converting to the WebM format, using its associated VP8 video codec and Ogg audio.

The libvorbis encoder (Ogg audio) provides the audio codec.

The audio bit rate will be around 96 kilobits per second.

### Listing H.5 FFmpeg command line

```
ffmpeg -i VID_20120122_133702.mp4
-acodec libvorbis
-ac 2
-b:a 96k
-ar 44100
-vcodec libvpx
-b:v 3072k
-s 720x480
VID_20120122_133702.webm
```

The output filename

The input file is an MP4, but it doesn't matter what format it's in as long as FFmpeg can decode it.

The audio sample frequency will be 44100 hertz.

Two audio channels will be used (stereo playback).

The size of the output video will be 720 pixels by 480 pixels, the same as the original in this case.

The video bit rate will be around 3,072 kilobits per second, which should provide comparable file size to the MP4 original from a phone.

## H.7 Encoding to Ogg/Theora

Every browser that supports Ogg/Theora also supports WebM, which means most of the time it'll be technically unnecessary to create an Ogg/Theora version of your video. But if you want to support the older versions of those browsers or you prefer the Ogg/Theora format for ideological reasons, your best bet is to download `ffmpeg2theora`. This is a command-line tool based on the FFmpeg libraries that works out all of the correct video encoding settings: <http://v2v.cc/~j/ffmpeg2theora/>.

Binaries are available for all the major OSes. It's similar to FFmpeg in how you use it; an example command is shown here.

### Listing H.6 Use ffmpeg2theora to convert to Ogg/Theora video

```
ffmpeg2theora
--optimize
--deinterlace
-v 7.8
-F 30
-x 720
-y 480
-A 96
-c 2
-H 44100
-o VID_20120122_132134.ogv
VID_20120122_132134.mp4
```

The video quality—this was chosen by trial and error to give a similar file size to the original.

The frame rate—ffmpeg2theora uses the input frame rate if you leave off this parameter.

Size of the video.

Audio bit rate.

Number of audio channels.

Audio sample rate.

Output filename.



# *appendix I*

## *HTML next*

---

This book concentrates on how best to use the major features of HTML5 available in browsers right now. In this appendix, we'll look at HTML5 capabilities that aren't yet finalized and are under heavy development in a browser's beta versions, such as video captioning, media capture, and full-screen modes. You'll also learn about several proposed features so you can plan for these features of the future web platform, such as peer-to-peer connectivity (for example, for video conferencing) and rotation lock (so that games on mobile devices don't keep flipping between landscape and portrait modes).

Specifically, we'll cover the following:

- Accessing and sharing media
- Providing subtitling and captions for media
- Capturing mouse events outside the bounds of an element
- Expanding elements to full screen
- Measuring orientation to control animation
- Locking the pointer to the center of the screen.

### ***I.1 Accessing and sharing media devices***

Many devices where HTML5 is expected to be used come equipped with built-in cameras, but until now you've needed to use Flash or write a native application to get access to them. One of the goals of the HTML5 spec was to build an open application platform to replace native apps in common use cases. With this in mind a W3C working group has been set up to produce standards for real-time media access and communication (<http://www.w3.org/2011/04/webrtc/>). The charter of this group specifically mentions six deliverables, summarized in table I.1 alongside pointers for the features we'll discuss in this section.

**Table I.1 Deliverables of the Web Real-Time Communications Working Group**

	<b>Deliverable</b>	<b>Being worked on?</b>	<b>In this appendix?</b>
1.	API functions to explore device capabilities; e.g., camera, microphone, speakers	In scope for the Device APIs & Policy Working Group	Not covered
2.	API functions to capture media from local devices (camera and microphone)	In scope for the Device APIs & Policy Working Group, experimental implementations available	Covered in section I.1.1
3.	API functions for encoding and other processing of those media streams		Not covered
4.	API functions for establishing direct peer-to-peer connections, including firewall/NAT traversal	Being worked on at the IETF, experimental implementations available	Discussed in section I.1.2
5.	API functions for decoding and processing (including echo canceling, stream synchronization, and a number of other functions) of those streams at the incoming end		Not covered
6.	Delivery to the user of those media streams via local screens and audio output devices	Part of the HTML5 specification work, experimental implementations available	Covered in chapter 8

In this section you're going to learn about the experimental implementation for point 2 in table I.1, `getUserMedia()`, as well as discuss point 4, which, in concert with `getUserMedia()`, will allow the creation of web applications for telephony and video conferencing.

### **I.1.1 Grab input with `getUserMedia()`**

The `getUserMedia()` function allows you to grab a media stream from the user's device and use it within the browser. The current focus is on audio and video streams, since the elements to output those already exist in HTML5, but there's no reason why other sources of data couldn't be accessed in the future and handled with the File API (see chapter 3) or new elements.

Opera, Google, and Mozilla have already implemented a significant chunk of the functionality targeted by the working group thanks to `getUserMedia()`. You will, of course, need a PC or laptop with a webcam. You could also use your mobile phone or tablet device, but then you'd need some way of making the files you create on your computer available over the network your device is on. This might involve setting up a local web server and possibly fiddling around with your firewall and router settings to allow access to it, or if you already have a web server online, you could upload your files to that.

					
<code>getUserMedia()</code>	21	19 <sup>1</sup>	N/A	12.0	N/A

In this section, to demonstrate how to use `getUserMedia()`, you're going to modify the video telestrator jukebox code from chapter 8 to accept video input from your camera. Figure I.1 shows what you're going to be finishing up with—a live video stream that you can telestrate.

The method signature for `getUserMedia()` is shown in the first listing. It follows the pattern of accepting an array of options along with callback functions, similar to the Geolocation API (see chapter 3).

#### Listing I.1 `getUserMedia()` method signature

```
getUserMedia (options,
              successCallback,
              errorCallback) ←
```

Called if an error occurs when grabbing the media stream.

← An object specifying what sort of media stream you're after, currently either `{audio: true}` or `{video: true}` but extensible in the future (e.g., devices with multiple cameras).

← Called if the media stream is grabbed successfully.



**Figure I.1** Author Rob Crowther after tweaking the code from the video telestrator application in chapter 8 to incorporate a live video stream. With the live video stream appearing on the browser, Rob was then able to telestrate, or draw, additional features on his face, perhaps to impress a potential client during a video meeting.

<sup>1</sup> In Firefox 19 you need to set `media.enabled` to `true` in `about:config` to turn on the experimental support.

The following shows practical code for making `getUserMedia` work in Opera and Chrome. It will grab a video stream from the camera and pipe the output directly into a video element. The annotations indicate where this code fits in the finished code (`index-8.html`) from chapter 8.

**Listing 1.2** `getUserMedia` working in Chrome, Opera, and Firefox

```

var context = canvas[0].getContext('2d');
navigator.getUserMedia =
  navigator.getUserMedia ||
  navigator.mozGetUserMedia ||
  navigator.webkitGetUserMedia;
if (navigator.getUserMedia) {
  navigator.getUserMedia({ video: true },
    successCallback,
    errorCallback);

  function successCallback(stream) {
    console.log('success');
    if (window.webkitURL) {
      v.src = window.webkitURL.createObjectURL(stream);
    } else {
      v.src = stream;
    }
    v.play();
  }

  function errorCallback(error) {
    console.error('An error occurred: [CODE ' + error.code + ']');
    return;
  }
} else {
  console.log('Native web camera streaming (getUserMedia)
    is not supported in this browser.');
```

**Request a video stream. In older examples you'll see a simple string 'video' passed; the current syntax is to pass in an object.**

**Find this line in the finished code from chapter 8; the new code goes after it. You can remove the `change_video()` function and binding.**

**Currently Opera has implemented an unprefixed version of `getUserMedia` in the beta version, whereas Chrome and Firefox have a prefixed version.**

**Everything below is conditional on support existing in the user's browser.**

**Called if the video stream is grabbed successfully.**

**Called if it all goes horribly wrong.**

**Chrome supports the File API; in that browser you have to pass the stream through `createObjectURL`.**

**In Opera and Firefox, attach the stream directly to the video element.**

Now that you have the stream in the video element, everything else functions as before. The canvas element grabs frames from the video, mixes them with the telestrator graphics, and outputs the whole thing.

Being able to let a user display a picture of themselves is a cool gimmick. You can probably see how this could be extended to more practical applications such as snapping photos for entrance badges. But the main goal of this functionality is to allow you to share a video stream across the internet, enabling such applications as video chat. The plan for the future is to combine `getUserMedia()` with peer-to-peer communication protocols. This will enable the creation of video conferencing and telephony applications within the browser; the next section briefly discusses the standard aimed at achieving this, WebRTC (Web Real Time Communication).

### I.1.2 Peer-to-peer media connections with WebRTC

The WebRTC specification is focused on initiating a peer-to-peer connection between two browsers and allowing them to send media streams to each other; a common application of this would be internet telephony or video chat. Google and Mozilla have recently announced their initial implementations of this standard in the development versions of Chrome and Firefox. The following listing shows an excerpt from the Mozilla blog post announcing the availability of the feature,<sup>2</sup> to give you an idea of how the final standard will work.

