



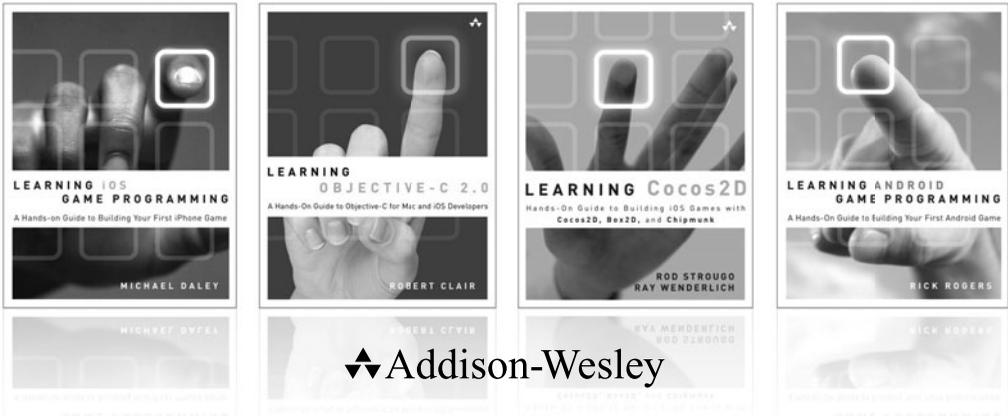
**LEARNING HTML5
GAME PROGRAMMING**

A Hands-on Guide to Building Online Games Using Canvas, SVG, and WebGL

JAMES L. WILLIAMS

Learning HTML5 Game Programming

Addison-Wesley Learning Series



Visit informit.com/learningseries for a complete list of available publications.

The **Addison-Wesley Learning Series** is a collection of hands-on programming guides that help you quickly learn a new technology or language so you can apply what you've learned right away.

Each title comes with sample code for the application or applications built in the text. This code is fully annotated and can be reused in your own projects with no strings attached. Many chapters end with a series of exercises to encourage you to reexamine what you have just learned, and to tweak or adjust the code as a way of learning.

Titles in this series take a simple approach: they get you going right away and leave you with the ability to walk off and build your own application and apply the language or technology to whatever you are working on.

◆ Addison-Wesley

informIT.com

Safari[®]
Books Online

Learning HTML5 Game Programming

A Hands-on Guide to Building Online
Games Using Canvas, SVG, and WebGL

James L. Williams

◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data:

Williams, James L. (James Lamar), 1981-

Learning HTML5 game programming : a hands-on guide to building online games using Canvas, SVG, and WebGL / James L. Williams.
p. cm.

ISBN 978-0-321-76736-3 (pbk. : alk. paper) 1. Computer games—Programming. 2. HTML (Document markup language) I. Title.
QA76.76.C672W546 2011
794.8'1526—dc23

2011027527

Copyright © 2012 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671-3447

ISBN-13: 978-0-321-76736-3

ISBN-10: 0-321-76736-5

Text printed in the United States on recycled paper at RR Donnelly in Crawfordsville, Indiana.

First printing September 2011

Associate

Publisher

Mark Taub

Senior Acquisitions

Editor

Trina MacDonald

Development

Editor

Songlin Qiu

Managing Editor

Kristy Hart

Project Editor

Anne Goebel

Copy Editor

Bart Reed

Indexer

Tim Wright

Proofreader

Sheri Cain

Technical

Reviewers

Romin Irani

Pascal Rettig

Robert Schwentker

Publishing

Coordinator

Olivia Basegio

Cover Designer

Chuti Prasertsith

Senior Compositor

Gloria Schurick



To Inspiration

Came over for a midnight rendezvous

And is gone by morning as if by cue

—Author



Table of Contents

Chapter 1 Introducing HTML5 1

Beyond Basic HTML	1
JavaScript	1
AJAX	2
Bridging the Divide	2
Google Gears	3
Chrome Frame	3
Getting Things Done with WebSockets and Web Workers	4
WebSockets	4
Web Workers	4
Application Cache	5
Database API	6
WebSQL API	6
IndexedDB API	7
Web Storage	7
Geolocation	8
Getting Users' Attention with Notifications	10
Requesting Permission to Display Notifications	11
Creating Notifications	11
Interacting with Notifications	12
Media Elements	13
Controlling Media	13
Handling Unsupported Formats	14
HTML5 Drawing APIs	15
Canvas	15
SVG	16
WebGL	16
Conveying Information with Microdata	16

Chapter 2 Setting Up Your Development Environment 19

Development Tools	19
Installing Java	19

Installing the Eclipse IDE and Google Plugin	20	
Google Web Toolkit	22	
Web Server Tools and Options	23	
Google App Engine	23	
Opera Unite	23	
Node.js and RingoJS	23	
Browser Tools	24	
Inside the Chrome Developer Tools	24	
Chrome Extensions	25	
Safari Developer Tools	26	
Firebug	26	
HTML5 Tools	27	
ProcessingJS	27	
Inkscape	27	
SVG-edit	27	
Raphaël	28	
3D Modeling Tools	29	
Blender	29	
Chapter 3	Learning JavaScript	31
What Is JavaScript?	31	
JavaScript's Basic Types	31	
Understanding Arithmetic Operators	32	
Understanding JavaScript Functions	32	
Functions as First-class Objects	33	
Comparison Operators	34	
Conditional Loops and Statements	35	
Controlling Program Flow with Loops	36	
Delayed Execution with setTimeout and setInterval	38	
Creating Complex Objects with Inheritance and Polymorphism	38	
Making Inheritance Easier with the Prototype Library	39	
Learning JQuery	41	
Manipulating the DOM with Selectors	42	
JQuery Events	43	
AJAX with JQuery	43	
Cross-Site Scripting	44	

JSON: The Other JavaScript Format	44
JavaScript Outside of the Browser	45
Mobile Platforms	45
JavaScript as an Intermediary Language	45
JavaScript on the Desktop	46
Server-Side JavaScript	48

Chapter 4 How Games Work 51

Designing a Game	51
Writing a Basic Design Document	51
Deciding on a Game Genre	52
The Game Loop	53
Getting Input from the User	53
Representing Game Objects with Advanced Data Structures	54
Making Unique Lists of Data with Sets	54
Creating Object Graphs with Linked Lists	56
Understanding the APIs in Simple Game Framework	57
Core API	57
Components API	58
Resources API and Networking APIs	58
Building <i>Pong</i> with the Simple Game Framework	59
Setting Up the Application	59
Drawing the Game Pieces	61
Making Worlds Collide with Collision Detection and Response	63
Understanding Newton's Three Laws	63
Making the Ball Move	64
Advanced Collision Detection and Particle Systems with Asteroids	66
Creating Competitive Opponents with Artificial Intelligence	67
Adding AI to Pong	68
Advanced Computer AI with Tic-Tac-Toe	68

Chapter 5 Creating Games with the Canvas Tag 71

Getting Started with the Canvas	71
Drawing Your First Paths	72
Drawing Game Sprites for Tic-Tac-Toe	73

Drawing Objects on the Canvas with Transformations	75
Ordering Your Transformations	76
Saving and Restoring the Canvas Drawing State	77
Using Images with the Canvas	78
Serving Images with Data URLs	78
Serving Images with Spritesheets	78
Drawing Images on the Canvas	78
Animating Objects with Trident.js	79
Creating Timelines	80
Animating with Keyframes	81
Creating Nonlinear Timelines with Easing	81
Animating Game Objects with Spritesheets	83
Simulating 3D in 2D Space	84
Perspective Projection	84
Parallaxing	85
Creating a Parallax Effect with JavaScript	85
Creating <i>Copy Me</i>	87
Drawing Our Game Objects	87
Making the Game Tones	88
Playing MIDI Files in the Browser	89
Playing Multiple Sounds at Once	90
Playing Sounds Sequentially	91
Drawing Our Game Text	91
Styling Text with CSS Fonts	92

Chapter 6 Creating Games with SVG and RaphaëlJS 95

Introduction to SVG	95
First Steps with RaphaëlJS	97
Setting Up Our Development Environment	97
Drawing the Game Board	98
Drawing Game Text	99
Custom Fonts	100
Specifying Color	103
Loading Game Assets	104
Converting SVG Files to Bitmap Images	105

Creating Our Game Classes	105
Shuffling Cards	107
Drawing and Animating Cards	107
Creating Advanced Animations	110
Paths	110
moveto and lineto	110
curveto	111
Exporting Paths from an SVG File	112
Animating Along Paths	113
Extending Raphaël with Plugins	113
Adding Functions	113
SVG Filters	113
Speed Considerations	114

Chapter 7 Creating Games with WebGL and Three.js 117

Moving to Three Dimensions	118
Giving Your Objects Some Swagger with Materials and Lighting	119
Understanding Lighting	120
Using Materials and Shaders	120
Creating Your First Three.js Scene	122
Setting Up the View	123
Viewing the World	128
Loading 3D Models with Three.js	129
Programming Shaders and Textures	131
Using Textures	134
Creating a Game with Three.js	136
Simulating the Real World with Game Physics	137
Revisiting Particle Systems	140
Creating Scenes	141
Selecting Objects in a Scene	142
Animating Models	142
Sourcing 3D Models	143
Benchmarking Your Games	144
Checking Frame Rate with Stats.js	144
Using the WebGL Inspector	145

