

Building highly functional, feature-rich web apps
for the Android platform



Beginning Android Web Apps Development

Develop for Android using HTML5, CSS3, and JavaScript

Jon Westfall | Rocco Augusto | Grant Allen



apress®

Beginning Android Web Apps Development

Develop for Android using HTML5, CSS3,
and JavaScript



**Jon Westfall
Rocco Augusto
Grant Allen**

apress®

Beginning Android Web Apps Development: Develop for Android using HTML5, CSS#, and Javascript

Copyright © 2012 by Jon Westfall, Rocco Augusto, Grant Allen

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

ISBN 978-1-4302-3957-4

ISBN 978-1-4302-3958-1 (eBook)

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

President and Publisher: Paul Manning

Lead Editor: Mark Beckner

Technical Reviewer: Stephen Hughes

Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Louise Corrigan,

Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson,

Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper,

Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing,

Matt Wade, Tom Welsh

Coordinating Editor: Adam Heath

Copy Editor: Chandra Clarke

Compositor: MacPS, LLC

Indexer: SPi Global

Artist: SPi Global

Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code.

Dedicated to my wife, Karey, and to my parents, Alan & Dianne

—Jon Westfall

*I dedicate this book to my daughter Rose... and Perry White, who is too good of a reporter
to not know Clark Kent is Superman. You Perry are a true gem.*

—Rocco Augusto

Contents at a Glance

| | |
|---|------------|
| Contents..... | v |
| About the Authors..... | ix |
| About the Technical Reviewer | x |
| Acknowledgments | xi |
| Introduction | xii |
| ■ Chapter 1: Harnessing the Power of the Mobile Web | 1 |
| ■ Chapter 2: Twitter Applications: Who's That Tweet? | 21 |
| ■ Chapter 3: Twitter Applications: I Love Ham | 39 |
| ■ Chapter 4: Basic Planning and Structuring of Your Application | 49 |
| ■ Chapter 5: Handling Multiple Screen Resolutions with CSS 3..... | 65 |
| ■ Chapter 6: Handling Different Browser Platforms | 85 |
| ■ Chapter 7: Building an Impressive User Experience with jQuery Mobile | 99 |
| ■ Chapter 8: Building Visually Rich Internet Applications..... | 121 |
| ■ Chapter 9: HTML5 Location-Based Applications | 145 |
| ■ Chapter 10: Using Cloud Services: A Transport Application..... | 167 |
| ■ Chapter 11: Pushing the Limits with Audio and Video | 187 |
| ■ Chapter 12: Supercharging the User Experience with AJAX | 211 |
| ■ Chapter 13: PackagingYour Applications | 233 |
| Index..... | 261 |

Contents

| | |
|---|------------|
| Contents at a Glance | iv |
| About the Authors | ix |
| About the Technical Reviewer | x |
| Acknowledgments | xi |
| Introduction | xii |
| | |
| ■ Chapter 1: Harnessing the Power of the Mobile Web | 1 |
| Basics of Web Design | 1 |
| Getting Started: HyperText Markup Language (HTML)..... | 1 |
| Getting Stylish: Cascading Style Sheets (CSS)..... | 3 |
| Getting Interactive: JavaScript..... | 6 |
| Getting Informative: Extensible Markup Language (XML) | 8 |
| JSON: Human-Readable Data Interchange | 10 |
| The Mobile Web, Yesterday and Today | 11 |
| Knowing the Speeds (Or “What is 3G anyway?”) | 11 |
| Languages and Protocols, Yesterday and Today | 12 |
| Concepts We Like—And What’s Ahead! | 15 |
| Concept 1: Think Like A User | 16 |
| Concept 2: Don’t Annoy the User | 16 |
| Concept 3: Test-Retest Reliability..... | 17 |
| Concept 4: Keep it Simple Stupid! | 17 |
| Coming Up | 19 |
| | |
| ■ Chapter 2: Twitter Applications: Who’s That Tweet? | 21 |
| JSONP | 22 |
| Setting Up Your Development Environment..... | 22 |
| Your First Mobile Web Application | 26 |
| Summary | 38 |
| | |
| ■ Chapter 3: Twitter Applications: I Love Ham | 39 |
| The HTML | 39 |
| The CSS | 41 |
| The JavaScript | 43 |
| Summary | 48 |