**Listing I.3 Initiating a peer-to-peer video chat with WebRTC**

```
function initiateCall(user) {
  navigator.mozGetUserMedia({video:true, audio:true},
    function(stream) {
      document.getElementById("localvideo").mozSrcObject = stream;
      document.getElementById("localvideo").play();
      document.getElementById("localvideo").muted = true;
      var pc = new mozRTCPeerConnection();
      pc.addStream(stream);

      pc.onaddstream = function(obj) {
        document.getElementById("remotevideo").mozSrcObject = obj.stream;
        document.getElementById("remotevideo").play();
      };

      pc.createOffer(function(offer) {
        pc.setLocalDescription(offer, function() {
          peerc = pc;
          jQuery.post("offer", {
            to: user,
            from: document.getElementById("user").innerHTML,
            offer: JSON.stringify(offer)
          },
            function() { console.log("Offer sent!"); }
          ).error(error);
        }, error);
      }, error);
    }, error);
  }
}
```

The same `getUserMedia` function you were using in the previous section.

Firefox object that creates a `PeerConnection`.

Local stream is added to the `PeerConnection` object.

An `addstream` event will be fired when the remote client connects; the remote stream will be part of the object parameter.

A connection is initiated by sending an offer via an intermediate server using a standard HTTP POST request.

Because there's no stable support for this feature in current browsers, we won't go into further detail at this point. Instead, in the next section you'll look in detail at another experimental feature that's complementary to audio playback: subtitling and captioning.

<sup>2</sup> Maire Reavy and Robert Numan, editor, "Hello Chrome, it's Firefox calling!", Mozilla Hacks.Mozilla.org, Feb. 4, 2013, <http://mng.bz/kbLL>.

## I.2 Media text tracks: providing media subtitles and captioning

Grabbing webcam and microphone input isn't the only experimental feature in the works for HTML5 media; another potentially very useful feature is text tracks. The central feature of text tracks is to provide subtitles and captioning for hearing-impaired users. All that boils down to is a file format for describing bits of information associated with time spans and a means of presenting that information within the browser. With an API, this sort of structure could be useful in all sorts of ways if you want things to happen in your pages at certain times during playback of media. For example, if your page contained both a video of a presentation and a widget showing the slides from the presentation, then you might want the slideshow widget to automatically switch to the next slide in time with the video.

Fortunately, HTML5 provides such an API. In this section you're going to learn how to use text tracks and the Text Track API by adding subtitles to one of the videos used in chapter 8. Figures I.2 and I.3 show the basic idea: subtitles overlaid on the video corresponding to the current action.

To make this work you'll need Chrome 18 or later, and you should enable the track element in the `about:flags` page; in more recent versions of Chrome (24 and later),



Figure I.2 On the playing video, the caption reads "PASS."



Figure I.3 Later on the playing video, the caption reads "INTERCEPTION."

it's enabled by default. The file `index-3.html` from the chapter 8 code will be used as the basis for your experimentation here.



<b>Text Track API</b>	18	N/A	10	N/A	N/A
-----------------------	----	-----	----	-----	-----

### Local web server required

If you try to run any of the examples in this section directly from the filesystem (with a `file:///` URI), then Chrome will fail to load the Text tracks because of cross-domain security restrictions. In order to make them work, you'll need to either run a local web server (see appendix F where this is discussed) or upload them to an online server.

## 1.2.1 Adding a text track to the videoText

Tracks come in cue files, files containing a series of timestamped cues (the word comes from theater and film/television; think of an actor onstage waiting for a cue to deliver a line). Chrome supports the WebVTT (Web Video Text Tracks) file format for cue files; a sample for you to use is shown in the following listing. To keep things simple in the long run, save this in a file with a name similar to the video file associated with it, such as `VID_20120122_133036.vtt`.

Listing 1.4 VTT Captions `VID_20120122_133036.vtt`

```
WEBVTT

1
00:00:00.400 --> 00:00:01.500
DOWN

2
00:00:01.800 --> 00:00:02.900
SET

3
00:00:03.500 --> 00:00:04.600
HUT

4
00:00:05.000 --> 00:00:07.000
PASS

5
00:00:08.000 --> 00:00:10.000
INTERCEPTION
```

**Text content of the cue.** (points to the cue text)

**A WebVTT file always starts with the identifier WEBVTT on a line by itself.** (points to the first line)

**The file is made up of a number of cues; each one starts with an identifier.** (points to the cue numbers 1-5)

**Each cue has a time span to which it applies, (hh:mm:ss.iii) written twice separated by -->.** (points to the time ranges)

**A blank line separates one cue from the next.** (points to the blank lines between cues)

**The file can contain as many cues as necessary.** (points to the entire content)

To associate the WebVTT file with a `<video>` element, add a `<track>` element, as shown in the next listing. Use the `index-3.html` file from chapter 8's code download;

then you can drop the video element shown in the listing in place of the one already in that file and save it with a new name.

**Listing I.5** index-vtt-1.html, video element with a captions track

```

<video controls
  width="720" height="480">
  <source src="videos/VID_20120122_133036.mp4" type="video/mp4">
  <source src="videos/VID_20120122_133036.webm" type="video/webm">
  <track src="tracks/VID_20120122_133036.vtt"
    kind="captions"
    default>
  Your browser does not support the video element, please
  try <a href="videos/VID_20120122_133036.mp4">downloading
  the video instead</a>
</video>

```

The kind of track this is. See more on kinds of tracks in section F.2.2.

The <track> element; it should go after any <source> elements but before any other content.

Use this track as the default.

And that's all there is to it. With these two additions you can now play the video to recreate the screenshots from the introduction. Check the file index-vtt-1.html in the code download for the complete listing.

## I.2.2 Adding multiple text tracks

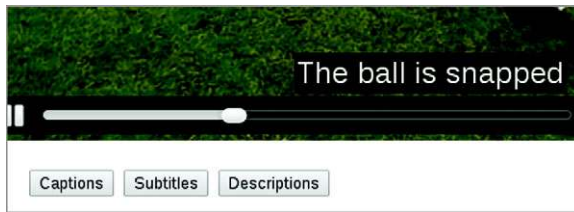
Things get more fun when you add multiple <track> elements. The kind attribute in listing I.5 can be set to several different values depending on the purpose of the timed track. A full list is shown in table I.2.

**Table I.2** Values for the kind attribute

Kind	Description
subtitles	Transcription or translation of the dialogue, suitable for when the sound is available but not understood (e.g., because the user doesn't understand the language of the media resource's audio track). Overlaid on the video.
captions	Transcription or translation of the dialogue, sound effects, relevant musical cues, and other relevant audio information; suitable for when sound is unavailable or not clearly audible (e.g., because it's muted or drowned out by ambient noise, or because the user is deaf). Overlaid on the video; labeled as appropriate for the hearing-impaired.
descriptions	Textual descriptions of the video component of the media resource, intended for audio synthesis when the visual component is obscured, unavailable, or not usable (e.g., because the user is interacting with the application without a screen while driving, or because the user is blind). Synthesized as audio.
chapters	Chapter titles, to be used for navigating the media resource. Displayed as an interactive (potentially nested) list in the user agent's interface.
metadata	Tracks intended for use from script. Not displayed by the user agent.

In this section, you're going to build a simple UI for switching from captions to subtitles to descriptions. Figure I.4 shows video with the caption selected; figure I.5 shows





**Figure I.4** Video with captions selected



**Figure I.5** Video with subtitles selected

the subtitles after clicking the middle button. The third button, for descriptions, you'll deal with in the following section.

For this to work you'll need additional WebVTT files. Listings I.6 through I.8 show the files you need for the captions, subtitles, and descriptions. Note that the long file-names are provided in code comments.

Listing I.6 Captions	Listing I.7 Subtitles	Listing I.8 Descriptions
<pre>// VID_20120122_133036- captions.vtt  WEBVTT  1 00:00:00.400 --&gt; 00:00:01.500 Players line up  2 00:00:01.800 --&gt; 00:00:02.900 Offense gets ready  3 00:00:03.500 --&gt; 00:00:04.600 The ball is snapped  4 00:00:05.000 --&gt; 00:00:07.000 It's a pass</pre>	<pre>// VID_20120122_133036- subtitles-enGB.vtt  WEBVTT  1 00:00:00.400 --&gt; 00:00:01.500 DOWN  2 00:00:01.800 --&gt; 00:00:02.900 SET  3 00:00:03.500 --&gt; 00:00:04.600 HUT  4 00:00:05.000 --&gt; 00:00:07.000 PASS</pre>	<pre>// VID_20120122_133036- description.vtt  WEBVTT  1 00:00:00.000 --&gt; 00:00:04.000 A rugby field in Oxfordshire, American Footballers get ready for the play  2 00:00:04.000 --&gt; 00:00:08.000 The ball is snapped, the quarterback drops back to pass  3 00:00:08.000 --&gt; 00:00:09.000 The pass is thrown wide of the receiver,</pre>

5	5	a defender makes the
00:00:08.000 -->	00:00:08.000 -->	interception
00:00:10.000	00:00:10.000	
It's picked off	INTERCEPTION	4
		00:00:09.000 -->
		00:00:12.000
		The defender sets off
		with the ball, the
		offensive players in
		pursuit

Now add the files to the `<video>` element in multiple `<track>` elements, as shown in the following listing.