Chapter 8 Creating Games Without JavaScript 147

- Google Web Toolkit 147
 - Understanding GWT Widgets and Layout 148
 - Exposing JavaScript Libraries to GWT with JSNI 149
 - RaphaëlGWT 150
 - Adding Sound with gwt-html5-media 151
 - Accessing the Drawing APIs with GWT 151
- CoffeeScript 153
 - Installing CoffeeScript 153
 - Compiling CoffeeScript Files 153
- A Quick Guide to CoffeeScript 154
 - Basics 154
 - Functions and Invocation 154
 - Aliases, Conditionals, and Loops 156
 - Enhanced for Loop and Maps 156
 - Classes and Inheritance 157
- Alternate Technologies 158
 - Cappuccino 158
 - Pyjamas 158

Chapter 9 Building a Multiplayer Game Server 161

- Introduction to Node.js 161
 - Extending Node with the Node Package Manager 162
 - Managing Multiple Node Versions 162
- Making Web Apps Simpler with ExpressJS 163
 - Serving Requests with URL Routing 163
 - Managing Sessions 165
 - Understanding the ExpressJS Application Structure 165
 - Templating HTML with CoffeeKup 166
- Persisting Data with Caching 168
- Managing Client/Server Communication 169
 - Communicating with Socket.IO 169
 - Setting Up a Simple Socket.IO Application with Express 170
 - Making Web Sockets Simpler with NowJS 171
- Debugging Node Applications 172

Creating a Game Server	173
Making the Game Lobby	173
Creating Game Rooms with NowJS Groups	174
Managing Game Participants and Moving Between Game Rooms	175
Managing Game Play	175

Chapter 10 Developing Mobile Games 179

Choosing a Mobile Platform	179
iOS	179
Android	180
WebOS	180
Windows Phone 7	180
Flick, Tap, and Swipe: A Quick Guide to Mobile Gestures	181
Deciding Between an Application and a Website	181
Storing Data on Mobile Devices	183
Relaxing in Your Lawnchair: An Easier Way to Store Data	183
Getting Started with Lawnchair	184
Client-Side Scripting Simplified with JQuery and Zepto	185
Using JQuery Variants	185
Using Zepto.js	187
Architecting Your Applications with JoApp	187
Choosing an Application Framework	188
PhoneGap	188
Diving into the PhoneGap APIs	189
Appcelerator Titanium	191
Diving into the Appcelerator Titanium APIs	191
Packaging Android Applications with Titanium and PhoneGap	191
Packaging an Application with Titanium	193
Packaging an Application with PhoneGap	195

Chapter 11	Publishing Your Games	199
Optimizing Your Game's Assets		199
Minification with Google Closure Compiler		199
Running Applications Offline with Application Cache		201
Hosting Your Own Server		203
Deploying Applications on Hosted Node.js Services		204
Publishing Applications on the Chrome Web Store		205
Describing Your Application's Metadata		206
Deploying a Hosted Application		207
Deploying a Packaged Application		208
Testing Your Applications Locally		208
Uploading Your Application to the Chrome Web Store		208
Configuring Your Application		210
Deciding Between Packaged and Hosted Chrome Apps		212
Publishing Applications with TapJS		212
Creating a TapJS Application		213
Packaging an Application for TapJS		215
Publishing a TapJS Application to Facebook		215
Publishing Games with Kongregate		217
Publishing HTML5 Applications to the Desktop		217
Index		219

Preface

I wrote this book to scratch an itch, but also because I could see the potential in the (at the time) nascent HTML5 gaming community. I wanted to help developers navigate the wilderness of HTML5 and learn about Canvas, WebGL, and SVG, along with best practices for each.

It sometimes took a bit of discussion to convince developers that HTML5 wasn't just a plaything. They were surprised to learn they could have rich content with all the niceties of a desktop application—such as double buffering, hardware acceleration, and caching inside the confines of the browser without a plugin. Many of them considered Flash as the sole option. It was interesting to watch the tides turn from “Flash for everything” to “Use Flash only where there are HTML5 gaps.”

During my writing of this book, the ecosystem around HTML5 game programming has rapidly evolved and matured. I am sure the technologies will continue to evolve, and I look forward to the advances the next year brings.

Key Features of This Book

This book covers areas contained in the “loose” definition of HTML5, meaning the HTML5 specification, WebGL, SVG, and JavaScript as they pertain to game programming. It includes sections on the math behind popular game effects, teaching you the hard way before providing the one to two lines of code solution. For those who are still getting accustomed to JavaScript, there is a chapter on alternative languages that can be used to produce games. These include languages that run directly in the JavaScript engine, those that compile to JavaScript, or those that are a combination of the two. Server-side JavaScript has taken the programming world by storm in recent months. For games, it presents an extra level of flexibility to structure games. Logic can start in a self-contained client instance and then progress to a scalable server instance with few changes in code. The book closes with a discussion of how and where you might publish your games. You have a multitude of choices for game engines and libraries. All the libraries used in this book are unobtrusive in their handling of data, and you could easily take the lessons learned and apply them to other libraries. This book does not discuss the low-level details of WebGL, instead opting for the use of a high-level library that permits low-level API access when needed. The goal of this book is to get you quickly up and running, not to teach you all there is to know about WebGL, which could be a book all by itself.

Target Audience for This Book

This book is intended for application developers who use or would like to learn how to use HTML5 and associated web technologies to create interactive games. It assumes knowledge of some programming languages and some basic math skills.

Code Examples and Exercises for This Book

The code listings as well as the answers for the exercises included in this book are available on the book's website. You can download chapter code and answers to the chapter exercises (if they are included in the chapter) at <http://www.informit.com/title/9780321767363>. The code listings are also available on Github at <https://github.com/jwill/html5-game-book>.

Acknowledgments

I have several people to thank for this book. The Pearson team (including Trina MacDonald, Songlin Qiu, and Olivia Basegio) has been invaluable during the project. Their goal is to make one's work that much more awesome, and I think they succeeded. Writing a book on a topic that's evolving rapidly involves a certain measure of guessing where the market will go. I'm glad to have had technical reviewers (Romin Irani, Pascal Rettig, and Robert Schwenker) who shared my passion for the subject matter, gave me speedy and precise feedback, and validated my predictions when I was right, yet got me back on track when I veered slightly off course. And lastly, to my family and friends who listened patiently without judgment, let me off easy when I flaked, and other times forced me to take a break; thanks, I needed that.

About the Author

James L. Williams is a developer based in Silicon Valley and frequent conference speaker, domestically and internationally. He was a successful participant in the 2007 Google Summer of Code, working to bring easy access to SwingLabs UI components to Groovy. He is a co-creator of the Griffon project, a rich desktop framework for Java applications. He and his team, WalkIN, created a product on a coach bus while riding to SXSW and were crowned winners of StartupBus 2011. His first video game was *Buck Rogers: Planet of Zoom* on the Coleco Adam, a beast of a machine with a blistering 3.58MHz CPU, a high-speed tape drive, and a propensity to erase floppy disks at bootup. He blogs at <http://jameswilliams.be/blog> and tweets as @ecspike.

This page intentionally left blank

Introducing HTML5

HTML5 is a draft specification for the next major iteration of HTML. It represents a break from its predecessors, HTML4 and XHTML. Some elements have been removed and it is no longer based on SGML, an older standard for document markup. HTML5 also has more allowances for incorrect syntax than were present in HTML4. It has rules for parsing to allow different browsers to display the same incorrectly formatted document in the same fashion. There are many notable additions to HTML, such as native drawing support and audiovisual elements. In this chapter, we discuss the features added by HTML5 and the associated JavaScript APIs.

Beyond Basic HTML

HTML (Hypertext Markup Language), invented by Tim Berners-Lee, has come a long way since its inception in 1990. Figure 1-1 shows an abbreviated timeline of HTML from the HTML5Rocks slides (<http://slides.html5rocks.com/#slide3>).

Although all the advancements were critical in pushing standards forward, of particular interest to our pursuits is the introduction of JavaScript in 1996 and AJAX in 2005. Those additions transformed the Web from a medium that presented static unidirectional data, like a newspaper or book, to a bidirectional medium allowing communication in both directions.

JavaScript

JavaScript (née LiveScript and formally known as ECMAScript) started as a scripting language for the browser from Netscape Communications. It is a loosely typed scripting language that is prototype-based and can be object-oriented or functional. Despite the name, JavaScript is most similar to the C programming language, although it does inherit some aspects from Java.

The language was renamed JavaScript as part of a marketing agreement between Sun Microsystems (now Oracle Corporation) and Netscape to promote the scripting language alongside Sun's Java applet technology. It became widely used for scripting client-side

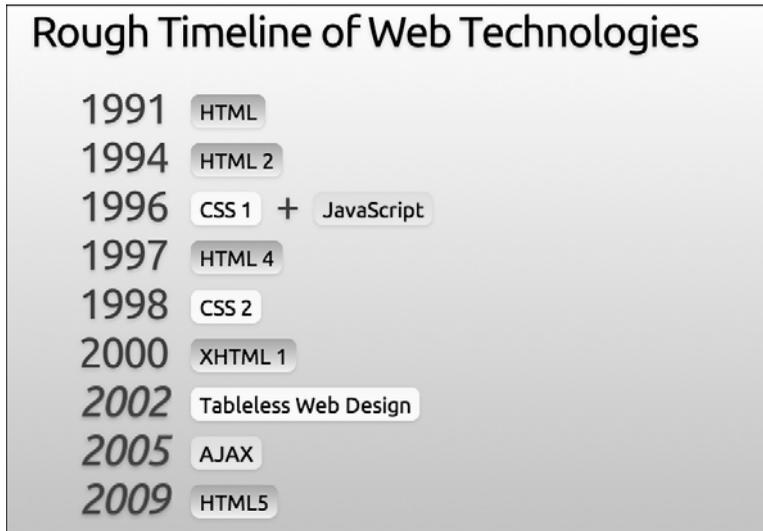


Figure 1-1 HTML timeline

web pages, and Microsoft released a compatible version named JScript, with some additions and changes, because Sun held the trademark on the name “JavaScript.”