| | |
|---|------------|
| Chapter 4: Basic Planning and Structuring of Your Application | 49 |
| Know Thy Audience | 49 |
| Giving People What They Want | 50 |
| Who Is My User? | 51 |
| Having a Plan | 53 |
| All Mobile Is Not the Same | 54 |
| Structuring Your Application | 57 |
| User Movement: Navigation or Storyboard | 59 |
| Structuring Your Development | 60 |
| Code Structure | 60 |
| Folder Structure | 62 |
| Summary | 63 |
| Chapter 5: Handling Multiple Screen Resolutions with CSS 3 | 65 |
| A History of Tired Eyes and Resolution Evolution | 65 |
| The Daily Droid | 69 |
| The Daily Droid's Base HTML Code | 72 |
| The Daily Droid's Semi-magical CSS Code | 74 |
| Media Queries | 79 |
| Summary | 83 |
| Chapter 6: Handling Different Browser Platforms | 85 |
| META Tags and the Viewport | 85 |
| A Little META History | 86 |
| The Viewport Element | 86 |
| The User Agent | 89 |
| The Nexus One User Agent | 89 |
| PHP User Agent Detection | 91 |
| JavaScript User Agent Detection | 92 |
| Introducing the JavaScript Agent Detection Code | 92 |
| .htaccess User Agent Detection | 97 |
| Summary | 98 |
| Chapter 7: Building an Impressive User Experience with jQuery Mobile | 99 |
| The Basics | 99 |
| Adding Multiple Pages | 102 |
| So – About Those Transitions | 109 |
| Let's Have a Dialog | 111 |
| Rolling Your Own Theme with ThemeRoller | 113 |
| Rolling it All Together: Simple Calc | 118 |
| Summary | 120 |
| Chapter 8: Building Visually Rich Internet Applications | 121 |
| Finding and Using Icons and Stock Photography | 121 |
| Iconfinder | 121 |
| Find Icons | 123 |
| Using an Icon | 123 |
| deviantART | 124 |
| iStockphoto | 125 |
| Guidance on Using Photos in Web Apps | 126 |

| | |
|---|------------|
| Web Fonts | 127 |
| Google Web Fonts | 128 |
| Font Issues to Consider | 129 |
| CSS Frameworks | 130 |
| 1140px Grid..... | 131 |
| Less Framework 4 | 132 |
| 320 and Up..... | 133 |
| Comparing Frameworks: About Jon!! | 135 |
| Adobe Fireworks..... | 141 |
| Summary | 144 |
| Chapter 9: HTML5 Location-Based Applications | 145 |
| The Mechanics of Geolocation | 145 |
| Understanding Device Capabilities | 145 |
| Understanding HTML5 Capabilities | 147 |
| Detecting Browser Geolocation Support..... | 147 |
| Exploring Our Sample Application | 150 |
| Building Our Basic Geolocation Application | 151 |
| Dealing with the Four Corners of Android's Geolocation World | 155 |
| Expanding Your Horizons with Maps | 160 |
| Adding a Map to Our Application | 160 |
| Gaming Your Location..... | 164 |
| Gaming Your Location—for Fun! | 165 |
| Summary | 166 |
| Chapter 10: Using Cloud Services: A Transport Application..... | 167 |
| Introducing the "Move Me" Example Application | 169 |
| Examining the Code | 169 |
| Dealing with Global State..... | 173 |
| Customizing Location Markers | 173 |
| Preparing Our Map..... | 174 |
| Performing Local Transport Searches | 176 |
| Running Our Code | 177 |
| Improving the "Move Me" Example Application | 179 |
| Dealing with Other Transport Possibilities..... | 180 |
| Limitations to our Approach..... | 181 |
| Introducing Transit Data Resources..... | 181 |
| Making Use of Transport Schedules and Timetables..... | 181 |
| Exploring GTFS Examples | 184 |
| Summary | 185 |
| Chapter 11: Pushing the Limits with Audio and Video | 187 |
| Audio for Mobile Web Apps..... | 187 |
| Utilizing the HTML5 audio Tag | 188 |
| Integrating Audio into Who's That Tweet? | 189 |
| Working with Audio Codecs | 190 |
| Using the Audacity Audio Editor..... | 192 |
| Audio Data API | 196 |
| Adding Video to Mobile Applications..... | 197 |
| Using the HTML5 video Tag | 197 |