**Listing I.9** `<video>` element with multiple `<track>` elements

```

<video controls
  width="720" height="480">
  <source src="videos/VID_20120122_133036.mp4" type="video/mp4">
  <source src="videos/VID_20120122_133036.webm" type="video/webm">
  <track src="tracks/VID_20120122_133036-captions.vtt"
    kind="captions"
    default
    label="Captions">
  <track src="tracks/VID_20120122_133036-subtitles-enGB.vtt"
    kind="subtitles"
    srclang="en-GB"
    label="English Subtitles">
  <track src="tracks/VID_20120122_133036-description.vtt"
    kind="descriptions"
    label="Text Description">
  Your browser does not support for video element, please
  try <a href="videos/VID_20120122_133036.mp4">downloading
  the video instead</a>
</video>

```

**As before, `<track>` elements come after `<source>` elements and before any other content.**

**Each track can also have a label; in the future it is envisaged user agents will offer a UI for selecting between tracks using this label.**

**The kind attribute differentiates the tracks.**

**If the track is of kind subtitles or captions, then the srclang attribute allows the browser to select the correct one based on the user's language preferences.**

Next, you'll need buttons to hang the functionality from. Just as you did in chapter 8, add a `<menu>` to the page under the `<video>` element that looks like the following code.

**Listing I.10** A `<menu>` for choosing the text track

```

<menu>
  <button>Captions</button>
  <button>Subtitles</button>
  <button>Descriptions</button>
</menu>

```

Finally, you need code that makes actions happen when the buttons are clicked. You can reuse the menu-handling function from chapter 8 with appropriate changes to reflect the new functions. The code is shown in the next listing.

## Listing I.11 Changing the track shown with JavaScript

```

$('menu').bind('click', function(event) {
  var action = $(event.target).text().trim();
  var p = $('#player video:first-of-type')[0];
  switch (action) {
    case 'Captions':
      p.textTracks[0].mode = "showing";
      p.textTracks[1].mode
        = "hidden";
      p.textTracks[2].mode = "hidden";
      break;
    case 'Subtitles':
      p.textTracks[0].mode = "hidden";
      p.textTracks[1].mode = "showing";
      p.textTracks[2].mode = "hidden";
      break;
    case 'Descriptions':
      p.textTracks[0].mode = "hidden";
      p.textTracks[1].mode = "hidden";
      p.textTracks[2].mode = "showing";
      break;
  }
  return false;
});

```

In this case, you know which tracks are which, so you can access them directly. You could use the kind attribute or any other DOM methods to work out which is which.

Use the text of the button to determine the action.

You can get at the text tracks through the textTracks array on the <video> element.

Whether or not the track is displayed is determined by the mode property of the track (more on this in I.2.3).

DISABLED, HIDDEN, and SHOWING are available as properties of TextTrack (see table I.3).

The complete listing is available as `index-vtt-3.html` in the book's code download.

If you experiment with this latest listing, you'll note that clicking the Descriptions button doesn't do anything. This is because tracks of kind `description` aren't intended for visual display but for aural accompaniment. But this does give us an opportunity to experiment with the rest of the Text Track API. In the next section you'll use the API to extract the text content from the description track.

### I.2.3 The Text Track API

You got a glimpse of the Text Track API in section I.2.2, where you used the `mode` attribute. In this section, you'll go into more depth, covering reading individual cues from a track and listening to events that are fired when the active cue changes. To begin, you will use the API to grab text from the `description` track you added in section I.2.2. Figure I.6 shows the basic idea; when the Descriptions button is clicked, content from the track is shown.



Figure I.6 JavaScript has been used to extract the text content of the normally invisible description track and display it on the screen.

Before you dive into the code, review table I.3, which lists the properties and methods of the Text Track API.

**Table I.3 The Text Track API**

Name	Type	Description
kind	String property	The kind of text track, corresponds to the <code>kind</code> attribute.
label	String property	A human-readable label, corresponds to the <code>label</code> attribute.
language	String property	The language of the track, such as <code>en-US</code> , corresponds to the <code>srclang</code> attribute.
mode	Integer property	The mode of the track: <code>DISABLED</code> (the track will not be loaded), <code>HIDDEN</code> (the track will be loaded but not displayed), or <code>SHOWING</code> (the track will be loaded and displayed if appropriate).
cues	Array property	An array containing the individual cues from the track.
activeCues	Array property	An array containing the cues that apply to the current point in the media.
<code>addCue()</code>	Method	Add a cue to the cues array.
<code>removeCue()</code>	Method	Remove a cue from the cues array.

The property you need here is the `activeCues` array, or a set of cues that should be displaying currently. Displaying the text from the active cue of the `description` track is as simple as grabbing the first element of the array and using the `text` property, as shown in the following listing. Replace the last case in the `select` statement in listing I.11 with this one.

**Listing I.12 Display the active cue**

```

case 'Descriptions':
  p.textTracks[0].mode = "hidden";
  p.textTracks[1].mode = "hidden";
  p.textTracks[2].mode = "showing";
  $('#desc').html(p.textTracks[2].activeCues[0].text);
  break;

```

The mode property seen in listing I.10.

The activeCues array is made up of TextTrackCue objects.

As you might guess from listing I.12, the cue objects have their own API. Each cue is of type `TextTrackCue`; a complete list of properties is shown in table I.4.

**Table I.4 The TextTrackCue API**

Attribute/method	Type	Description
<code>track</code>	<code>TextTrack</code>	The track to which this cue belongs.
<code>id</code>	<code>string</code>	Unique identifier for the cue.
<code>startTime</code>	<code>double</code>	The time the cue starts.

**Table I.4** The TextTrackCue API (*continued*)

Attribute/method	Type	Description
<code>endTime</code>	double	The time the cue ends.
<code>pauseOnExit</code>	boolean	Returns <code>true</code> if the media will pause at the end of the cue.
<code>vertical</code>	string	Returns a string describing the TextTrack writing direction. Either empty (horizontal), "rl" for vertical growing left, or "lr" for vertical growing right.
<code>snapToLines</code>	boolean	Returns <code>true</code> if the cue is set to render at a point that's a multiple of the height of the starting line plus the starting point or <code>false</code> if its position is a point at a percentage of the overall size of the video.
<code>line</code>	long (or "auto")	Returns a number giving the position of the line or "auto" if there are multiple cues.
<code>position</code>	long	A number giving the position of the text of the cue within each line, to be interpreted as a percentage of the video
<code>size</code>	long	A number giving the size of the box within which the text of each line of the cue is to be aligned, to be interpreted as a percentage of the video.
<code>align</code>	string	"start", "middle", "end", "left", or "right".
<code>text</code>	string	The text of the cue.
<code>getCueAsHTML()</code>	DocumentFragment	Returns the text of the cue as HTML.

There's a working version of this code in `index-vtt-4.html` in the code download in case you don't want to piece it together from the snippets here. If you load it, play the video and click the Descriptions button a few times; you should see the descriptions appear below the video. But you'll also probably see the odd error message like that shown in figure I.7.

The error in figure I.7 can have two main causes:

- There isn't a cue available for the current time.
- The text track hasn't loaded yet.

**Figure I.7** An error trying to access the currently active cue in the text track

Determining whether there is a cue currently available follows the normal rules of JavaScript; simply do a test like `if (typeof p.textTracks[2].activeCues[0] !== 'undefined')` before attempting to access the text property. In most real-life cases, you'll do this as a matter of course. But it's clearly the second issue that's the problem here because our description cue file has no gaps in its time coverage. One approach to solving the second issue would be to listen for the load event of the text track, something we'll discuss further in the next section when you learn about events. In the meantime, we'll look at two alternative approaches to solving the second issue:

- Loading all the text tracks in advance
- Checking to see if the text track has loaded

#### LOADING THE TEXT TRACKS IN ADVANCE

Unless the default attribute is applied to the track, it will be in the default mode of `DISABLED`. For the browser to load the tracks, you need to set them to either `HIDDEN` or `VISIBLE`. It's easy enough to do this in the ready event you already have in your code, as shown in the following listing.

**Listing I.13** Adjust the document ready event

```
$(document).ready(
  function() {
    $('#playlist').bind('click', change_video);
    var p = $('#player video:first-of-type')[0];
    p.textTracks[0].mode = "hidden";
    p.textTracks[1].mode = "showing";
    p.textTracks[2].mode = "hidden";
```

← The document ready event you're already using.

← The three new lines are added here. If you have more than three tracks, you might consider using a loop instead.

Find the complete working example in the `index-vtt-4a.html` file.

#### CHECKING TO SEE IF THE TRACK IS LOADED

Text tracks have a ready state similar to other dynamically loadable objects (for example, XMLHTTP requests). The complete list of values for text tracks is shown in table I.5.