AJAX

AJAX (Asynchronous JavaScript and XML) started a new wave of interest in JavaScript programming. Once regarded as a toy for amateurs and script kiddies, AJAX helped developers solve more complex problems.

At the epicenter of AJAX is the `XMLHttpRequest` object invented by Microsoft in the late 1990s. `XMLHttpRequest` allows a website to connect to a remote server and receive structured data. As opposed to creating a set of static pages, a developer was empowered to create highly dynamic applications. Gmail, Twitter, and Facebook are examples of these types of applications.

We are currently in the midst of another JavaScript renaissance, as the major browser makers have been using the speed of their JavaScript engines as a benchmark for comparison. JavaScript as a primary programming language has found its way into server-side web components, such as Node.js, and mobile application frameworks, such as WebOS and PhoneGap.

Bridging the Divide

Even the best of standards takes a while to gain uptake. As a means to not let the lack of features limit innovation, Google created Chrome Frame and Google Gears (later, simply Gears) to bring advanced features to older browsers.

Google Gears

Google Gears, which was initially released in May 2007, has come to define some of the advanced features of the HTML5 draft specification. Before the advent of HTML5, many applications used Gears in some way, including Google properties (Gmail, YouTube, Doc, Reader, and so on), MySpace, Remember the Milk, and WordPress, among others. Gears is composed of several modules that add functionality more typical of desktop applications to the browser. Let's take a moment and talk about some of its features.

In its first release, Gears introduced the Database, LocalServer, and WorkerPool modules. Gears' Database API uses an SQLite-like syntax to create relational data storage for web applications. The data is localized to the specific application and complies with generalized cross-site scripting rules in that an application cannot access data outside its domain. The LocalServer module enables web applications to save and retrieve assets to a local cache even if an Internet connection is not present. The assets to serve from local cache are specified in a site manifest file. When an asset matching a URL in the manifest file is requested, the LocalServer module intercepts the request and serves it from the local store.

The WorkerPool module helps address one of the prevalent problems with JavaScript-intensive websites: long-running scripts that block website interaction. A website by default has a single thread to do its work. This is generally not a problem for very short, bursty actions (such as simple DOM manipulation) that return quickly. Any long-running task, such as file input/output or trying to retrieve assets from a slow server, can block interaction and convince the browser that the script is unresponsive and should be forcefully ended. The WorkerPool module brought the concept of multithreading computing to the browser by letting your WorkerPool create "workers" that can execute arbitrary JavaScript. Workers can send and receive messages to and from each other, provided they are in the same WorkerPool, so they can cooperate on tasks. Workers can work cross-origin but inherit the policy from where they are retrieved. To account for the fact that several properties such as `Timer` and `HttpRequest` are exposed by the `window` object, which is not accessible to workers, Gears provides its own implementations.

Another API of interest is the Geolocation API. The Geolocation API attempts to get a fix on a visitor by using available data such as the IP address, available Wi-Fi routers with a known location, cell towers, and other associated data.

Google ceased principal development of Gears in November 2009 and has since shifted focus to getting the features into HTML5. Thankfully, all these features we've discussed found their way into HTML5 in some shape or form.

Chrome Frame

Chrome Frame is a project that embeds Google Chrome as a plugin for Internet Explorer 6 and higher versions, which have weak HTML5 support. Chrome Frame is activated upon recognition of a meta tag. Chrome Frame currently does not require admin rights to be installed, thus opening opportunities on systems that are otherwise locked down.

You can find more information about Chrome Frame at <http://code.google.com/chrome/chromeframe/>.

Getting Things Done with WebSockets and Web Workers

One of the additions to HTML5 is APIs that help the web application communicate and do work. WebSockets allow web applications to open a channel to interact with web services. Web Workers permit them to run nontrivial tasks without locking the browser.

WebSockets

WebSockets allow applications to have a bidirectional channel to a URI endpoint. Sockets can send and receive messages and respond to opening or closing a WebSocket. Although not part of the specification, two-way communication can be achieved in several other ways, including Comet (AJAX with long polling), Bayeux, and BOSH.

Listing 1-1 shows the code to create a WebSocket that talks to the echo server endpoint. After creating the socket, we set up the functions to be executed when the socket is opened, closed, receives a message, or throws an error. Next, a “Hello World!” message is sent, and the browser displays “Hello World!” upon receipt of the return message.

Listing 1-1 **WebSocket Code for Echoing a Message**

```
var socket = new WebSocket(ws://websites.org:8787/echo);
socket.onopen = function(evt) { console.log("Socket opened"); };
socket.onclose = function(evt) { console.log("Socket closed"); };
socket.onmessage = function(evt) { console.log(evt.data); };
socket.onerror = function(evt) { console.log("Error: "+evt.data); };

socket.send("Hello World!");
```

Web Workers

Web Workers are the HTML5 incarnation of WorkerPools in Google Gears. Unlike WorkerPools, we don't have to create a pool to house our Web Workers. Listing 1-2 shows the code to create a simple worker and set a function for it to execute upon receipt of a message. Listings 1-2 and 1-3 show the HTML code for creating a web page with a Web Worker that displays the current date and time on two-second intervals.

Listing 1-2 **Web Page for Requesting the Time**

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Web Worker example</title>
```

```
</head>
<body>
  <p>The time is now: <span id="result" /></p>
  <script>
    var worker = new Worker('worker.js');
    worker.onmessage = function (event) {
      document.getElementById('result').innerText = event.data;
    };
  </script>
</body>
</html>
```

The associated JavaScript worker.js file is shown in Listing 1-3.

Listing 1-3 Worker.js File for Getting a Date and Time

```
setInterval(function() {w
  postMessage(new Date());
}, 2000);
```

In the two listings, we see that workers can send messages using `postMessage()` and can listen for messages on the closure `onmessage`. We can also respond to errors and terminate workers by passing a function to `onerror` and executing `terminate()`, respectively.

Workers can be shared and send messages on `MessagePorts`. As with other aspects of the Web Worker spec, this portion is in a state of flux and somewhat outside the needs of the examples in this book. Therefore, using `SharedWorkers` is left as an exercise for the reader to investigate.

Application Cache

Application Cache provides a method of running applications while offline, much like the `LocalServer` feature in Gears. A point of distinction between the two features is that Application Cache doesn't use a JSON file, using a flat file instead to specify which files to cache. A simple manifest file to cache assets is shown in Listing 1-4.

Listing 1-4 Sample Application Manifest

```
CACHE MANIFEST
# above line is required, this line is a comment
mygame/game.html
mygame/images/image1.png
mygame/assets/sound2.ogg
```

The Application Cache has several events it can respond to: `onchecking`, `error`, `cached`, `noupdate`, `progress`, `updateready`, and `obsolete`. You can use these events to

keep your users informed about the application’s status. Using the Application Cache can make your game more tolerant to connectivity outages, and it can make your users happy by letting them start game play quicker (after the assets are cached). Also, if you choose, Application Cache can be used to allow users to play your game offline. Don’t worry too much about it right now. In Chapter 11, “Publishing Your Games,” we discuss using the Application Cache in more detail.

Database API

At present, there are multiple ways to store structured data using HTML5, including the WebSQL API implemented by Webkit browsers and the competing IndexedDB API spearheaded by Firefox.

WebSQL API

WebSQL provides structured data storage by implementing an SQL-like syntax. Currently, implementations have centralized around SQLite, but that isn’t a specific requirement.

There isn’t a “createDatabase” function in WebSQL. The function `openDatabase` optimistically creates a database with the given parameters if one doesn’t already exist. To create a database name `myDB`, we would need to make a call in the form