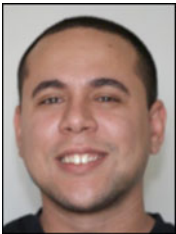
| | |
|--|------------|
| Codecs | 199 |
| Using Handbrake to Transcode Videos | 200 |
| Exploring on Your Own: Music Service APIs | 202 |
| “Scrobbling” Tracks to Last.fm | 203 |
| Tapping into the Power of Amazon’s Product Advertising API..... | 207 |
| Summary | 209 |
| Chapter 12: Supercharging the User Experience with AJAX | 211 |
| What Is AJAX? | 211 |
| Asynchronous? | 211 |
| So What About the JavaScript and XML? | 215 |
| AJAX—of—the—Day | 216 |
| My News! | 220 |
| First: Create a Pipe | 221 |
| Second: Get the Output and Display It! | 224 |
| Username Availability | 226 |
| AJAX Considerations | 229 |
| POST vs. GET | 230 |
| Setting Asynchronous to False? | 230 |
| Summary | 231 |
| Chapter 13: Packaging Your Applications | 233 |
| Compressing Your Application | 233 |
| What Is Compression? | 234 |
| Compression Tools and Utilities | 236 |
| Finding a Hosting Solution | 241 |
| Evaluating Hosting Providers | 241 |
| 1and1 | 245 |
| File Transfer Protocol | 246 |
| Deploying an Application Using Secure FTP | 248 |
| Versioning Your Software | 252 |
| Going Native | 254 |
| PhoneGap | 255 |
| Titanium Mobile | 258 |
| Closing Time | 259 |
| Index | 261 |

About the Authors



Jon Westfall is a researcher and technologist working in New York City at Columbia Business School. His current appointment is as the Associate Director for Research and Technology at the Center for Decision Sciences, a center within Columbia Business School at Columbia University in New York City. Additionally, he holds an appointment as a Lecturer in the Columbia University Psychology department, periodically teaching a course in Judgment and Decision Making. In addition to his research, he also has career ties in information technology, where he has worked as a consultant since 1997, founding his own firm, Bug Jr. Systems. As a consultant, he has developed custom software solutions (including native Windows 32 applications,

Windows .NET applications, Windows Phone 7 and Android mobile applications, as well as ASP, ASP.NET, and PHP web applications). He has also served as a senior network and systems architect and administrator (on both Windows and Unix networks, and hybrids) and has also been recognized as a Microsoft Most Valuable Professional (MVP) 2008 – 2012. In his spare time, he enjoys writing both technical books as well as fiction. His novel, *Mandate*, as well as other writings are available via his website at <http://jonwestfall.com>. He can be contacted through his website, and followed on Twitter (@jonwestfall).



Rocco Augusto is a Web Developer based out of the City of Roses – Portland, Oregon. Rocco first started dabbling in the art of Web Design and Development in middle school when the code bug bit him and refused to let go. When not tinkering away at some new markup or design idea you can usually find Rocco enjoying some time with his beautiful wife, daughters, and their unruly puppies.



Grant Allen has worked in the IT field for over 20 years, as a CTO, enterprise architect, and database administrator. Grant's roles have covered private enterprise, academia, and the government sector around the world, specialising in global-scale systems design, development, and performance. He is a frequent speaker at industry and academic conferences, on topics ranging from data mining to compliance, and technologies such as databases (DB2, Oracle, SQL Server, MySQL), content management, collaboration, disruptive innovation, and mobile ecosystems like Android. His first Android application was a task list to remind him to finish all his other unfinished

Android projects. Grant works for Google, and in his spare time is completing a Ph.D on building innovative high-technology environments. Grant is the author of *Beginning DB2*, and lead author of *Beginning Android 4*, *Oracle SQL Recipes*, and *The Definitive Guide to SQLite*.

About the Technical Reviewer



Steven Hughes has been a Microsoft Windows Phone MVP for the past decade for his passion and dedication in the mobile community. Steven became involved with handheld computers since the early '90s including the beta testing and the prototype design of several hardware and software designs. His passion and knowledge of mobile technology and the mobile industry has advised and consulted many on its use and has earned the nickname 'fyiguy' as result. Steven loves to share information and help people; you may see his contributions and articles on several websites, publications, podcasts, and other productions pertaining to mobile technology. Steven is also the Chief News and Review Editor for BostonPocketPC.com and has written several

detailed reviews and articles on various facets of mobile technology as well. Steven is a Moderator in the Microsoft Answers forums and also co-manages the New England Windows Phone User Group. Steven is employed as a Biomedical Engineer for the VA New England Healthcare System. When he has some free time he generally spends it with his family or outdoors playing soccer, hitting the slopes, strumming his guitar, catching a movie in his self-constructed custom home theater, or riding the trails on his mountain bike.