**Table I.5** Track `readyState` values

State	Value	Description
NONE	0	The track has not been loaded.
LOADING	1	The track is in the process of being loaded.
LOADED	2	The track has been loaded.
ERROR	3	Attempting to load the track led to an error.

Clearly, all you now need to do is check that the `readyState` of the track is 2 before you attempt to access the text property. The next listing shows an updated descriptions case for the menu-handling function.

## Listing I.14 Check the track ready state

```

case 'Descriptions':
    p.textTracks[0].mode = "hidden";
    p.textTracks[1].mode = "hidden";
    p.textTracks[2].mode = "hidden";
    var t = v.find('track[kind="descriptions]');
    if (t[0].readyState == 2) {
        $('#desc').html(p.textTracks[2].activeCues[0].text);
    }
    break;

```

Use standard jQuery to get the descriptions track.

The ready state is available on the track element.

Find the complete working example in the index-vtt-4b.html file.

### I.2.4 Using TextTrack events

Although the examples in the previous sections show some useful techniques and allow you to explore the API, it's more in keeping with JavaScript to deal with text tracks in an event-driven style. The track element has a load event that allows you to call a function when loading is complete in the same way you've done hundreds of times before. Because we have limited space here, you're not going to do that right now; instead, you're going to learn about an event that's specific to timed tracks: cuechange.

The cuechange event is fired every time a new cue is to be displayed. If you handle the cuechange event, then instead of showing the current description whenever the user clicks the Menu button, you can instead show the descriptions at the appropriate time. The following listing updates the switch statement in the menu handler to attach an event to the description track's oncuechange property when the Descriptions button is clicked.

## Listing I.15 Listening to the cuechange event

```

switch (action) {
    case 'Captions':
        p.textTracks[0].mode = "showing";
        p.textTracks[1].mode = "hidden";
        p.textTracks[2].oncuechange = null;
        $('#desc').html('');
        break;
    case 'Subtitles':
        p.textTracks[0].mode = "hidden";
        p.textTracks[1].mode = "showing";
        p.textTracks[2].oncuechange = null;
        $('#desc').html('');
        break;
    case 'Descriptions':
        p.textTracks[0].mode = "hidden";
        p.textTracks[1].mode = "hidden";
        $('#desc').html(p.textTracks[2].activeCues[0].text);
        p.textTracks[2].oncuechange = function() {
            if (typeof this.activeCues[0] !== 'undefined') {

```

If the captions or subtitles are playing, remove the oncuechange event handler.

When the user clicks the Descriptions button, attach an oncuechange handler.

```

        $('#desc').html(this.activeCues[0].text);
    }
};
break;
}

```

The body of the handler is the same code you were using already.

Now when the user clicks the Descriptions button, the descriptions will be updated automatically as the cues change. Grab file index-vtt-5.html to try it for yourself.

Although you've used the API to read the descriptions in this section, a more common use would be to perform an action at a particular time with the media. If you refer to table I.2, you'll note that the last kind of track is metadata. This can be used for any kind of data you want to use from within your scripts; for example, you could have a series of cues populated with data in JSON format.

Before we move on to other things, let's take a look at styling the cues as they appear on the video.

### 1.2.5 Styling text tracks

Text tracks can contain simple markup. Typographical elements like `<b>` and `<i>` are allowed. Figure I.8 shows an updated captions file in action.

The new version of the captions file is shown in the following listing.

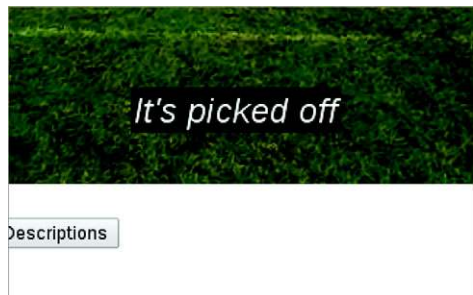


Figure I.8 Captions using the `<i>` element

#### Listing I.16 A cue file with simple formatting

```

WEBVTT

1
00:00:00.400 --> 00:00:01.500
<i>Players line up</i>

2
00:00:01.800 --> 00:00:02.900
<i>Offense gets ready</i>

3
00:00:03.500 --> 00:00:04.600
<i>The ball is snapped</i>

4
00:00:05.000 --> 00:00:07.000
<i>It's a pass</i>

5
00:00:08.000 --> 00:00:10.000
<i>It's picked off</i>

```

In the future it will also be possible to style the cues in CSS using the `::cue` pseudo; unfortunately, Chrome hasn't yet implemented this.



That's enough media APIs and features for now. For the remainder of this chapter, you're going to learn about experimental APIs that will be particularly useful for games or mobile devices (or games on mobile devices!).

### **I.3 APIs for gaming and mobile**

This section groups together a set of HTML5 APIs that are targeted at gaming, with particular reference to gaming on mobile devices. In this section you will

- Build a test bed, which you'll use to explore the APIs
- Target mouse events at a single element with `setCapture`
- Expand an element to full screen
- Replace mouse events with touch events
- Replace mouse events with orientation events
- Use the vibration and battery APIs
- Use the pointer lock API to enable immersive experiences

#### **I.3.1 Preparing a test bed—the return of Wilson**

We need something with which to demonstrate all these APIs, so initially you're going to build a simple canvas app (see chapter 6 for background), which draws an object that will then follow the mouse around the screen. You'll use this as the basis for all the experiments until the end of the appendix, so it's worth spending time understanding how to put it together even if the techniques are familiar to you.

Your starting point for API exploration is a Wilson head, which follows your mouse pointer around. The result is shown in figure I.9.

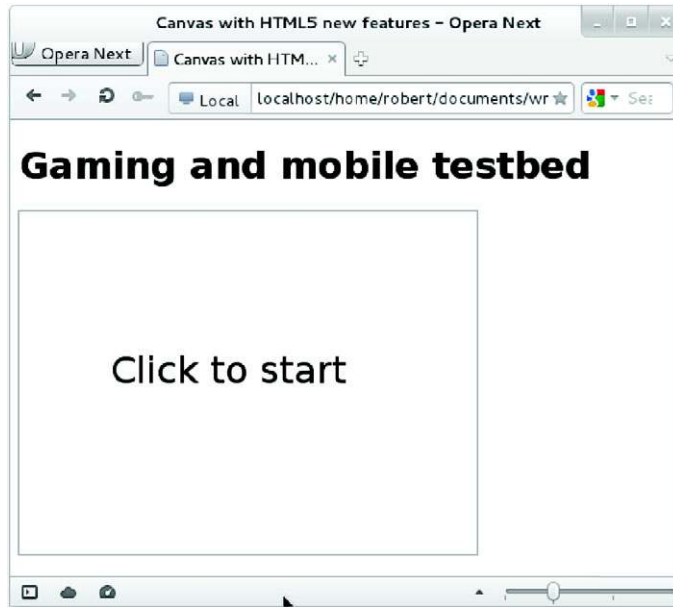
The process for creating this test bed is as follows:

- Step 1: Create a basic page with a `<canvas>` element.
- Step 2: Create a function that draws an image at a particular position on the canvas.
- Step 3: Detect and record mouse movement.
- Step 4: Update the position of the image each time the animation is updated.

## **Gaming and mobile testbed**



**Figure I.9** We have a Wilson following our mouse pointer! Debugging information displays in the background showing the values of important variables.



**Figure I.10** A simple test bed for exploring APIs for gaming and mobile applications

#### STEP 1: CREATE A BASIC PAGE WITH A CANVAS ELEMENT

To start, you need a simple HTML5 document like the one shown in figure I.10.

The code for figure I.10 is shown in the following listing. In addition to the code in this listing, you'll need the `requestAnimationFrame` polyfill from <https://gist.github.com/1579671> that you used in chapter 8. It's also in the code download.

#### Listing I.17 Example app page framework

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Canvas with HTML5 new features</title>
  <script>
    function go() { }
    function draw_welcome() {
      var canvas = document.getElementById('canvas');
      canvas.width = 400;
      canvas.height = 300;
      if (canvas.getContext) {
        var ctx = canvas.getContext('2d');
        ctx.font = "24pt sans-serif";
        ctx.fillText('Click to start ',
          canvas.width/2-120,
          canvas.height/2);
      }
    }
    window.addEventListener("load", draw_welcome, false);
  </script>
</head>
```

The `go()` function, which will be doing most of the setup in this and later sections.

Function to display a welcome message when the page loads.

Attach the function to the load event.