```
var db = openDatabase("myDB", "1.0", "myDB Database", 100000);
```

where we pass “myDB” as the name, assign the version “1.0”, specify a display name of “myDB Database”, and give it an estimated size of 100KB. We could have optionally specified a callback to be executed upon creation. Figure 1-2 shows the content of the Chrome Developer Tools Storage tab, which we will cover in more detail in Chapter 2, “Setting Up Your Development Environment,” after executing the preceding line of code.



Figure 1-2 Storage tab showing a created database

In the window to the right, we can run arbitrary SQL code, as shown in Figure 1-3, where we created a table, inserted some information, and ran a query.



Figure 1-3 Storage tab showing SQL statements

Although not universally supported, the specification does call out the existence of both asynchronous and synchronous database connections and transactions. Our current example creates an asynchronous connection; to create a synchronous one, we would call `openDatabaseSync` with the same parameters. After the initial connection, there is no distinction when it comes to database transactions besides calling `transaction(...)` for read/write transactions and `readTransaction` for read-only transactions.

A word of caution: Synchronous connections are not well supported and, in general, you should structure your code to run asynchronously.

IndexedDB API

IndexedDB stores objects directly in object stores. This makes it easier to implement JavaScript versions of NoSQL databases, like those of the object databases MongoDB, CouchDB, and SimpleDB. At the time of this writing, the implementations of the APIs weren't synchronized and used different naming schemes and strictness to the specification. The Internet Explorer implementation requires an ActiveX plugin. I encourage you to check out <http://nparashuram.com/trialtool/index.html#example=/ttd/IndexedDB/all.html> to see some examples in action on Firefox, Chrome, and Internet Explorer. The Chrome code in most cases will work seamlessly on Safari.

Web Storage

Web Storage provides several APIs for saving data on the client in a fashion similar to browser cookies. There is a `Storage` object for data that needs to persist between restarts named `localStorage` and one for data that will be purged once the session ends named `sessionStorage`. The data is stored as key/value pairs. These two objects implement the functions listed in Table 1-1.

Table 1-1 Web Storage Functions

Function Name	Description
<code>setItem(key:String, value)</code>	Creates a key/value pair given the specified values. Some implementations require the value to be a string.
<code>getItem(key:String)</code>	Returns the item specified by the given key.
<code>removeItem(key:String)</code>	Removes the item identified by the given key.
<code>clear()</code>	Clears all key/value pairs from the Storage object.
<code>key(index:long)</code>	Returns the key for the specific index.

Each `Storage` object also has a `length` property indicating the number of present key/value pairs.

Web Storage offers a more fluent API we can use in lieu of the `getItem` and `setItem` functions listed in Table 1-1. The alternate API uses an array-like means of referencing a key. To set a `localStorage` key/value pair with the values of a hometown newspaper, we could use the following, for example:

```
localStorage['newspaper'] = 'The Baltimore Sun';
```

Likewise, we could retrieve that value with just the left half of the preceding expression:

```
localStorage['newspaper'];
```

In the context of game programming, we could use Web Storage to store user high scores as well as data for saved games.

Geolocation

The Geolocation API doesn't have an explicit function to ask for the user's permission to track his or her position. Instead, the browser handles this transparently for us. When the Geolocation API first requests position information from a website for which it doesn't have permission, a contextual pop-up appears to request permission from the user.

We can check to see if the browser supports the Geolocation API by checking for the following object:

```
navigator.geolocation
```

If it resolves to a non-null value, we have the ability to geolocate.

The calculated position of a user is defined by the `Position` object, which contains a `Coordinates` object named `coords` and a timestamp indicating when the fix was retrieved. Table 1-2 shows the properties of the `coords` object.

Table 1-2 **Coordinates Object Properties**

Property Name	Return Value	Description
<code>latitude</code>	double	The latitude of the position fix.
<code>longitude</code>	double	The longitude of the position fix.
<code>altitude</code>	double	The altitude of the position fix in meters. If this is unavailable, the value will be null.
<code>accuracy</code>	double	The margin of error of the lat-long fix in meters. If this is unavailable, the value will be null.
<code>altitudeAccuracy</code>	double	The margin of error of the altitude value. If this is unavailable, the value will be null.
<code>heading</code>	double	The direction in which the device is traveling in degrees (0° to 360°, inclusive). If this is unavailable, the value will be NaN.
<code>speed</code>	double	The speed in meters that the device is traveling. If this is unavailable, the value will be null.

After we have verified that geolocation is available, obtaining a position fix on a device is simple. We just call `getCurrentPosition` with either one, two, or three parameters, corresponding to the functions to run if getting a fix is successful, if it fails, and the options on the request, respectively.

Listing 1-5 shows the code needed to retrieve a location, draw it on a map with a marker, and draw a proximity circle around the marker.

Listing 1-5 Drawing a Map with Geolocation

```
if(navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(function(pos) {
    var latitude = pos.coords.latitude;
    var longitude = pos.coords.longitude;

    var options = {
      position:new google.maps.LatLng(latitude, longitude)
      ,title:"Your location"};

    var marker = new google.maps.Marker(options);

    var circle = new google.maps.Circle({
      map:map, radius:pos.coords.accuracy
    });
    circle.bindTo('center', marker, 'position');

    marker.setMap(map);

    map.setCenter( new google.maps.LatLng(latitude, longitude));
  },
  function(error) {
    console.log(error.message);
  });
}
```

After verifying that geolocation is available, we first attempt to retrieve a fix on the position of the device. In this example, we are passing in the two parameter functions of `getCurrentPosition` to execute if successful, an error occurs, or if the user declines geolocation. After getting the latitude and longitude portions, we create a marker centered at that position with the title “Your location.” To the marker, we attach a circle whose radius is equivalent to the accuracy of the position fix. Lastly, if there is an error, our error-handling function prints out the error message to the console. Figure 1-4 shows a sample position fix using the OpenStreetMap tile set.

Although we did not use it, we could have also specified an options object that indicates several preferences on the retrieved data. We could also set up a listener to execute every time there is a position change returned from the `watchPosition` function. Geolocation is an expensive API. Use it judiciously and don’t be afraid to cache the location.

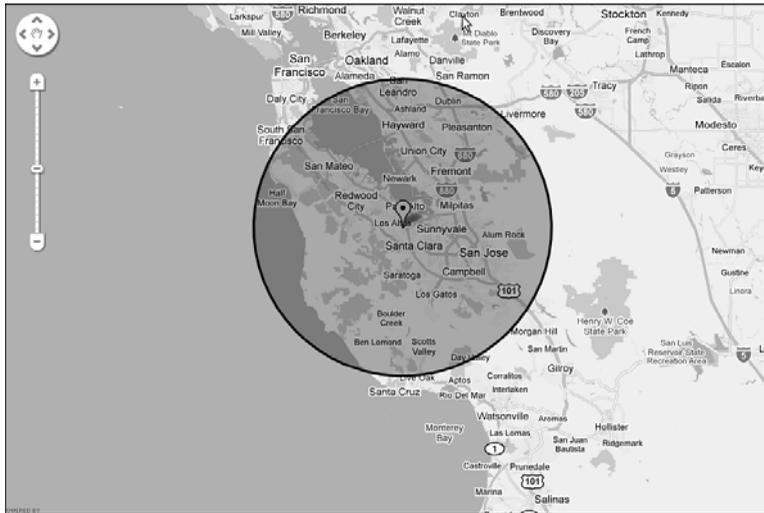


Figure 1-4 Geolocation from the browser

We could use geolocation to create localized leader boards, or on a multiplayer server to match players who are physically close to one another.

Getting Users' Attention with Notifications

In HTML4, the options to communicate messages to the user were limited. You could show the user an alert window or show a message in a `div` element. Showing an alert window is well supported on all browsers, but it is highly disruptive. It is something that requires immediate attention and doesn't let you move on until you have handled it. One sure way to annoy a user is by making him lose a life because some message obscured his view. Showing a message in a `div` element fares slightly better, but there isn't a standard way to add them. These types of messages can be easily ignored. On one side we have notifications that crave attention, and on the other we have notifications that can be easily ignored. There has to be a middle ground. Enter web notifications.

On the Mac OS X and Ubuntu platforms natively, and with a plugin on Windows, an application can send configurable messages to users and notify them of events or changes it deems important. An example of such a notification is shown in Figure 1-5.



Figure 1-5 Desktop notification message

Like their desktop counterparts, web notifications can contain an image along with a contextual message.

Requesting Permission to Display Notifications

Before we can display notifications to users, we first have to get their permission. Explicit permission protects the users from being bombarded with unwanted notifications. We can request permission to display notifications by executing the following:

```
window.webkitNotifications.requestPermission();
```

This will show a contextual message in the browser to allow the user to approve or deny access, as shown in Figure 1-6. Instead of a no-argument function call, we can also pass a function to execute when the user responds to the prompt.



Figure 1-6 Web notification permissions message

We can likewise verify permission by running the following command:

```
window.webkitNotifications.checkPermission();
```

In this case, `checkPermission()` returns an integer that indicates the permission level, as shown in Table 1-3.

Table 1-3 Notification Permission Level

Constant Name	Value
<code>PERMISSION_ALLOWED</code>	0
<code>PERMISSION_UNKNOWN</code>	1
<code>PERMISSION_DENIED</code>	2

Looking at the name, you would expect notifications to work in at least the major Webkit browsers, namely Chrome and Apple Safari. Although Safari uses Webkit, it doesn't implement the Notification API. If the spec is implemented globally, the namespace could presumably change from `webkitNotifications` to simply `notifications`.

Creating Notifications

You can create two types of notifications: simple and HTML. Simple notifications display a simple message with an optional title and icon image, whereas HTML notifications display an arbitrary URL. For example, we can create a simple notification by executing the following:

```
var msg = window.webkitNotifications.createNotification(  
    '', 'Test Notification', 'Hello World'  
);
```

Our notification will have the title “Test Notification” with the message “Hello World.” Because we passed an empty string for the icon image, the API omits it. We can do this for any other parameter. Do this to hide parameters you don’t want displayed. Passing no value to the function will cause a text message of “undefined” or a broken image link. Figure 1-7 shows our notification running in the browser. As you can see, it is pretty Spartan, and we have no control over the design besides the parameters we passed it.



Figure 1-7 Simple web notification

As mentioned before, HTML notifications can get their content from an arbitrary URL such as a website or an image. The function just takes the desired URL to display in the form:

