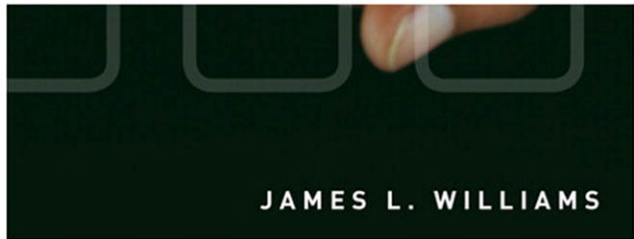




# LEARNING **HTML5** GAME PROGRAMMING

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

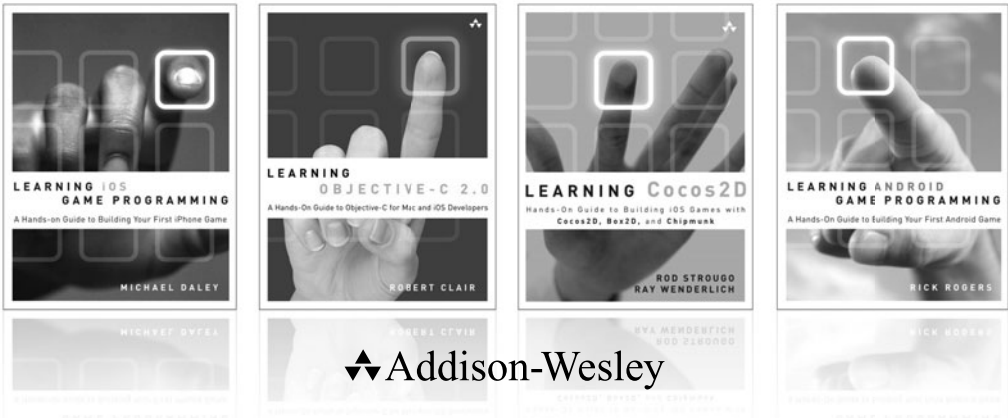


JAMES L. WILLIAMS

# Learning HTML5 Game Programming

---

# Addison-Wesley Learning Series



Visit [informit.com/learningseries](http://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.

# 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](http://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
<b>Chapter 3</b>	<b>Learning JavaScript 31</b>
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

<b>Chapter 11</b>	<b>Publishing Your Games</b>	<b>199</b>
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
<b>Index</b>		<b>219</b>

# 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 Schwentker) 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

**H**TML5 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

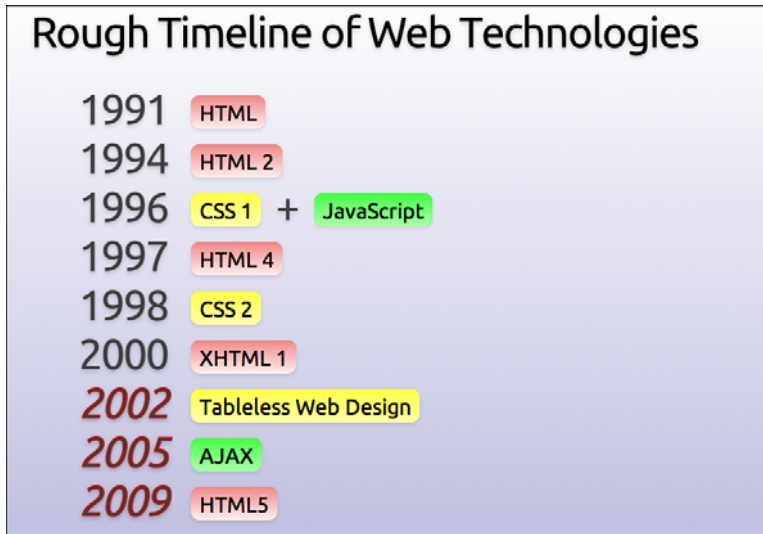


Figure 1-1 HTML timeline

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://websockets.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 `LocalStorage` 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.

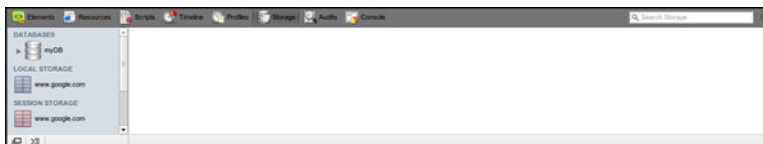


Figure 1-2 Storage tab showing a created database

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.