```

<body>
  <h1>Gaming and mobile testbed</h1>
  <canvas id="canvas" onclick="go()"></canvas>
</body>
</html>

```

## STEP 2: CREATE A FUNCTION TO DRAW AN IMAGE AT A PARTICULAR POSITION

Our image will be the Wilson character from chapter 7, this time in canvas form. Because you'll need to maintain state information about where Wilson is, what he's aiming for, and how quickly he's moving toward it, the function to draw Wilson will be part of an object. Listing I.18 shows the initial version of this. It's a long and mostly irrelevant listing as far as the new features are concerned, but the rest of the examples won't work without this bit of code, so you need it. There's no need to understand it thoroughly. This code should go between the `go()` function and the `draw_welcome()` function in listing I.17.

**Listing I.18** The wilson object

```

var wilson = {
  x: 0,
  y: 0,
  target_x: 0,
  target_y: 0,
  v_x: 0,
  v_y: 0,
  draw: function (canvas) {
    var tl_x = wilson.x - 70;
    var tl_y = wilson.y - 70;
    if (canvas.getContext){
      var context = canvas.getContext('2d');
      context.beginPath();
      context.arc(tl_x + 70, tl_y + 70,
                  70, 0, 2 * Math.PI, false);
      context.fillStyle = 'yellow';
      context.fill();
      context.beginPath();
      context.arc(tl_x + 45, tl_y + 57,
                  7, 0, 1 * Math.PI, true);
      context.moveTo(tl_x + 100, tl_y + 57);
      context.arc(tl_x + 95, tl_y + 57,
                  7, 0, 1 * Math.PI, true);
      context.fillStyle = '#777777';
      context.fill();
      context.beginPath();
      context.arc(tl_x + 70, tl_y + 90,
                  30, 0, 1 * Math.PI, false);
      context.lineTo(tl_x + 100, tl_y + 90);
      context.fillStyle = 'ffffff';
      context.fill();
      context.stroke();
      context.fillStyle = 'black';
      context.lineWidth = 3;

```

← Variables to store the current state of Wilson.

← For ease of use, you're storing the center point, but the drawing code works from the top-left corner down, so calculate the offset here.

```

context.lineJoin = 'round';
context.lineCap = 'round';
context.beginPath();
context.moveTo(tl_x + 30,tl_y + 40);
context.lineTo(tl_x + 30,tl_y + 70);
context.lineTo(tl_x + 60,tl_y + 70);
context.lineTo(tl_x + 60,tl_y + 40);
context.lineTo(tl_x + 30,tl_y + 40);
context.moveTo(tl_x + 60,tl_y + 60);
context.lineTo(tl_x + 80,tl_y + 60);
context.moveTo(tl_x + 80,tl_y + 40);
context.lineTo(tl_x + 80,tl_y + 70);
context.lineTo(tl_x + 110,tl_y + 70);
context.lineTo(tl_x + 110,tl_y + 40);
context.lineTo(tl_x + 80,tl_y + 40);
context.stroke();
    }
}
}

```

### STEP 3: DETECT AND RECORD MOUSE MOVEMENT

Next, record the mouse movement. The function in the following listing will add an event listener to the <canvas> element, which updates the wilson object when mouse movement is detected. Add this code directly after the wilson object you added in listing I.18.

**Listing I.19** Listen to mouse events and update Wilson's target position

```

function get_mouse_pos(obj, evt){
    var top = 0, left = 0;
    while (obj && obj.tagName != 'BODY') {
        top += obj.offsetTop;
        left += obj.offsetLeft;
        obj = obj.offsetParent;
    }
    var mouseX = evt.clientX - left + window.pageXOffset;
    var mouseY = evt.clientY - top + window.pageYOffset;
    return { x: mouseX, y: mouseY };
}

function follow_mouse() {
    var canvas = document.getElementById('canvas');
    var context = canvas.getContext('2d');
    canvas.addEventListener('mousemove', function(evt){
        var mousePos = get_mouse_pos(canvas, evt);
        wilson.target_x = mousePos.x;
        wilson.target_y = mousePos.y;
    }, false);
};

```

← Calculate the mouse position relative to the top left of a given element.

← Set up the event listener to capture mouse movement.

Note that you're not making any attempt to update the canvas within this handler. You want all drawing to happen in the requestAnimationFrame loop to minimize resource usage, so this function simply records the position and exits. If the browser is ready to make use of the position, it will; otherwise, it will be replaced by the next mousemove event.

**STEP 4: UPDATE THE POSITION OF THE IMAGE EACH TIME THE ANIMATION IS UPDATED**

So now you need a function to be called each animation frame to move Wilson toward the current mouse position. The two functions in the listing that follows should be added to the wilson object so that you can use them later.

**Listing I.20 Move Wilson toward the target position**

Calculates how far to move the current position to the target position; the farther away, the faster it will move.

```

get_v: function(t,c) {
    var v = Math.floor(Math.sqrt(t*2) - Math.sqrt(c*2));
    if (isNaN(v)) { v = 0; }
    if (v == 0 && c != t) { v = (t - c) / Math.abs(t - c); }
    return v;
},
update_xy: function() {
    wilson.v_x = wilson.get_v(wilson.target_x,wilson.x);
    wilson.v_y = wilson.get_v(wilson.target_y,wilson.y);
    wilson.x += wilson.v_x;
    wilson.y += wilson.v_y;
    if (isNaN(wilson.x) || wilson.x < 0) { wilson.x = 0; }
    if (isNaN(wilson.y) || wilson.y < 0) { wilson.y = 0; }
},

```

If the previous calculation produced an invalid number, use zero.

If the motion is 0 but the points don't yet match, move 1 pixel in the correct direction.

Update the x and y velocities.

Add the velocity to the current position.

Check that the bounds haven't been exceeded in some way.

The code in the previous listing is a bit rough and ready, but it will produce a fairly natural-looking deceleration toward the target point everywhere but at the edges without your having to worry about mapping floating point numbers into an integer coordinate space. When you write your killer gaming app based on this sample, you should definitely spend a little more time on it!

Now you're ready to hook the various components together in the `go()` function. Replace your `go()` function from listing I.17 with the version in the following listing.

**Listing I.21 Draw Wilson**

```

function go() {
    var canvas = document.getElementById('canvas');
    canvas.width--;
    canvas.width++;
    if (canvas.getContext) {
        var context = canvas.getContext('2d');
        wilson.x = canvas.width/2;
        wilson.y = canvas.height/2;
        wilson.target_x = wilson.x;
        wilson.target_y = wilson.y;
        wilson.draw(canvas);
        follow_mouse();
        (function anim_loop(){
            requestAnimationFrame(anim_loop);
            canvas.width--;
            canvas.width++;
            wilson.update_xy();

```

Clear the canvas.

Set Wilson's draw point to be the midpoint of the canvas.

Draw Wilson.

```

        wilson.draw(canvas);
    } ();
}

```

Now that you have a basic example application in place, let's look at some APIs!

### I.3.2 The Mouse Event Capture API: continuing movement beyond the bounds of an element

The first API you're going to look at is Mouse Event Capture, comprising the `setCapture()` and `releaseCapture()` methods. The problem this API is trying to solve is that mouse events immediately stop the moment the mouse pointer moves outside of the element where they're being captured. The problem is illustrated in figure I.11.

**NOTE** The Mouse Event Capture API is not yet part of any standard, but that's more because no one has decided where to put it than that it's not useful or won't be standardized. The HTML5 and W3C recommendation requirement of "two compatible implementations" has already been met. It's possible it will appear in the DOM Level 3 specification.



**Mouse Capture**

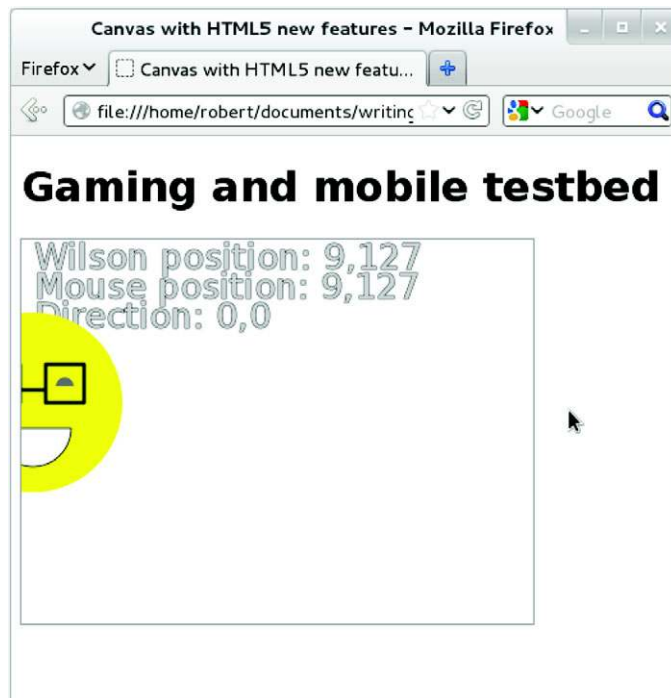
N/A

4

5.5

N/A

N/A



**Figure I.11** Although the mouse pointer has moved to the right, Wilson has not followed because the pointer movement occurred outside of the bounds of the element.



**Figure I.12** With the capture events API Wilson continues to follow the pointer when it leaves the element or even the browser window.

This is obviously annoying behavior if your user is controlling a game with their mouse, because as soon as the mouse pointer leaves the element, the game piece they're manipulating will stop responding. Figure I.12 shows the difference when `setCapture()` is used (you'll just have to trust that I did the same thing with the mouse pointer—or download the sample code and try it for yourself).

The next listing shows how you could use the event capturing API to work around the issue. It's a replacement for the `follow_mouse()` function you implemented in the previous section.

#### Listing I.22 Following the mouse with `setCapture()`

```
function follow_mouse() {
    var canvas = document.getElementById('canvas');
    var context = canvas.getContext('2d');
    function mouse_down(e) {
        e.target.setCapture();
        e.target.addEventListener("mousemove",
                                mouse_moved, false);
    }
    function mouse_up(e) {
        e.target.removeEventListener("mousemove",
                                    mouse_moved, false);
    }
    function mouse_moved(evt) {
        var mousePos = get_mouse_pos(canvas, evt);
    }
}
```

← The `setCapture()` method needs to be called inside a mousedown event.

← The movement-tracking function is the same as before except it's now a declared function instead of an anonymous one.