Acknowledgments

I'd like to acknowledge my editors here at Apress (Mark Beckner, Adam Heath, Chris Nelson, & Jonathan Gennick), as well as my co-authors Rocco & Grant and technical editor, Steven, for their hard work in this project. On a personal note, the support given to me by my wife, Karey, my parents, Alan & Dianne, and my extended family (especially Dan, Sue, Greg, Scott, & Mark) cannot be overstated. I'm also grateful for the support of my mentors, Eric Johnson and Elke Weber, my colleagues, Cindy Kim, Margaret Lee, Ye Li, Christoph Ungemach, Soo Baik, Galen Treuer, and Min Bang, and my current and former interns, Katherine Chang, Meaghan Gartner, Mary Reiser, Yechao Liu, Soo Jung Lee, & Nina Rouhani. Finally, I'd like to thank my friends who encouraged me to become as geeky as I am, directly and indirectly. This includes Steve Jocke, Tony Rylow, Ashley Newman, Maria Gaglio, Marie Batteiger, JD Jasper, Jason Dunn, Don Sorcinelli, Eric Hicks, Darius Wey, Jack Cook, Johan van Mierlo, Annie Ma, Holly Feiler, Dot Bertram, & Cathy Bischoff.

—Jon Westfall

Introduction

Both of the first author's (Jon's) parents were artists. They each could draw fantastical pictures that resembled real life, and were shocked to see that their son could barely muster up a stick figure. If you've always felt that your inner artist was best expressed through what you could build with the help of a computer and the Internet, then this book can guide your virtual paintbrush. The finished product? A mobile web application for Android devices, which can in turn inspire creativity and productivity in millions of prospective users. It is our hope that this book will give you all that you need to get up and running and creating your masterpieces in no time.

Who This Book Is For

This book is written at a beginner's level. For the most part, we assume nothing as we write about everything from what HTML is to how to apply CSS to querying databases and displaying content using JavaScript. For some, this may mean that they would like to skim certain introductory materials (and assuredly miss many bad jokes). However, even advanced users will likely gain something from the tricks we unroll our sleeves to reveal.

How This Book Is Structured

We've split the content in this book into several chapters, with three "unofficial" parts.

In the **first part**, we introduce you (Chapter 1) to the basic languages of the web: HTML, CSS, JavaScript, and more. We then jump into two applications (Chapters 2–3) quickly to get your feet wet, and then back out to discuss planning concerns you might need to address when designing your own apps (Chapters 4–6).

In the **second part**, we start to jazz things up a bit. We go into building impressive user interfaces (Chapter 7) and working with visual content (Chapter 8). We then show you two more applications (Chapters 9–10) that speak to the unique nature of mobile applications: Using location information to guide your apps (and users), as well as tapping into the cloud for information and data.

Finally, in the **last part**, we talk about the next level of interactivity to add to your applications. We touch on adding audio and video (Chapter 11), doing things behind the user's back to provide impressive functionality (Chapter 12) and wrapping it all up and uploading to the web or building a full app for your formerly browser-bound creation (Chapter 13).

While we've grouped chapters into a logical order, after Chapter 1 you should feel free to explore the rest of the content. While many topics build upon one another, reading what interests you first may help you get a good grasp of what concepts from earlier chapters you'll definitely want to check out. At the same time, there are nuggets of information in each chapter that will stand upon their own, especially discussions on design, psychology, and user experience! We hope you enjoy the journey!

Downloading the code

The code for the examples shown in this book is available on GitHub at <https://github.com/jonwestfall/Beginning-Android-Web-Apps-Development>.

Contacting the Author

We're always happy to hear from our readers, and if you have questions, comments, or thoughts about the book (or life in general), you can contact any of us through our personal websites or social media.

Jon Westfall: <http://jonwestfall.com>

Twitter: @jonwestfall

Rocco Augusto: <http://nerdofsteel.com/>

Twitter: @therocco

Grant Allen: <http://www.artifexdigital.com>

Twitter: @fuzzytwtr

Harnessing the Power of the Mobile Web

Welcome to the first chapter of this book. In this chapter, we'll endeavor to not only tell you about what you'll find in this book, but to also compare it to what has come before. You see, quite simply, it is only now that the true power of mobile web applications and mobile-optimized websites is being realized, despite the existence of the "web" on mobile phones in some form for 10 years.

Before we show off the neat stuff we have planned for this book, it's probably best to make sure everyone is on the same page, lingo-wise. So we'll start talking about the basic terms in web design. In the second section, we'll talk about the precursors to today's mobile web. And finally, in the last section, we'll talk about the concepts that will guide this book and give you a sneak peek at some of the applications we'll be developing!

Basics of Web Design

There are a few concepts that it's best to discuss up front. Forgive us if you've heard this all before. However, if you're completely new to web design (i.e., you've never written a single web page or blog), then this should be a good place to start. And, if we're starting at the beginning, then we should start with the lingua franca of the web: HTML.