```
var msg =window.webkitNotifications.createHTMLNotification(
    'http://example.com'
);
```

HTML notifications give you no means to resize them, and unless the URL has code to optimize the notification for small screens, scroll bars will probably be included. On a 1680×1050 screen, the default size seems to be approximately 300 pixels wide by 50 pixels high, but because the notifications API is still a draft at the time of this writing, that is certainly subject to change. Until fine-grained height and width attributes are added, stick with simple notifications.

Interacting with Notifications

The resulting notification has two basic functions for controlling it: `show()`, which surfaces the notification to the user, and `cancel()`, which hides the notification if it’s currently visible or prevents it from being displayed if it is not visible. Web notifications can also execute functions in response to notification events. Table 1-4 shows a list of the applicable functions you can specify to respond to events.

Table 1-4 Web Notification Functions

Function Name	Description
<code>onclick</code>	This function will execute if the notification is clicked <i>and</i> the underlying platform supports it. Avoid this event if at all possible.
<code>onclose</code>	This function will execute after the <code>close</code> event is fired. This could be when the user closes the notification or if it is closed programmatically.

Table 1-4 **Web Notification Functions**

Function Name	Description
<code>ondisplay</code>	This function will execute after the <code>show()</code> function is called and the notification is visible to the user.
<code>onerror</code>	This function executes after <code>show()</code> is called in the event of an error.

You can check the current status of the draft specification at <http://dev.chromium.org/developers/design-documents/desktop-notifications/api-specification>.

Media Elements

When HTML was originally designed, it was concerned with mostly textual links. Native display of images would come much later. It is not hard to understand why you would need a plugin or browser extension to play audio or video. In most cases, this meant Flash. HTML5 has tried to address that issue with the inclusion of the audio and video tags.

The audio and video tags allow us to play media in the browser natively. Also, a group of properties can be set to control playback. Here is the most basic HTML form for embedded media (in this case, an audio file):

```
<audio src="song.mp3" autoplay />
```

This creates an audio HTML element, assigns the source to `song.mp3`, and instructs the page to “autoplay” the content. It is equivalent to the following JavaScript code:

```
var song = new Audio();  
song.src = "song.mp3";  
song.autoplay = true;  
song.load();
```

Controlling Media

In addition to the `autoplay` attribute listed in the previous example, several other attributes can be used to control our media. For example,

```
<video src="vid.avi" controls />
```

or

```
var vid = new Video();  
vid.src = "vid.avi";  
vid.controls = true;
```

tells the browser to provide a default set of controls for starting and pausing playback, setting the volume level, and seeking in the stream. In the absence of such a property, the

developer can provide a custom set of controls using the JavaScript functions and properties listed in Tables 1-5 and 1-6.

Table 1-5 **Media Tag Functions**

Function Name	Description
<code>play()</code>	Starts playing the media from the current position and sets the paused property to false
<code>pause()</code>	Halts playing the media and sets the paused property to true
<code>load()</code>	Resets the element and applies any settings, such as pre-fetching

Table 1-6 **Media Element Properties**

Property Name	Accepted Values	Description
<code>currentTime</code>	integer	Sets the position in the media stream for playback
<code>duration</code>	N/A (read-only)	Indicates the length of the source media in seconds
<code>loop</code>	true or false	Specifies whether or not to play the media from the beginning when the end of the stream is reached
<code>autoplay</code>	true or false	Specifies whether or not to play the media as soon as possible
<code>muted</code>	true or false	Specifies whether or not to set the volume at 0.0

The list of properties has been truncated for brevity and usefulness. To see a full list of available properties, check out the HTML5 draft spec at <http://dev.w3.org/html5/spec>.

Handling Unsupported Formats

At the time of this writing, the audio and video elements in different browsers don't necessarily all support the same types of audio and video. The reason a particular browser doesn't support a particular format might be due to the age of the format, competition with an endorsed format, or patent restrictions that the browser's parent company doesn't want to deal with. Media tags have several methods to deal with this.

Listing Multiple Sources

Instead of specifying a single source, the developer can choose to list multiple sources to let the browser choose the appropriate one to use. The following snippet lists two sources

for a video tag and the fallback message if neither format is supported or the browser doesn't support the video tag.

```
<video>
  <source src="video.ogv" />
  <source src="video.avi" />
  <!-- Neither is supported, can show message or fallback to Flash -->
  <div><span>Use a modern browser</span></div>
</video>
```

Although listing multiple sources is an option for a static page, it's not great for applications with dynamic content. For those instances, using the tool Modernizr is recommended. We'll discuss Modernizr in more detail in Chapter 2, but consider this a primer.

Using Modernizr

Modernizr (www.modernizr.com) inspects browser capabilities at runtime and injects the properties into a JavaScript object. To see whether the browser can play audio or video, we would check the value of `Modernizr.audio` or `Modernizr.video` to see if it evaluates to true.

Checking support for a particular format is slightly different. Verifying support for MP3 files is done by checking the value of `Modernizr.audio.mp3`, but the value returned isn't true or false. The HTML5 spec states that the browser should return its confidence level that it can play the format. The return value will be "probably," "maybe," or an empty string. When we use `Modernizr.audio.mp3` in a conditional clause, any non-empty value is treated as true and the empty string is treated as false.

CSS3

CSS3 doesn't fit the scope of this book, and readers are encouraged to explore the specification if they are interested in it. Like HTML5, CSS3 extends its predecessor (CSS2) by adding new features and codifying previous proposals, such as web fonts and speech, which were introduced in previous versions but not widely supported. A useful website for further information is <http://www.css3.info>.

HTML5 Drawing APIs

An interesting area of the HTML5 spec is the new drawing APIs. Canvas, SVG, and WebGL provide bitmapped, vector, and three-dimensional drawing capabilities, respectively.

Canvas

The canvas element started its life as an Apple extension to Webkit, the layout engine powering Safari and Chrome, to display Dashboard gadgets and additions to the Safari browser. It was later adopted by Opera, Firefox, and related browsers, eventually becoming a component of the HTML5 specification. The beta release of Internet Explorer 9 (IE9)

has brought native support to all major browsers, although support in IE9 is not as complete as the aforementioned browsers.

The canvas element can be most simply described as a drawable region with height and width attributes using JavaScript as the medium to draw and animate complex graphics such as graphs and images. A full set of 2D drawing functions is exposed by the JavaScript language. Given the close relationship between JavaScript and ActionScript, a Flash drawing or animation using ActionScript can be easily ported to JavaScript with only moderate effort. Canvas will be covered in more detail in Chapter 5, “Creating Games with the Canvas Tag.”

SVG

SVG (Scalable Vector Graphics) is a mature W3C specification for drawing static or animated graphics. The ability to inline SVG without the use of an object or embed tag was added in HTML5. Vector graphics use groupings of mathematics formulas to draw primitives such as arcs, lines, paths, and rectangles to create graphics that contain the same quality when rendered at any scale. This is a marked benefit over images whose discernible quality degrades when they are displayed at a scale larger than that for which they were designed.

SVG takes a markedly different approach from the canvas element in that it represents drawings in XML files instead of purely in code. XML is not the more concise representation of data, so a file may contain many repeated sections. This can be addressed by compressing the file, which can greatly reduce its size. As with the canvas element, interaction can be scripted using JavaScript. Prior to IE9, IE supported an incompatible vector format called VML. As of IE9, all major desktop browsers support a fairly common feature set of SVG 1.1. Chapter 6, “Creating Games with SVG and RaphaëlJS,” puts SVG front and center.

WebGL

WebGL is a JavaScript API for 3D drawing that enables the developer to assess graphics hardware and control minute details of the rendering pipeline. It is managed by the Khronos group and shares much of its syntax with OpenGL 2.0 ES. At the time of this writing, WebGL is not supported in Internet Explorer 6+ or the stable branches of Opera and Safari. It is available in the stable builds of Firefox and Chrome/Chromium and in development builds of Opera and Safari. Chapter 7, “Creating Games with WebGL and Three.js,” dives into WebGL.

Conveying Information with Microdata

A web application or API parsing a page can interpret HTML marked up with microdata and respond to it. For instance, a search engine that returns results marked up with microdata could be parsed by a browser extension or script to better present the data to a visually impaired or colorblind user. Microformats are a preceding concept that serves the

same goal. One key difference between microformats and HTML5 microdata is the way that the data is denoted. As shown in Listing 1-6, microformats use the `class` property of an object to indicate the fields on an object.

Listing 1-6 hCard Microformat Example

```
<div class="vcard">
  <div class="fn">James Williams</div>
  <div class="org">Some Company</div>
  <div class="tel">650-555-3055</div>
  <a class="url" href="http://example.com/">http://example.com/</a>
</div>
```

Microdata uses the same concept with slightly different notation. Instead of marking properties using classes, the `itemprop` keyword is used. The keyword `itemscope` marks an individual unit. At its core, microdata is a set of name/value pairs composed into items. Listing 1-7 shows a microdata example. The `itemtype` property indicates a definition of the object and specifies valid properties. You could use microdata to encode the names and scores on a leader board page or instructions and screenshots from a game.

Listing 1-7 Microdata Example

```
<p itemprop="address" itemscope
  itemtype="http://data-vocabulary.org/Address">

<span itemprop="street-address">1600 Amphitheatre Parkway</span><br>

<span itemprop="locality">Mountain View</span>,
  <span itemprop="region">CA</span>