```

wilson.target_x = mousePos.x;
wilson.target_y = mousePos.y;
}
canvas.addEventListener('mousedown', mouse_down , false);
canvas.addEventListener('mouseup', mouse_up , false);
};

```

**If you comment out this line, then mouse events will continue to be captured by the canvas after the mouse button is released.**

**For best results here, implement additional bounds checking on Wilson's movement; otherwise, he'll leave the canvas at the right or bottom.**

That's all you need to know about capturing mouse events on an element, but there's an alternative approach you might want to consider. Instead of attempting to capture mouse movement as it moves outside the element, you could make the element take up the full screen. You'll learn about the Full-Screen API in the next section.

### I.3.3 The Full-Screen API: expanding any element to full screen

The Full-Screen API allows any element to expand to take up the entire screen. The element will be the only thing displayed; no browser chrome will be visible. Figure I.13 shows Wilson in full-screen mode in Firefox12. The text "Press ESC at any time to exit fullscreen" will fade out after a few seconds; it's added as a security measure so that it's obvious to users that they've entered full-screen mode. Otherwise, a nefarious script could simulate their entire desktop in order to steal passwords and other personal information.

A summary of the Full-Screen API is shown in table I.6.



**Figure I.13** Wilson entering full-screen mode in Firefox with the user information overlay showing



**Table I.6 The Full-Screen API**

Property/event name	Type	
<code>requestFullscreen()</code>	Method	Ask for an element to go to full screen.
<code>fullscreenEnabled</code>	Read-only boolean	Is the page currently in full-screen mode?
<code>fullscreenElement</code>	Read-only enabled	If full screen is enabled, this property will be set to the element that's full screen.
<code>fullscreenchange</code>	Event	The <code>fullscreenEnabled</code> state has changed.



**Full-Screen API**    15    9    N/A    N/A    N/A

**ENTERING FULL-SCREEN MODE**

Entering full-screen mode is quite straightforward, even accounting for experimental browser implementations. Undo any changes you made to your example in section I.3.2, and then place the code from the following listing at the top of your `go()` function.

**Listing I.23 Request FullScreen for the <canvas> element**

```
function go() {
    var canvas = document.getElementById("canvas");
    if (canvas.requestFullscreen) {
        canvas.requestFullscreen();
    } else if (canvas.mozRequestFullscreen) {
        canvas.mozRequestFullscreen();
    } else if (canvas.webkitRequestFullscreen) {
        canvas.webkitRequestFullscreen();
    }
}
```

The browser has implemented the standard.

`requestFullscreen` must be called from an event handler. In this case the `go()` function is being called from a click event, so you're okay.

Mozilla's experimental implementation.

Chrome's experimental implementation.

**STYLING THE FULL-SCREEN BACKGROUND**

If you try your new example in both Firefox and Chrome, you'll immediately notice a compatibility issue: Firefox defaults the full-screen background to black; Chrome defaults it to white. Fortunately, this problem can be overcome with CSS. Check out the following listing, which uses the experimental `:full-screen` pseudo class to set a consistent background color.

**Listing I.24 Set CSS styles that only apply in full-screen mode**

```
canvas:-moz-full-screen {
    background: #006;
    outline: none;
}
canvas:-webkit-full-screen {
    background: #006;
    outline: none;
}
```

```

}
canvas:fullscreen {
  background: #006;
  outline: none;
}

```

### EXITING FULL-SCREEN MODE

Now that the full screen has a pleasant dark-blue background in all browsers, the next issue to consider is what happens when the user exits full-screen mode by hitting Esc. In a more complex app, you may want to pause an activity or take the opportunity to switch to a different mode of interaction. To do this, listen to the `fullscreenchange` event. Our next listing has some example code.

#### Listing I.25 Add a listener to the `fullscreenchange` event

```

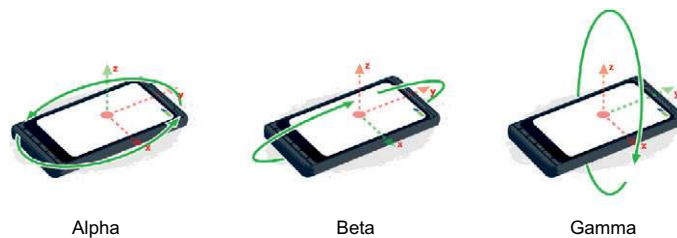
document.addEventListener("fullscreenchange", function () {
  console.log(document.fullscreen);
}, false);
document.addEventListener("mozfullscreenchange", function () {
  console.log(document.mozFullScreen);
}, false);
document.addEventListener("webkitfullscreenchange", function () {
  console.log(document.webkitIsFullScreen);
}, false);

```

Feel free to experiment with these events further; we're not going to go into any more detail. In the next section, you're going to jump to mobile; to get full advantage you should have an iPhone or Android device handy.

### I.3.4 The Device Orientation API: controlling on-screen movement by tilting a device

The Device Orientation API delivers events to your web page that correspond to the movement of the device. The device can be rotated around three axes; have a look at figure I.14.



**Figure I.14** The directions of motion used in the Device Orientation API. (Based on diagrams at [https://developer.mozilla.org/en/DOM/Orientation\\_and\\_motion\\_data\\_explained](https://developer.mozilla.org/en/DOM/Orientation_and_motion_data_explained).)



**Figure I.15** Full-screen mode in Firefox Android version, using device orientation to control Wilson

Figure I.15 shows Wilson in full-screen mode on an Android device being controlled by the Device Orientation API, although the angle of the device is hard to tell from a flat screenshot!

So how do you take advantage of the Device Orientation API? It all depends on the `deviceorientation` event. The following listing adapts the now inaccurately named `follow_mouse()` function to listen to this event. For this listing to work, you'll need a device with a built-in accelerometer such as an Android or iOS phone or tablet.

**Listing I.26** Update the `follow_mouse()` function to use device-orientation data

```
function follow_mouse() {
    var canvas = document.getElementById('canvas');
    var context = canvas.getContext('2d');
    function handleOrientation(orientData) {
        var absolute = orientData.absolute;
        var alpha = orientData.alpha;
        var beta = orientData.beta;
        var gamma = orientData.gamma;
        wilson.v_x = -1 * beta;
        wilson.v_y = gamma;
    }
    window.addEventListener("deviceorientation",
        handleOrientation, true);
};
```

**Rotation around the y-axis in degrees ranging between -90 and 90.**

**A flag indicating whether the orientation returned is in the context of earth's coordinate frame or relative to the device.**

**Rotation around the z-axis in degrees ranging between 0 and 360.**

**Rotation around the x-axis in degrees ranging between -180 and 180.**

**Plug the beta and gamma rotation directly into the wilson object's velocity properties.**

Because of the slightly different approach in setting the velocity—with mouse events you're aiming at a target; with orientation events you're linking the velocity directly to

the angle—the `update_xy()` function in the `wilson` object also needs updating. The following listing has the code.

**Listing I.27 Update Wilson's X and Y positions**

```
update_xy: function(canvas) {
  wilson.x += wilson.v_x;
  wilson.y += wilson.v_y;
  if (isNaN(wilson.x) || wilson.x < 0) { wilson.x = 0; }
  if (isNaN(wilson.y) || wilson.y < 0) { wilson.y = 0; }
  if (wilson.x > canvas.width) { wilson.x = canvas.width; }
  if (wilson.y > canvas.height) { wilson.y = canvas.height; }
},
```

← No need to calculate the velocity; use it directly.

← The bounds of Wilson's movement are no longer limited to the bounds of mouse movement in the element, so add a check to keep him in view.

#### **FUTURE IMPROVEMENTS: LOCKORIENTATION**

If you play with this example on your mobile device, you'll probably notice a minor annoyance: Everything is set up assuming landscape mode, but as you rotate the device, it's very easy to flip the orientation to portrait mode. At the moment your only option is to detect the orientation change and adjust your code to deal with both portrait and landscape modes. But plans are afoot to provide web apps with the same ability to lock orientation that native apps get. Unfortunately, experimental implementations aren't yet available.

### **I.3.5 The Vibration API: accessing a mobile device's vibration function**

Mobile devices offer alternative methods for feedback as well as the alternative methods for input you looked at in the preceding sections. The Vibration API is a proposal from Mozilla to provide access to a mobile's built-in vibration function. You can adapt the example from section I.3.3 to vibrate when Wilson hits the edges of the screen by adjusting the `update_xy()` function again, as shown in the next listing.



**Vibration API**    N/A    11    N/A    N/A    N/A

**Listing I.28 Vibrate when screen edges are reached**

```
update_xy: function(canvas) {
  wilson.x += wilson.v_x;
  wilson.y += wilson.v_y;
  if (isNaN(wilson.x) || wilson.x < 0) {
    wilson.x = 0;
    navigator.mozVibrate(50);
  }
  if (isNaN(wilson.y) || wilson.y < 0) {
    wilson.y = 0;
    navigator.mozVibrate(50);
  }
}
```

← Vibrate for 50 milliseconds.