Getting Started: HyperText Markup Language (HTML)

In the late 1980s, the computer language we know today as HTML was born. HTML isn't a true programming language, per se, in that it isn't compiled. Rather, HTML is interpreted by special software called a **web browser**. Browsers, such as Microsoft Internet Explorer, Mozilla Firefox, and Google Chrome on your Desktop computer, and Dolphin HD, Opera Mini, and Skyfire on your Android device, download HTML files from a **web server**, interpret them, and display them. This entire process is fairly simple. A

web server can be any sort of computer that makes a list of files available to other computers on the network over something called HyperText Transport Protocol (HTTP, as in `http://` at the beginning of web addresses, which are also called **URLs**). Browsers download these HTML files over HTTP and read them, looking for special features known as **tags**. These tags function in the same way as tags in older word processor programs did—specifying how text and other elements of the page should look when displayed in the viewer. Consider the web page in Figure 1–1.

This is normal text. However let's get fancy and make **this bold** (this is *italicized*).
The tag to the left just made this a new line.

The tag to the left here just made this a new paragraph.

Figure 1–1. An example web page named *hello.html*

Let's look at the HTML code that made up the page shown in Listing 1–1:

Listing 1–1. *hello.html*

```
<html>
<head>
<title>This is the text that appears in the browser's Title bar!</title>
</head>
<body>
This is normal text. However let's get fancy and make <strong>this bold</strong> (this
is <em>italicized</em>).
<br /> The tag to the left just made this a new line.
<p> The tag to the left here just made this a new paragraph.</p>
</body>
</html>
```

The code might look a bit strange, but let's walk through it line by line. The first line, which simply reads `<html>`, lets the browser know that it's reading an HTML document. You'll notice that the last line of the document, `</html>`, is similar. This line “finishes” the HTML object—closing the tag and telling the browser that the page is over. By having sets of tags like this, the browser knows what formatting to apply and where to stop applying it.

The second through fourth lines of the code are known as the page header. This is where programmers store important information that the browser needs to know in order to format the page properly. In this case, the only tag I've placed within the header is a `<title>` tag, which specifies what should be shown in the title bar of the user's web browser. The header would be the location where one would most commonly find certain documents, such as Cascading Style Sheets, JavaScript, and META information for search engine optimization, special instructions for different browsers, favicons (those little icons that appear next to a bookmark entry in your browser), and other important information about the page that is not related to the documents' content, which brings us to line 5 - the bodytag.

The bodytag tells the browser that the content to display to the user is about to be given. From here, we see straight text—the same that's in the rendered page shown in Figure 1–1. However, you'll notice a few special tags we've added in. The first, ``, tells the browser that the text between it and its end tag `` should be in bold to give it a

stronger visual oomph. A second tag, ``, does the same by emphasizing the content or by making the content italic.¹ A third tag, `
`, starts a new line (br stands for line break!). The `
` tag is a little different than most HTML tags. Since the tag does not require itself to enclose content on the page in the same that the `` and `` tags do, this tag closes on itself. Finally, the `<p>` tag starts a new paragraph.

At their cores, all web pages are some form of HTML, although most we'll discuss in this book are much more complicated. Thankfully, we'll walk you through them, so you won't be overwhelmed!

If this is your first outing into the world of HTML and web applications, then it would probably be a good idea to familiarize yourself with the basics of HTML before jumping full on into the book. One of the best resources on the Internet for learning HTML and browsing through basic code examples can be found at the W3Schools (<http://www.w3schools.com/>). Once you've gotten your feet a little wet with HTML, or in case you're already soaked from the neck down, it would be time to move on to some of the more intermediate portions of web application design and technologies that we will be using in this book.

Getting Stylish: Cascading Style Sheets (CSS)

Imagine that you're writing up a simple web page to aid in your parenting—a family chore list. It might look something like the list in Figure 1–2.

Family Chore List

- **Tommy:** Take out the trash
- **Beth:** Clean out the fridge.
- **Mittens:** catch mice.

Figure 1–2. Family Chore List

By just glancing at the finished product, there does not appear to be a lot going on here. We have a standard boring black and white document that completely lacks any style or individuality. Let us take a look at the code behind the scenes shown in Listing 1–2.

Listing 1–2. *chores.html*

```
<html>
<head>
<title> Family Chore List </title>
</head>
<body>
<h1>Family Chore List</h1>
```

¹ It's worth noting that the `` and `<i>` tags you may be used to were used in HTML 4 for the same purpose as `` and `` respectively. Their use has been deprecated in HTML5 in favor of the tags above.

```

<ul>
<li><strong>Tommy</strong>: Take out the trash</li>
<li><strong>Beth</strong>: Clean out the fridge. </li>
<li><strong>Mittens</strong>: catch mice. </li>
</ul>
</body>
</html>

```

Let us break down the teensy morsel of code within the bodyelement. Here, the unordered list on the page is created using the `ul` tag. An unordered list is great to use anytime you want to create a bulleted list of items. If your list requires a little more order, you might opt to use the `ol`, or ordered list, HTML tag.