<span itemprop="postal-code">94043</span><br>

<span itemprop="country-name">USA</span>

</p>
```

Summary

HTML5 marks a groundbreaking change in how we interact with the browser. This chapter highlighted the major additions that apply to our needs. You learned how Google Chrome Frame brings HTML5 features to IE browsers as well as the multiple ways to draw assets.

In exploring HTML5, in addition to its drawing APIs, you learned about features that allow you to run computationally heavy tasks without blocking the browser, setting up

bidirectional communications channels between applications, and enabling offline execution of applications.

You can download chapter code at www.informit.com/title/9780321767363.

Numerics

2.5D, 84

2D, billboard, 140

3D

Blender, 29, 129

Camera object (Three.js), 128-129

lighting, 120

materials, 120

models,

loading with Three.js, 129-131

sourcing, 143

normal, 121

picking, 142

shading

flat shading, 121

Gouraud shading, 121

Lambertian shading, 121

Phong shading, 122

simulating in 2D space, 84

parallaxing, 85-87

perspective projection, 84

snowman scene

setting up in Three.js, 123-127

viewing in Three.js, 128-129

textures, 134-135

vertex, 118-119

3D Studio MAX, 129

37signals, 153

A

accessing drawing APIs with GWT, 151-152

actions performed in game loop, 53

adding functions to Raphael, 113

AI (artificial intelligence)

- Pong, 68

- Minimax algorithm, 69-70

- tic-tac-toe, 68

AJAX (Asynchronous JavaScript and XML), 2, 43

- aliases, CoffeeScript, 156

- ambient lighting, 120

Android, 180

- application layers, 182

- applications, packaging

- with Appcelerator Titanium, 193-194

- with PhoneGap, 195-198

- audio element support, 192

Angry Birds, 64**animating**

- cards, 107-110

- models, 142-143

- objects along paths, 113

animation

- time-based, 140

- Trident.js, 79

- easing, 81-82

- keyframes, 81

- spritesheets, 83

- timelines, creating, 80

- z-ordering, 86

APIs

- Canvas, 15-16

- Components API, 58

- Core API, 57-58

- drawing APIs for GWT, accessing, 151-152

- for Appcelerator Titanium, 191

- for PhoneGap, 189

- Geolocation API, 8-10

- IndexedDB API, 7

- JFugue, 89

- networking, 58

- node-cache, 168

- storage APIs, Lawnchair, 183-185

- SVG, 16

- WebGL, 16

- WebSQL API, 6-7

Appcelerator Titanium

- Android applications, packaging, 193-194

- APIs, 191

Application Cache, 5-6

- applications, running offline, 201

- manifest file, 201-203

application frameworks

- Appcelerator Titanium, 191

- Android applications, packaging, 193-194

- APIs, 191

- PhoneGap, 188

- Android applications, packaging, 195-198

- APIs, 189

- documentation, 190

- Event API, 189

- FileReader object, 190

- FileUpload object, 190

- FileWriter object, 190

applications

- attributes, configuring, 210-211

- deploying games as, 183

- extensions, 206

- hosted versus packaged, 212

packaging for TapJS, 215

publishing

- on Chrome Web Store, 206-208
- with Kongregate, 217
- with TapJS, 212, 215-217

simplifying with ExpressJS, 163

- application structure, 165
- CoffeeKup, installing, 166
- CoffeeKup, layout files, 167-168
- CoffeeKup, registering, 167
- session management, 165
- URL routing, 163-165

TapJS, creating, 213

uploading to Chrome Web Store, 208-210

applying textures to spheres, 135

arithmetic operators, JavaScript, 32

arrays

- sets, 54
- sorting, 55

Ars Technica, 153

aspect ratio, 128

Asteroids, 66-67

asynchronous connections, WebGL API, 7

attributes of applications, configuring, 210-211

audio

- Copy Me* game tones, creating, 88-89
- multiple sounds
 - playing at once, 90
 - playing sequentially, 91

audio element support (Android)192

audio tag (HTML5), controlling media, 13-14

B

beginPath() function, 72

benchmarking

- frame rate, checking with Stats.js, 144
- with WebGL Inspector, 145

Berners-Lee, Tim, 1

Bezier curves, 112

Bezier, Pierre, 111

billboarding, 140

bitmap images, creating with SVG files, 105

Blender, 29, 84, 129

Blender Conference, 130

browser tools

- Chromer Developer tools, 24-25
- Firebug, 26
- Safari Developer tools, 26

browsers

- Geolocation API support, verifying, 8
- Google Gears, 3

building Pong with SGF

- AI, 68
- game physics, 64-66
- game pieces, drawing, 61-63
- host page, 59-60
- main.js file, 60-61

C

CACHE section (Application Cache manifest file), 201

caching data, 168

Camera object (Three.js), 128-129

Canvas, 15-16, 71

- comparing with SVG, 95-96
- displaying in Jo, 188

- drawing state, saving and restoring, 77
- images, drawing, 79
- paths, drawing, 72
- sprites, drawing, 73-74
- transformations, 75-77

capacitive screens, gestures, 181**Cappuccino, 158****cards**

- animating, 107-110
- drawing, 105
- flipping, 108
- shuffling, 107

Chrome (Google), extensions, 25**Chrome Developer tools, 24-25****Chrome Frame, 3****Chrome Web Store**

- applications, publishing
 - hosted application, deploying, 207-208
- metadata, describing, 206
- packaged application,
 - deploying, 208
- testing applications, 208
- applications, uploading, 208-210

classes

- CoffeeScript, 157-158
- JavaScript, inheritance, 38-40

client-side scripting

- JQTouch, 187
- jQuery, 185
- jQueryMobile, 185-186
- Zepto.js, 187

client/server communication

- NowJS, 171
- Web Sockets, Socket.IO, 169-170

clipping planes, 128**code, minification, 199-201****CoffeeKup**

- installing, 166
- layout files, 167-168
- registering with ExpressJS, 167

CoffeeScript, 45

- aliases, 156
- classes, 157-158
- conditional statements, 156
- files, compiling, 153
- for loops, 156-157
- functions, 154
- installing, 153
- semicolons, use of, 154
- splats, 155
- var keyword, 154

collision detection (Pong), 65-66**color, specifying in Raphael, 103-104****color picking, 142****comments, minification, 199-201****Comparator (JavaScript), 55-56****comparing**

- CoffeeScript and JavaScript, 154
- microdata and microformats, 17
- SVG and Canvas, 95-96
- XML and JSON, 44

comparison operators (JavaScript), 34-35**compilers, Google Closure Compiler, 199-201****compiling CoffeeScript files, 153****Components API, 58****conditional loops (JavaScript)**

- for loops, 37
- if-else statement, 35
- switch-case statement, 36
- while loops, 36

conditional statements, CoffeeScript, 156
configuring application attributes, 210-211
console debugging, 172-173
controlling
 media in HTML, 13-14
 program flow with loops (JavaScript)
 for loops, 37
 while loops, 36
Conway's Game of Life, 136
Copy Me, 87
 game text, drawing, 91
 game text, styling, 92
 game tones, creating, 88-91
 objects, drawing, 87-88
Core API, 57-58
creating
 Copy Me game tones, 88-91
 game rooms with NowJS groups,
 174-175
 notifications, 11-12
 particle systems in Three.js, 140-141
 physics system with JigLibJS, 139-140
 TapJS applications, 213
 timelines in Trident.js, 80
 vertex in Three.js, 118-119
 Web Sockets, 4
cross-platform frameworks, Jo, 187-188
cross-site scripting, 44
CSS (Cascading Style Sheets), 92, 315
cube mapping, 135
Cufon, 100-102
curveto instruction (Raphael), 111
customizing fonts, 100-102

D

data URIs, 78
Database API
 IndexedDB API, 7
 WebSQL API, 6-7
debugging Node applications, 172-173
deciding genre for game, 52-53
deploying
 games
 as applications, 183
 as website, 181
 hosted applications, 207-208
 packaged applications, 208
describing metadata, 206
design document, writing, 51-52
desktop applications, JavaScript, 46-47
development tools
 Blender, 29
 Chrome Developer tools, 24-25
 Eclipse IDE, installing, 20-21
 Firebug, 26
 GWT, installing, 22
 Inkscape, 27
 Java, installing, 19-20
 ProcessingJS, 27
 Raphael, 29
 Safari Developer tools, 26
 SVG-edit, 27
directional lighting, 120
displaying canvas in Jo, 188
documentation, PhoneGap, 190
drawImage function, 78, 86

drawing

- cards, 105-110
- Copy Me* game objects, 87-88, 91
- images on Canvas, 79
- Pong game pieces, 61-63

drawing APIs

- Canvas, 15-16
- for GWT, 151-152
- SVG, 16
- WebGL, 16

drawing state (Canvas), saving and restoring, 77

Dynamic DNS services, 204

E

easing, 81-82

Eclipse IDE, installing, 20-21

equals method, 54

Event API (PhoneGap), 189

events (jQuery), 43

exporting paths from SVG file, 112

ExpressJS, 163

- application structure, 165
- sessions, managing, 165
