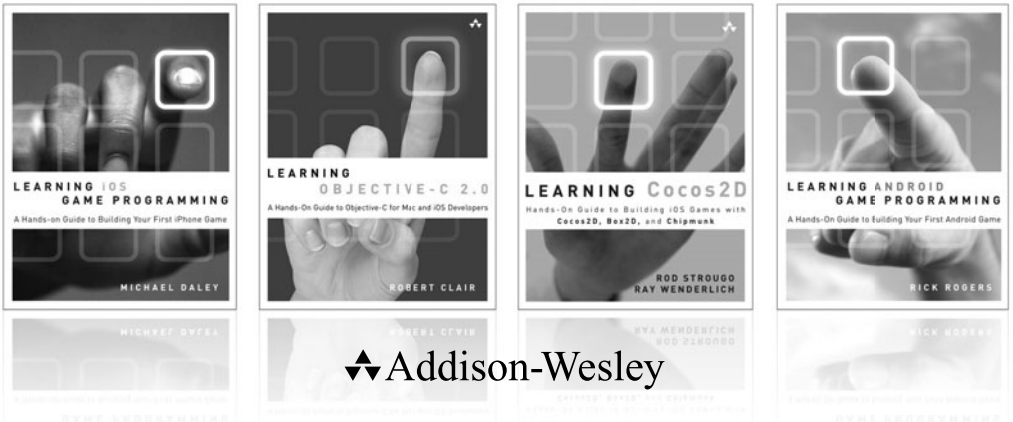
# LEARNING HTML5 GAME PROGRAMMING

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.

# 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

*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



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

# 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*

# 1

# 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 become widely used for scripting client-side

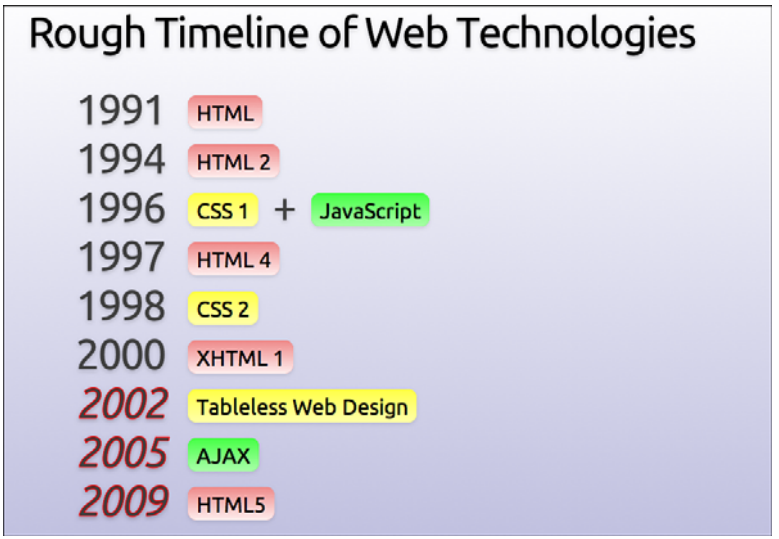



Figure 1-1 HTML timeline

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

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