While the page is fairly nice and simple, you might want to spice it up. Perhaps around Christmas time, you'd like splash some color on your family chores page that would make even the most bah humbug elf smile with glee (see Figure 1–3).



Figure 1–3. Christmas Chore List with green and red adding a holiday feel

Perhaps on the Fourth of July, you might want to fill your family with patriotic gusto (see Figure 1–4).



Figure 1–4. Patriotic Chore List with the red, white, and blue

Each time we change the colors, we modify the HTML source code by adding in appropriate tags. Take a look at the patriotic version of chores.html in Listing 1–3.

Listing 1–3. *patriotic chores.html*

```

<html>
<head>
<title> Family Chore List </title>
</head>
<body bgcolor=blue>
<font color=red><h1>Family Chore List</h1></font>
<font color=white>
<ul>
<li><strong>Tommy</strong>: Take out the trash</li>
<li><strong>Beth</strong>: Clean out the fridge. </li>
<li><strong>Mittens</strong>: catch mice. </li>
</ul>

```

```
</font>
</body>
</html>
```

Making modifications straight to the HTML is fine for small pages. However, imagine how much time adding those font tags might take if there were 12 children and countless pets to coordinate. Or perhaps you have multiple pages, one for each child and you don't want them to feel left out if their sibling has nice color combinations and they don't. Never fear—we can use something called a Cascading Style Sheet, or CSS, to keep it all in check. Basically, a CSS file is a small document consisting of a set of styles to be applied to an HTML document that can be changed at anytime, affecting every page it is connected to, without ever having to edit the original HTML document(s). Listing 1–4 provides an example CSS file.

Listing 1–4. *patriotic.css*

```
body {background-color: blue}
h1 {color: white}
li {color: red}
```

Notice how the format of the file is simply the HTML tag you wish to edit (H1 for example and the attributes you'd like to give it). In this case, we want the color of text within h1 to be white. We can simplify chores.html to include a link to this CSS file, as shown in the code of Listing 1–5.

Listing 1–5. *chores.html with CSS reference*

```
<html>
<head>
<title> Family Chore List </title>
<link rel="stylesheet" type="text/css" href="patriotic.css" />
</head>
<body>
<h1>Family Chore List</h1>
<ul>
<li><strong>Tommy</strong>: Take out the trash</li>
<li><strong>Beth</strong>: Clean out the fridge. </li>
<li><strong>Mittens</strong>: catch mice. </li>
</ul>
</body>
</html>
```

We'll get exactly the same output as is shown in Figure 1–4. Now, imagine how this works if we scale upward. First of all, the parents no longer need to edit the HTML tags directly to change styles. Depending on the holiday, they simply could have multiple CSS files they could link to (simply changing the fourth line of the code in Listing 1–5). Second, they could extend the CSS even further to specify spacing, fonts (Mittens hates serifs), and more. Finally, if they have more than one page, they could simply link the CSS sheet at the top of each page to their current "theme" and all the pages would look alike. While the examples above are extremely simple, they illustrate the power of CSS. We'll examine CSS in more detail as we continue through the book!

Getting Interactive: JavaScript

Sometimes great design isn't enough to make your point. Sometimes you want to do something a bit flashy, or something unique, or something downright useful. One of the simplest ways to do that is by using JavaScript. JavaScript is a scripting language that runs inside the viewer's web browser. For example, perhaps you've gone to a site before and gotten a pop-up message like the one in Figure 1-5.

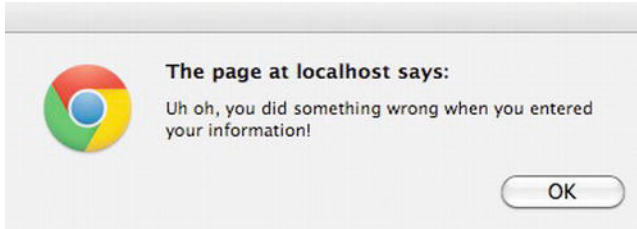


Figure 1-5. A JavaScript warning

Typically, you see these messages while filling out a forms page or perhaps in an online shopping cart telling you that your item is out of stock or some such annoying message. While you might be used to seeing these messages on web pages on your computer, they can also be shown in a mobile web browser (see Figure 1-6).