- URL routing, 163-165

extending Raphael with plug-ins, 113-114

extensions, 25, 206

F

Facebook integration, TapJS, 214-217

FALLBACK section (Application Cache manifest file), 202

FileReader object (PhoneGap), 190

files (CoffeeScript), compiling, 153

FileUpload object (PhoneGap), 190

FileWriter object (PhoneGap), 190

filters, SVG, 113

Firebug, 26

first-class objects, 33-34

flat shading, 121

flipping cards, 108

fonts, Cufon, 100-102

for loops

- CoffeeScript, 156-157
- JavaScript, 37

format, data URIs, 78

forward kinematics, 142

FOV (field of view), 128

fragment shaders, 121

frame rate, checking with Stats.js, 144

frames per second versus time-based animation, 140

functions

- adding to Raphael, 113
- beginPath(), 72
- CoffeeScript, 154
- drawImage, 78, 86
- JavaScript, 32-34, 38
- node-cache API, 168
- requestAnimationFrame, 123
- updateDynamicsWorld, 139

G

game assets, loading in Raphael, 104-105

game loop, actions performed, 53

game physics

- rigid body dynamics, 137-138
- soft-body dynamics, 138

game pieces, drawing (Pong), 61-63

game play, managing for multiplayer games, 175-176

game rooms

- creating with NowJS groups, 174-175
- moving between, 175

game server lobby, creating, 173-174

genre of game, deciding on, 52-53

Geolocation API, 3, 8-10

geometry shaders, 121

gestures, 181

JQTouch support, 187

jQueryMobile support, 186

Zepto.js support, 187

GLSL (OpenGL Shader Language), 131-133

GLUEScript, 46

Google App Engine, 23

Google Chrome

extensions, 25

V8, 161

Google Chrome Frame, 3

Google Closure Compiler, minification, 199-201

Google Gears, 3

Google plugin for Eclipse, installing, 20-21

Google SketchUp, 143

Gouraud shading, 121

gradients, 103

Grouchnikov, Kirill, 79

GWT (Google Web Toolkit), 45, 147

drawing APIs, 151-152

gwt-html, 5-media module, 151

JSNI, 149

Pyjamas, 158

RaphaëlGWT, 150

widgets, RootPanel, 148-149

installing, 22

gwt-html, 5-media module (GWT), 151

H

host page, Pong, 59-60

hosted applications

deploying, 207-208

versus packaged applications, 212

hosted Node.js services, Nodester, 204-205

hosting your own server, 203-204

HTML host page, Pong, 59-60

HTML, 51

Application Cache, 5-6

applications, running offline, 201

manifest file, 201-203

applications, publishing to desktop, 217-218

canvas tag, 71

drawing state, saving and restoring, 77

paths, drawing, 72

sprites, drawing, 73-74

transformations, 75-77

data URIs, 78

drawing APIs

Canvas, 15-16

SVG, 16

WebGL, 16

Geolocation API, 8-10

gwt-html, 5-media module (GWT), 151

IndexedDB API, 7

media elements, 13-14

microdata, 17

spritesheets, 78

unsupported media elements, handling

listing multiple sources, 14-15

with Modernizr, 15

Web Storage, 7-8

Web Workers, 4-5

WebSockets, 4

WebSQL API, 6-7

HTML5 tools

Inkscape, 27

ProcessingJS, 27

Raphael, 29

SVG-edit, 27

I

if-else statement (JavaScript), 35

images

- bitmap, creating with SVG files, 105
- drawing on Canvas, 79
- serving, 78

IndexedDB API, 7**inertia, Newton's first law, 63****inheritance, 38**

- CoffeeScript, 158
- Prototype library, 39-40

injection attacks, cross-site scripting, 44**Inkscape, 27, 97****installing**

- CoffeeKup, 166
- CoffeeScript, 153
- Eclipse IDE, 20-21
- Google plugin for Eclipse, 20-21
- GWT, 22
- Java, 19-20
- n script file, 162
- node-inspector, 172

interacting with notifications, 12**inverse kinematics, 142****iOS, 179****is-a relationships (JavaScript), inheritance, 38**

J

Java, installing, 19-20**JavaScript, 1**

- AJAX, 2
- and CoffeeScript, comparing, 154
- arithmetic operators, 32
- as intermediary language, 45
- basic types, 31
- Comparator, 55-56

- comparison operators, 34-35

- conditional loops

- for loops, 37
- if-else statement, 35
- switch-case statement, 36
- while loops, 36

- functions, 32-33

- first-class objects, 33-34
- setInterval, 38
- setTimeout, 38

- inheritance, 38-40

jQuery

- AJAX, 43-44
- events, 43
- ready function, 41
- selectors, 42

JSON, 44-45

- linked lists, 56-57
- mobile platforms, 45
- modules, 48
- on the desktop, 46-47
- server-side, 48
- set class, 54

Jetty, 98**JFugue, 89****JigLibJS**

- physics system, creating, 139-140
- setting up, 138

Jo, 187-188**JQTouch, 187****jQuery, 41, 185**

- AJAX, 43
- cross-site scripting, 44
- events, 43
- ready function, 41
- selectors, 42

JQueryMobile, 185-186
 JSNI (JavaScript Native Interface), 149
 JSON, 44-45
 JSONP (JSON with padding), 45
 JVM (Java Virtual Machine), 48

K

key/value store databases, 183
 keyframes, 81, 142
 KHTML, 217
 Knuth, Donald, 107
 Kongregate, publishing games, 217

L

Lambertian shading, 121
 launching games

- as applications, 183
- as website, 181

 Lawnchair, 183

- records
 - removing, 185
 - retrieving, 184
 - store, creating, 184

 layout files, CoffeeKup, 167-168
 libraries (JavaScript), Prototype, 39-40
 lighting, 120-122
 linear gradients, 103
 lineto instruction (Raphael), 110-111
 linked lists, 56-57
 listing multiple media sources in HTML, 14-15
 LiveScript, 31
 loading

- 3D models with Three.js, 129-131
- game assets in Raphael, 104-105

 lobby for multiplayer games, creating, 173-174

Local Server module (Google Gears), 3
 localStorage object (Web Storage), 7-8
 LOD (level of detail), 121
 loops

- CoffeeScript, for loops, 156-157
- JavaScript
 - for loops, 37
 - while loops, 36

M

main.js file, Pong, 60-61
 MakeHuman, 143
 managing

- ExpressJS sessions, 165
- multiplayer games, game play, 175-176
- multiple Node versions, 162

 manifest files, Application Cache, 201-203
 manifest.json

- hosted applications, deploying, 207-208
- metadata, describing, 206
- packaged applications, deploying, 208

 materials, 120
 matrices, 75-76
 media elements (HTML5), 13

- controlling, 13-14
- unsupported, handling, 5
 - listing multiple sources, 14-15
 - with Modernizr, 15

 metadata, describing, 206
 methods, equals, 54
 microdata, 17
 microformats, comparing with microdata, 17
 MIDI, creating *Copy Me* game tones, 88-91
 minification, 199-201
 Minimax algorithm, 69-70

mobile games

- Android, packaging applications
 - with Appcelerator Titanium, 193-194
 - with PhoneGap, 195-198
- platform, selecting, 179
 - Android, 180
 - iOS, 179
 - WebOS, 180
 - Windows Phone 7, 180
- mobile JavaScript platforms, 45
- models, animating, 142-143
- Modernizr, handling unsupported media elements in HTML, 15
- modules
 - Google Gears, 3
 - JavaScript, 48
- momentum, Newton's second law, 63
- morph targets, 142
- moveto instruction (Raphael), 110-111
- moving between game rooms, 175
- multiplayer games
 - game play, managing, 175-176
 - game rooms, creating with NowJS groups, 174-175
 - lobby, creating, 173-174
 - participants, managing, 175
- multiple media sources, listing (HTML5), 14-15
- multiple Node versions, managing, 162
- multiple sounds
 - playing at once, 90
 - playing sequentially, 91
- multitouch screens, gestures, 181

N

-
- n script file, installing, 162
 - Network module (Appcelerator Titanium), 191
 - NETWORK section (Application Cache manifest file), 201
 - networking APIs, 58
 - Newton's laws, 63
 - Node applications, debugging, 172-173
 - Node Package Manager, 162
 - node-cache project, 168
 - node-inspector, installing, 172
 - Node.js, 23, 204
 - applications, debugging, 172-173
 - ExpressJS, 163
 - application structure, 165
 - CoffeeKup, 166-168
 - installing, 166
 - layout files, 167-168
 - registering, 167
 - sessions, managing, 165
 - URL routing, 163-165
 - multiple versions, managing, 162
 - Node Package Manager, 162
 - require statement, 161-162
 - Socket.IO, 169-170
 - nodes, 56-57
 - Nodester, 204-205
 - nonlinear timelines, creating, 81-82
 - normal, 121
 - NoSQL key/value stores, 183
 - notifications
 - creating, 11-12
 - interacting with, 12
 - requesting permission to display, 11
 - NowJS, 171

NowJS groups, creating game rooms, 174-175
 NPM modules, managing with Nodester, 205

O

Objective-J, 158
 objects, JavaScript, 31
 offline access, running applications with Application Cache, 201-203
 OpenGL ES, 117
 Opera Unite, 23
 operating systems, selecting mobile platforms
 Android, 179-180
 WebOS, 180
 Windows Phone 7, 181
 operator overloading, 55
 ordering transformations, 76-77
 orthographic projection, 84