```

    if (wilson.x > canvas.width) {
        wilson.x = canvas.width;
        navigator.mozVibrate(50);
    }
    if (wilson.y > canvas.height) {
        wilson.y = canvas.height;
        navigator.mozVibrate(50);
    }
},

```

← Vibrate for 50 milliseconds.

The Vibration API can also create a pattern if you pass it an array rather than a single number. The values are again times in milliseconds, but now they alternate between vibrating and not vibrating. The following listing shows an example of this.

#### Listing I.29 Vibrating in a pattern

```

navigator.mozVibrate([100,
                    100,
                    200,
                    200]);

```

← Vibrations.

Pauses. →

### I.3.6 Battery API: adjusting application processing based on battery life

The Battery API allows you to adjust how much processing your app does depending on the state of the battery. In a real app, you could avoid doing any heavy processing or reduce the number of network connections when the battery is low. In our example app, there isn't much opportunity to cut back processing, so you're just going to draw less of Wilson as the battery level drops. Figure I.16 shows the end result in Firefox on an Android phone.



**Figure I.16** By integrating the Battery API, you can make your app do less work as the charge level drops.

The Battery API consists of four properties and four events. See the summary in table I.7.

**Table I.7** The Battery API

Property/event name	Type	Description
charging	Read-only boolean	Is the power connected?
chargingTime	Read-only double	Seconds remaining until the battery is charged.

Table I.7 The Battery API (continued)

Property/event name	Type	Description
dischargingTime	Read-only double	Seconds remaining until the battery is discharged.
level	Read-only double	A value between 0.0 and 1.0 representing the current battery charge level, where 1.0 is full.
chargingchange	Event	The value of charging has changed.
chargingtimechange	Event	The chargingTime has changed.
dischargingtimechange	Event	The dischargingTime has changed.
levelchange	Event	The level has changed.

In this example you're just going to take advantage of the charging and level properties. The following table shows the browser compatibility; this API will work on mobile devices but also laptops.

					
<b>Battery API</b>	20	10	N/A	N/A	N/A

For this example, you can either continue working with the code from the previous section or, if you don't have access to a mobile phone, you can use the code from section I.3.3 as a starting point. The changes required to the draw() function are shown next.

Listing I.30 Using the battery object in the draw function

```

draw: function (canvas, battery) {
  var tl_x = wilson.x - 70;
  var tl_y = wilson.y - 70;
  if (canvas.getContext){
    var context = canvas.getContext('2d');
    context.beginPath();
    context.arc(tl_x + 70, tl_y + 70,
      70, 0, 2 * Math.PI, false);
    context.fillStyle = 'yellow';
    context.fill();
    if (battery.charging
      || (!battery.charging
        && (battery.level > 0.5))) {
      context.beginPath();
      //...
    }
  }
}

```

This code is the same as before and has been elided for brevity.

The battery object is passed into the draw function so the browser-compatibility code can be all in one place.

Always draw the yellow circle.

If the battery is charging...

...or the battery isn't charging and...

...the battery charge level is above 50%, then draw the eyes and mouth.

```

    if (battery.charging) {
        context.fillStyle = 'black';
        //...
    }
}

```

← Draw only the glasses if the battery is charging.

← This code is the same as before and has been elided for brevity.

As the annotation mentions, the `battery` object needs to be passed in, which necessitates a small change in the `go()` function. The next listing shows the code for getting a reference to the battery status and passing it to `wilson.draw()`.

#### Listing I.31 Passing the battery object to the draw() function

```

var battery = navigator.battery ||
    navigator.mozBattery ||
    navigator.webkitBattery;
wilson.draw(canvas, battery);

```

That's enough mobile excitement for now; in the next section you're going back to the desktop and the Pointer Lock API, a necessary component of most 3D games.

### I.3.7 The Pointer Lock API: tracking mouse motion instead of pointer position

Pointer lock may sound like it's another way of doing `setCapture`, but it's targeted at a different use case. Whereas `setCapture` allows you to continue tracking the mouse pointer position even when it moves outside the target element, pointer lock takes the pointer position out of the equation entirely. Instead of tracking the position of the mouse pointer, it tracks motion from the mouse itself. The difference is that the pointer position is limited by the bounds of the screen; the mouse can carry on moving. This is crucially important in immersive games like first-person shooters, where the mouse is used to orient the player. Figure I.17 shows an example taken from <http://media.tojicode.com/q3bsp/>; note that the mouse pointer doesn't even appear.

The Pointer Lock API involves only a few properties, methods, and events. A summary is shown in table I.8.

**Table I.8 The Pointer Lock API**

Property/event name	Type	Description
<code>requestPointerLock()</code>	Method	Ask for the pointer to be locked.
<code>pointerLockElement</code>	Read-only element	If the pointer is locked, this property will be set to the element that requested it.
<code>pointerlockchange</code>	Event	The pointer lock status has changed.
<code>pointerlockerror</code>	Event	There was an error requesting pointer lock.



**Figure I.17** Pointer lock in action along with WebGL; note that the mouse pointer isn't visible.

The Pointer Lock API has experimental implementations in Chrome (with the `--enable-pointer-lock` command-line switch) and Firefox.

					
<b>Pointer Lock API</b>	18	14	N/A	N/A	N/A

To experiment with the Pointer Lock API, you're going to need a world to explore. Although ideally you'd create your own 3D world, that would take quite some time (please refer to chapter 9 if you'd like to give it a go). In the meantime, you can fake a world with a panoramic photograph. A suitable large image is included in the code download. The following listing shows where you can add the image.

#### Listing I.32 Add a background image to canvas

```
<canvas id="canvas" onclick="go()">
  
</canvas>
```

You'll take this image and add it as a background to the `<canvas>` element. Because the image is 9073 pixels wide, it should stretch across more than a single screen on all but the largest of displays. Figure I.18 shows the initial screen in Firefox 14.

The first requirement is a function to draw a correctly scaled slice of the image on the canvas, as shown in listing I.33.



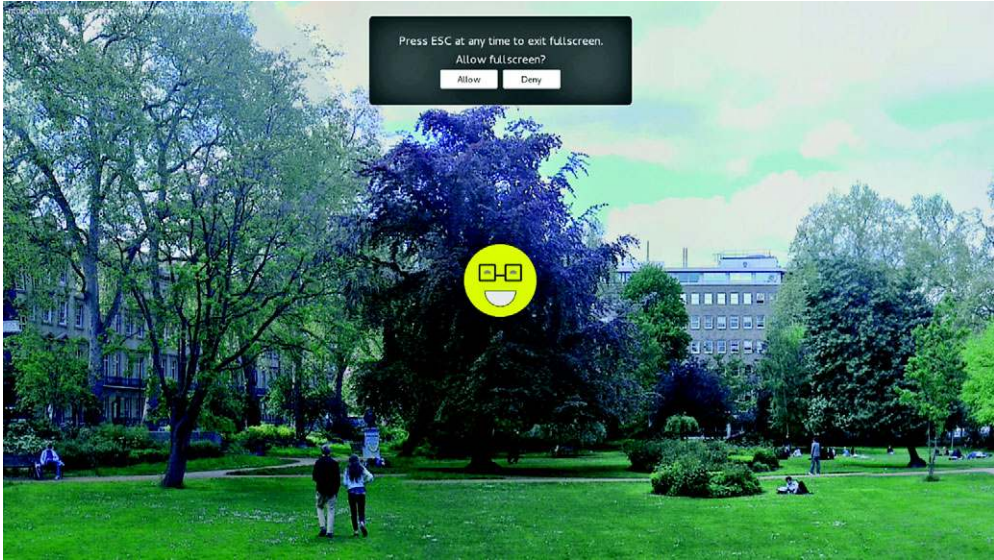


Figure I.18 Wilson exploring a London park

### Listing I.33 Draw a segment of the background image

```
function draw_background(canvas, image, x_offset) {
  var scale = canvas.height / image.height;
  var x = x_offset * scale;
  var slice = canvas.width / scale;
  var ctx = canvas.getContext('2d');
  ctx.drawImage(image,
    x, 0, slice, image.height,
    0, 0, canvas.width, canvas.height);
}
```

Calculate a scaling factor to match the image to the canvas height.

Use the scaling factor to convert the offset into a screen length.

Use the scaling factor to convert the screen width into an image offset so you can grab a correctly scaled slice of the image.

The next listing shows the code to activate the Pointer Lock API. This code should go at the top of the `go()` function.

### Listing I.34 Request pointer lock when the mode changes to full screen

```
canvas.requestPointerLock = canvas.requestPointerLock ||
  canvas.mozRequestPointerLock ||
  canvas.webkitRequestPointerLock;

function on_full_screen() {
  canvas.requestPointerLock();
  follow_mouse();
}

document.addEventListener("fullscreenchange",
  on_full_screen, false);
document.addEventListener("mozfullscreenchange",
  on_full_screen, false);
```

Map the different custom implementations to a single function.

The request for a pointer lock must be made inside a fullscreenchange event.