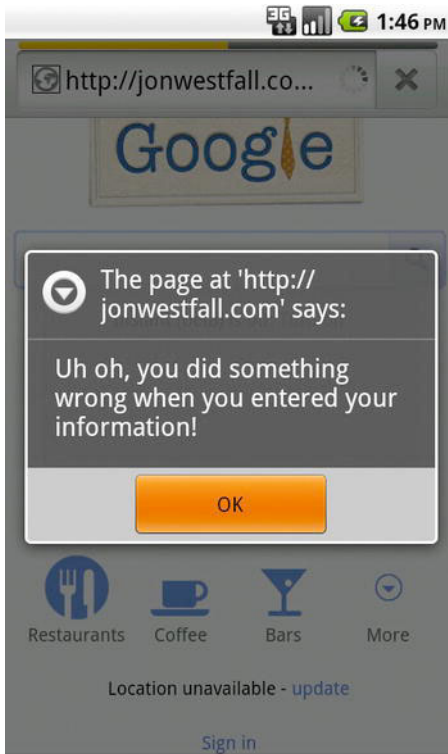


Figure 1-6. A JavaScript warning on an Android phone

The code that creates these messages is remarkably simple. Listing 1–6 integrated the code into the chores.html page we saw in the CSS example above.

Listing 1–6. *chores.html with JavaScript reference*

```
<html>
<head>
<title> Family Chore List </title>
<link rel="stylesheet" type="text/css" href="patriotic.css" />
<script type="text/javascript">
function ShowWarning() {
alert("Mittens - your mousing numbers are down this week - NO CATNIP FOR YOU");
}
</script>
</head>
<body onload=ShowWarning();>
<h1>Family Chore List</h1>
<ul>
<li><strong>Tommy</strong>: Take out the trash</li>
<li><strong>Beth</strong>: Clean out the fridge. </li>
<li><strong>Mittens</strong>: catch mice. </li>
</ul>
</body>
</html>
```

Let's start by talking about the new section of code right below the CSS link, inside the headsection, with the tag script. The scripttag tells the browser that a section of scripting code (in this case, of the type text/javascript) is about to be given. The browser then interprets the code. Since it's in the headsection, the browser simply stores this code for later use. This piece of code is called a function, which you can think of as a list of commands wrapped up in a "shortcut". Here the command is another function named alert. As you can imagine, JavaScript functions can get quite complex, with functions including other functions and interacting with user input.

Once the function is loaded into the browser's menu, we need to tell the browser when we want to execute it. In this case, I've changed the bodytag to include the line onload=ShowWarning();. This tells the browser that, when the page is loaded, I want it to run the function ShowWarning. The two parentheses indicate a spot where I could include information to pass to the function. This becomes useful for creating things like calculators or for checking input in a form. For example, I could write up something like Listing 1–7.

Listing 1–7. *chores.html with JavaScript reference passing a variable*

```
<html>
<head>
<title> Family Chore List </title>
<link rel="stylesheet" type="text/css" href="patriotic.css" />
<script type="text/javascript">
function ShowWarning(catname) {
alert(catname + " - your mousing numbers are down this week - NO CATNIP FOR YOU");
}
</script>
</head>
<body onload=ShowWarning("Mittens");>
<h1>Family Chore List</h1>
```

```
<ul>
<li><strong>Tommy</strong>: Take out the trash</li>
<li><strong>Beth</strong>: Clean out the fridge. </li>
<li><strong>Mittens</strong>: catch mice. </li>
</ul>
</body>
</html>
```

The code in Listing 1–7 will produce the exact same message as the code in Listing 1–6. However, in Listing 1–7, I’ve passed the feline’s name as a variable. The function `ShowWarning` now expects that I’ll pass a variable to be named “`catname`”, and it can use that information in its code. When I call `ShowWarning()` in the `bodytag`, I simply add the cat’s name to be passed to the function. I can pass more than one thing, if I want to. As mentioned, this could get quite complex, depending on how much I want to chastise poor Mittens.

As you can see, coupling JavaScript along with HTML and CSS can produce pages that look good, are easy to update, and can interact with the user. But sometimes you might need to produce a document that doesn’t give style information—it just gives general information. A prime example of this is given in the next section, as we start to get into the wonderful world of XML!

Getting Informative: Extensible Markup Language (XML)

If you spend any time on the Web, you may have noticed an odd little icon on some pages that looks something like this.