P

packaged applications
 deploying, 208
 versus hosted applications, 212
 packaging applications for TapJS, 215
 paper (Raphael)
 creating, 98-99
 functions, adding, 113
 parabolic arc, 64
 parallaxing, 85-87
 participants, managing in multiplayer games, 175
 particle systems, 66
 Asteroids, 66-67
 creating in Three.js, 140-141
 paths
 animating objects on, 113
 creating with RaphaelGWT, 150

 drawing in Canvas, 72
 exporting from SVG file, 112
 RaphaelJS, 110
 permission to display notifications, requesting, 11
 persistence
 data caching, 168
 Nodester, 205
 perspective projection, 84
 PhoneGap, 188
 Android applications, packaging, 195-198
 APIs, 189
 documentation, 190
 FileReader Object, 190
 FileUpload Object, 190
 FileWriter Object, 190
 Phong reflection, 122
 Phong shading, 122
 physics
 Angry Birds, 64
 applying to Pong game pieces, 64-66
 forward kinematics, 142
 Newton's laws, 63-64
 particle systems, 66
 Asteroids, 66-67
 creating in Three.js, 140-141
 rigid-body dynamics, 137-138
 soft-body dynamics, 138
 physics engines, JigLibJS
 physics system, creating, 139-140
 setting up, 138
 picking, 142
 plane, 127
 platforms
 Android, application layers, 182
 cross-platform JavaScript frameworks, Jo, 187-188
 deploying games for, 182

- for mobile games, selecting, 179
 - Android, 180
 - iOS, 179
 - WebOS, 180
 - Windows Phone 7, 180
- plug-ins (Raphael), extending, 113-114**
- point lighting, 120**
- Pong, building with SGF**
 - AI, 68
 - game physics, 64-66
 - game pieces, drawing, 61-63
 - host page, 59-60
 - main.js file, 60-61
- ProcessingJS, 27**
- program flow, controlling with loops**
 - for loops, 37
 - while loops, 36
- programming shaders, GLSL, 131-133**
- Prototype library (JavaScript), inheritance, 39-40**
- publishing applications**
 - on Chrome Web Store
 - applications, testing, 208
 - hosted application, deploying, 207-208
 - metadata, describing, 206
 - packaged application, deploying, 208
 - Kongregate, 217
 - to desktop, 217-218
 - with TapJS, 212, 215-217
- Pyjamas, 158**

Q-R

- game assets, loading, 104-105
- paths, animating objects on, 113
- plug-ins, 113-114
- RaphaelGWT, 150**
- RaphaelJS**
 - cards
 - animating, 107-110
 - drawing, 105
 - flipping, 108
 - shuffling, 107
 - curveto instruction, 111
 - development environment, setting up, 97
 - fonts, customizing, 101-102
 - game text, creating, 99
 - moveto instruction, 110-111
 - paper, creating, 98-99
 - paths, 110
- ray casting, 142**
- ready function (jQuery), 41**
- records**
 - removing with Lawnchair, 185
 - retrieving with Lawnchair, 184
- registering CoffeeKup with ExpressJS, 167**
- removing records with Lawnchair, 185**
- requestAnimationFrame function, 123**
- requesting permission to display notifications, 11**
- requests (AJAX), performing in jQuery, 43**
- require statement (Node.js), 161-162**
- restoring Canvas drawing state, 77**
- retrieving**
 - images
 - with data URIs, 78
 - with spritesheets, 78
 - records with Lawnchair, 184
- reversing timelines, 81**
- rigging, 142-143**

rigid-body dynamics, 137-138
 RingoJS, 23, 48
 Roosendaal, Ton, 129
 RootPanel widget, 148-149

S

Safari Developer tools, 26
 saving Canvas drawing state, 77
 scene graphs, 123
 scripting languages, JavaScript, 1
 selecting

- application frameworks
 - Appcelerator Titanium, 191
 - PhoneGap, 188-190
- game genre, 52-53
- mobile platform, 179
 - Android, 180
 - iOS, 179
 - WebOS, 180
 - Windows Phone 7, 180

 selectors (jQuery), 42
 semicolons in CoffeeScript, 154
 server-side JavaScript, 48
 servers, hosting your own, 203-204
 serving images, 78
 session management, ExpressJS, 165
 sessionStorage object (Web Storage), 7-8
 set class (JavaScript), 54
 setInterval function (JavaScript), 38
 sets, 54-55
 setTimeout function (JavaScript), 38
 setting up JigLibJS, 138
 SGF, 57-59, 66-68
 shaders, 121

- GLSL, 131-133
- variables, 132

 shading

- flat shading, 121
- Gouraud shading, 121
- Lambertian shading, 121
- Phong shading, 122

 ShapeBuilder API, 152
 shuffling cards, 107
 simplifying applications with ExpressJS, 163

- application structure, 165
- CoffeeKup
 - installing, 166
 - layout files, 167-168
 - registering, 167
- session management, 165
- URL routing, 163-165

 simulating 3D in 2D space, 84

- parallaxing, 85-87
- perspective projection, 84

 snowman scene, setting up in Three.js, 123-127
 Socket.IO, 169-170
 soft-body dynamics, 138
 sorting arrays and sets, 55
 sound, adding with gwt-html5-media module (GWT), 151
 sourcing 3D models, 143
 specifying color in Raphael, 103-104
 speed considerations for SVG, 114
 spheres applying textures, 135
 SpiderMonkey, 46
 splats, 155
 sprites, drawing in Canvas, 73-74
 spritesheets, 78, 83
 starting

- applications with Nodester, 205
- timelines in Trident.js, 80

Stats.js, checking frame rate, 144

storage APIs, Lawnchair, 183

- records
 - removing, 185
 - retrieving, 184
- store, creating, 184

Storage objects (Web Storage), 7-8

storing structured data

- IndexedDB API, 7
- WebSQL API, 6-7

structured data, storing

- IndexedDB API, 7
- WebSQL API, 6-7

styling text for *Copy Me* game, 92

Suzanne Awards, 130

SVG (Scalable Vector Graphics), 16, 95

- Bezier curves, 112
- comparing with Canvas, 95-96
- files, converting to bitmap images, 105
- filters, 113
- paths, exporting, 112
- speed considerations, 114

SVG-edit, 27

switch-case statement (JavaScript), 36

synchronous connection, WebSQL API, 7

T

tags, canvas, 71

- drawing state, saving and restoring, 77
- images, drawing, 79
- paths, drawing, 72
- sprites, drawing, 73-74
- transformations, 75-77

TapJS

- applications
 - creating, 213
 - packaging, 215
 - publishing, 212, 215-217

testing applications with Chrome, 208

texels, 134

text

- Copy Me*, game
 - drawing, 91
 - styling, 92
- creating with RaphaelJS, 99
- fonts, Cufon, 100-102

textures, 134

- applying to spheres, 135
- cube mapping, 135
- UV mapping, 134

themes, 206

Three.js, 117, 136

- 3D models, loading, 129-131
- Camera object, 128-129
- lighting, 120
- materials, 120
- particle systems, creating, 140-141
- ray casting, 142
- snowman scene
 - setting up, 123-127
 - viewing, 128-129
- vertex, creating, 118-119

Tic-Tac-Toe game

- AI, 68
- sprites, drawing on canvas, 73-74

time-based animation versus frames per second, 140

timelines, 83

- creating in Trident.js, 80
- keyframes, 81
- nonlinear, creating, 81-82
- reversing, 81

Titanium Appcelerator, 45

transformations, 75-76

- drawing state, saving and restoring, 77
- ordering, 76-77

transitions

JQTouch, 187

jQueryMobile support, 186

Trident.js, 79

easing, 81-82

keyframes, 81

spritesheets, 83

timelines, reversing, 81

timelines, creating, 80

TurboSquid, 143

U

unsupported media elements in HTML5, handling

listing multiple sources, 14-15

with Modernizr, 15

updateDynamicsWorld function, 139**uploading applications to Chrome Web Store, 208-210****URL routing with ExpressJS, 163-165****user input, 53****UV mapping, 134**

V

V8, 161**var keyword (CoffeeScript), 154****variables for shaders, 132****vectors, normal, 121****verifying Geolocation API support on browsers, 8****vertex, 118-119****vertex shaders, 121, 132-133****video tag (HTML5), 13-14****viewing snowman scene in Three.js, 128-129**

W

web browsers

Geolocation API support, verifying, 8

Google Chrome V8, 161

Google Gears, 3

MIDI files, playing, 89

web notifications

creating, 11-12

interacting with, 12

permission to display, requesting, 11

web server tools, 23**Web Sockets**

simplifying with NowJS, 171

Socket.IO, 169-170

Web Storage, 7-8**Web Workers, 4-5****WebGL, 16, 117****WebGL Inspector, 145****webhosting**

hosted applications, deploying, 207-208

hosted Node.js services, Nodester, 204-205

hosting your own server, 203-204

packaged applications, deploying, 208

WebOS, 45, 180**websites**

CSS, 315

launching games as, 181

Nodester, 204

PhoneGap, 190

WebSockets, 4**WebSQL API, 6-7****while loops (JavaScript), 36****widgets (GWT), RootPanel, 148-149**

Windows Phone 7, 180

WorkerPool module (Google Gears), 3

writing

design document, 51-52

shaders in GLSL, 132-133

X-Y-Z

XML, comparing with JSON, 44

XMLHttpRequest object, 2

XULJet, 46-47

XULRunner, 46, 217

z-ordering, 85

Zepto.js, 187