```

document.addEventListener("webkitfullscreenchange",
                        on_full_screen, false);

function pointer_lock_change() {
    if (document.pointerLockElement === canvas ||
        document.mozPointerLockElement === canvas ||
        document.webkitPointerLockElement === canvas) {
        console.log("Pointer Lock was successful.");
    } else {
        console.log("Pointer Lock was lost.");
    }
}

document.addEventListener("pointerlockchange",
                        pointer_lock_change, false);
document.addEventListener("mozpointerlockchange",
                        pointer_lock_change, false);
document.addEventListener("webkitpointerlockchange",
                        pointer_lock_change, false);

function pointer_lock_error() {
    console.log("Error while locking pointer.");
}

document.addEventListener("pointerlockerror",
                        pointer_lock_error, false);
document.addEventListener("mozpointerlockerror",
                        pointer_lock_error, false);
document.addEventListener("webkitpointerlockerror",
                        pointer_lock_error, false);

```

The `<pointerlockchange>` event will be fired when the request is made; you can test for success of the request by checking the `document.pointerLockElement`.

The `<pointerlockerror>` event will let you investigate any errors that may occur.

Next, you need to update the `follow_mouse()` function again, as shown in the following listing. The Pointer Lock API adds two additional properties to a mouse event: `movementX` and `movementY`. These can be used in a similar way to the orientation events in section I.35.

#### Listing I.35 Follow the mouse movement

```

function follow_mouse() {
    document.addEventListener("mousemove", function(e) {
        wilson.v_x = e.movementX ||
                    e.mozMovementX ||
                    e.webkitMovementX ||
                    0;

        wilson.v_y = e.movementY ||
                    e.mozMovementY ||
                    e.webkitMovementY ||
                    0;

        offset += wilson.v_x;
    }, false);
};

```

## I.4 Summary

In this appendix you've had a glimpse of the future of HTML5. A lot of effort is directed toward accessing device capabilities (webcams, microphones, orientation

sensors, and so on) as well as toward building seamless gaming and application experiences (full-screen and pointer lock) to rival native applications. As these standards are finalized and implementations mature over the next few years, we should see a lot of exciting new web applications. Now that you've read this appendix (and the rest of this book), you should be well equipped to take an active role in developing the World Wide Web of tomorrow!

# appendix J

## Links and references

---

In this appendix, you'll find a chapter-by-chapter list of many of the links to useful resources, articles, and demos strewn throughout *HTML5 in Action*. Links for each chapter start with important applications and references for building your apps. Near the bottom of each link list, you may also find interesting tidbits such as fun links, live demos, and extra libraries for future projects.

### Chapter 1: Introduction

- *Modernizr*—<http://modernizr.com/>
- *Remy Sharp's HTML5 Shiv (included in Modernizr)*—<http://remysharp.com/2009/01/07/html5-enabling-script/>
- *WHATWG*—[www.whatwg.org/](http://www.whatwg.org/)
- *Hello! HTML5 and CSS3*—[www.manning.com/crowther/](http://www.manning.com/crowther/)
- *ARIA Attributes*—<http://mng.bz/6hb2>
- *Google's Microdata Vocabulary*—<http://schema.org/>
- *Is This HTML5?*— <http://mng.bz/PraC>

### Chapter 2: Forms and validation

- *Webshims Lib*—<http://afarkas.github.com/webshim/demos/>
- *H5F*—<https://github.com/ryanseddon/H5F>
- *Webforms2*—<https://github.com/westonruter/webforms2>
- *html5Widgets*—<https://github.com/zoltan-dulac/html5Forms.js>
- *Modernizr Polyfills*—<http://mng.bz/cJhc>

### Chapter 3: Working with files on the client side

- *File API*—[www.w3.org/TR/FileAPI/](http://www.w3.org/TR/FileAPI/)
- *File Writer API*—[www.w3.org/TR/file-writer-api/](http://www.w3.org/TR/file-writer-api/)

- *File System API*—[www.w3.org/TR/file-system-api/](http://www.w3.org/TR/file-system-api/)
- *Geolocation API*—[www.w3.org/TR/geolocation-API/](http://www.w3.org/TR/geolocation-API/)

## Chapter 4: Messaging

- *Apache*—<http://apache.org/>
- *PHP*—<http://php.net/>
- *MySQL*—<http://dev.mysql.com/>
- *jQuery*—<http://jquery.com/>
- *Node.js*—<http://nodejs.org/>
- *Connect*—<https://github.com/senchalabs/connect>
- *Mustache*—<http://mustache.github.com>
- *WebSocket-Node*—<https://github.com/Worlize/WebSocket-Node>
- *EventEmitter.js*—<https://github.com/Wolfy87/EventEmitter>
- *Polyfills EventSource.js*—<http://mng.bz/ahX0>

## Chapter 5: Web storage and working offline

- *Offline API (in HTML5 spec)*—<http://mng.bz/5u67>
- *IndexedDB*—[www.w3.org/TR/IndexedDB/](http://www.w3.org/TR/IndexedDB/)
- *Web SQL (deprecated)*—[www.w3.org/TR/webdatabase/](http://www.w3.org/TR/webdatabase/)

## Chapter 6: 2D Canvas

- *HTML5 Canvas Cheat Sheet*—<http://mng.bz/5r65>.
- *explorercanvas*—<http://code.google.com/p/explorercanvas/>
- *Game Physics guide*—<http://gafferongames.com/game-physics/>
- *playtomic*—<https://playtomic.com/>
- *MDN window.requestAnimationFrame*—<http://mng.bz/D14s>
- *requestAnimationFrame for polyfills*—<http://mng.bz/h9v9>
- *JavaScript Madness: Keyboard Events*—<http://unixpapa.com/js/key.html>
- *Sketchpad*—<http://mudcu.be/sketchpad/>
- *Rome: 3 Dreams of Black*—<http://ro.me>
- *ImpactJS*—<http://impactjs.com/>

## Chapter 7: SVG

- *Official SVG page*—[www.w3.org/Graphics/SVG/](http://www.w3.org/Graphics/SVG/)
- *W3C SVG animation*—[www.w3.org/TR/SVG11/animate.html](http://www.w3.org/TR/SVG11/animate.html)
- *Canceling animation requests*—<http://mng.bz/3Eq1>
- *Raphael.js*—[http://raphaeljs.com/](http://dmitrybaranovskiy.github.io/raphael/)
- *Svgweb*—<http://code.google.com/p/svgweb/>
- *SVG a element*—<http://tutorials.jenkov.com/svg/a-element.html>
- *svg-edit*—<https://code.google.com/p/svg-edit/>

## Chapter 8: Video and audio

- *FFmpeg*—<http://ffmpeg.org>
- *FFmpeg Mac Version*—<http://ffmpegmac.net/>
- *FFmpeg2theora*—<http://v2v.cc/~j/ffmpeg2theora/>
- *Image Filters with Canvas*—<http://mng.bz/3OsN>
- *Playback Rate Bug*—[https://bugzilla.mozilla.org/show\\_bug.cgi?id=495040](https://bugzilla.mozilla.org/show_bug.cgi?id=495040)

## Chapter 9: WebGL

- *WebGL Cheat Sheet*—<http://blog.nihilogic.dk/2009/10/webgl-cheat-sheet.html>
- *OpenGL ES Shading Language Reference*—<http://mng.bz/1TA3>
- *Introduction to 3D graphics*—<http://mng.bz/STHc>
- *Simple JavaScript Inheritance*—<http://ejohn.org/blog/simple-javascript-inheritance/>
- *Sylvester*—<http://sylvesterjcgolan.com/>
- *Wolfram Identity Matrix explanation*—<http://mathworld.wolfram.com/IdentityMatrix.html>
- *Opera's Introduction to WebGL*—<http://mng.bz/4Lao>
- *MDN 2D WebGL content and WebGL utilities file*—<http://mng.bz/2585>
- *MDN WebGL rotation*—<http://mng.bz/O5Z2>
- *MDN WebGL tutorials*—<https://developer.mozilla.org/en/WebGL>
- *Joe Lambert's Request polyfill*—<http://mng.bz/3epb>
- *Learning WebGL*—<http://learningwebgl.com>
- *three.js*—<https://github.com/mrdoob/three.js/>
- *Copperlicht*—[www.ambiera.com/copperlicht/](http://www.ambiera.com/copperlicht/)
- *IEWebGL*—<http://iewebgl.com/>
- *Secrets of the JavaScript Ninja*—[www.manning.com/resig/](http://www.manning.com/resig/)
- *X-Wing WebGL app*—[http://oos.moxiecode.com/js\\_webgl/xwing/](http://oos.moxiecode.com/js_webgl/xwing/)
- *Vlad Vukićević's blog*—<http://blog.vlad1.com>