Figure 1–7. *An RSS icon*

This little orange icon tells the reader about an RSS feed that the current website has available. RSS feeds look pretty uninteresting and unintelligible to a user (take a look at Figure 1–8 for the start of an RSS feed). However, other web pages and scripts can use them to grab a lot of information from one source and display it in different ways to the user.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" media="screen" href="/~d/styles/rss2full.xsl"?><?xml-stylesheet typ
<channel>
  <title>JonWestfall.Com</title>
  <link>http://jonwestfall.com</link>
  <description>More Than You Want To Know!</description>
  <lastBuildDate>Thu, 16 Jun 2011 13:21:06 +0000</lastBuildDate>
  <docs>http://backend.userland.com/rss092</docs>
  <language>en</language>
  <!-- generator="WordPress/3.1.3" -->

  <atom10:link xmlns:atom10="http://www.w3.org/2005/Atom" rel="self" type="application/xml" href="
    <title>Keep your Android Tablet Up To Date Daily With Increased Battery Life!</title>
    <description>Recently I found that I had a problem with my Samsung Galaxy Tab (Although you
    &lt;p>&lt;a href="http://feedads.g.doubleclick.net/~a/EFCCCh7JV-yrNNULtsPwYhvAs_A/0/da"&lt;img src="ht
    &lt;a href="http://feeds.feedburner.com/~ff/jonwestfall?a=FFCZ10Xe7kc:Yhg4XtccZAQ:y1l2AUoC8zA"&lt;img
    &lt;div>&lt;img src="http://feeds.feedburner.com/~r/jonwestfall/~4/FFCZ10Xe7kc" height="1" width
    <link>http://feedproxy.google.com/~r/jonwestfall/~3/FFCZ10Xe7kc/</link>
    <feedburner:origLink>http://jonwestfall.com/2011/05/keep-your-android-tablet-up-to-date-
  </item>
```

Figure 1–8. The start of a blog's RSS feed, showing new entries

For example, Figure 1–9 is the beginning of the RSS feed for my personal blog. Each element contains a variety of data that isn't very pretty to look at but provides all the information one might want to view my blog in a special piece of software called an RSS reader. While certain applications, like Microsoft Outlook, have built-in RSS readers, many prefer to use a dedicated reader client. One popular RSS reader is Google Reader, which can take the link to my RSS feed and produce a nice view of my blog so that the Google Reader user can quickly see what articles I've posted recently.

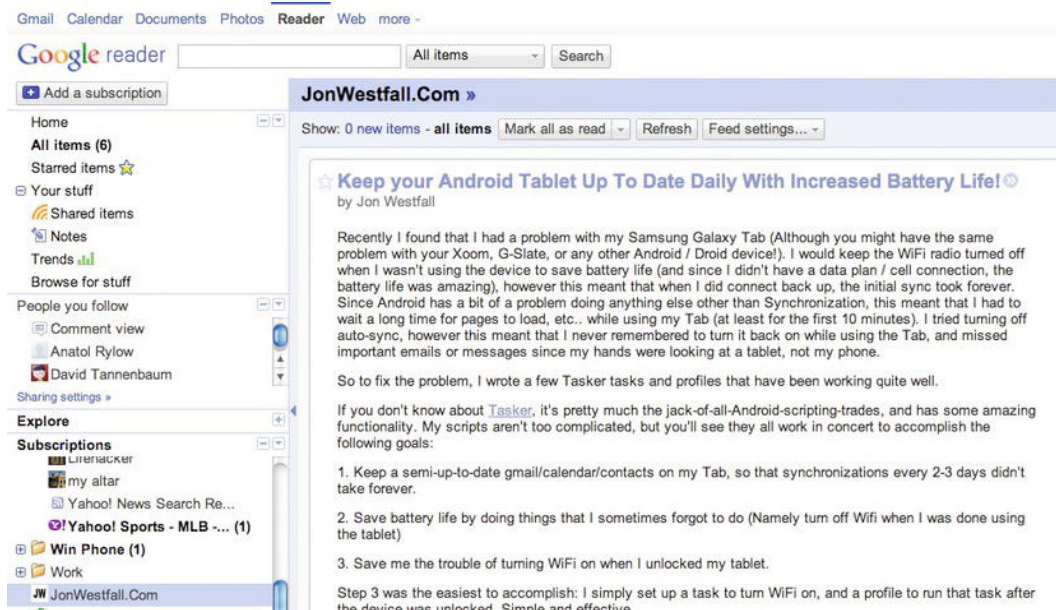


Figure 1–9. My personal blog, displayed within Google Reader

Now, you might be asking why I'd want people to view my website somewhere other than at its usual web address. The simple answer is that it might be more convenient for my users to view all the blogs they read (mine and others) within one piece of software. Software, such as Google Reader, can keep track of hundreds of RSS feeds, from news sources, blogs, and even just simple status updates like my Twitter feed. All of these pieces of information are retrieved by Google Reader in a format known as Extensible Markup Language (XML). XML isn't a format you'd want to have your human viewers see, but it is one that you'd want to use if you were sharing information between web pages or between web services.

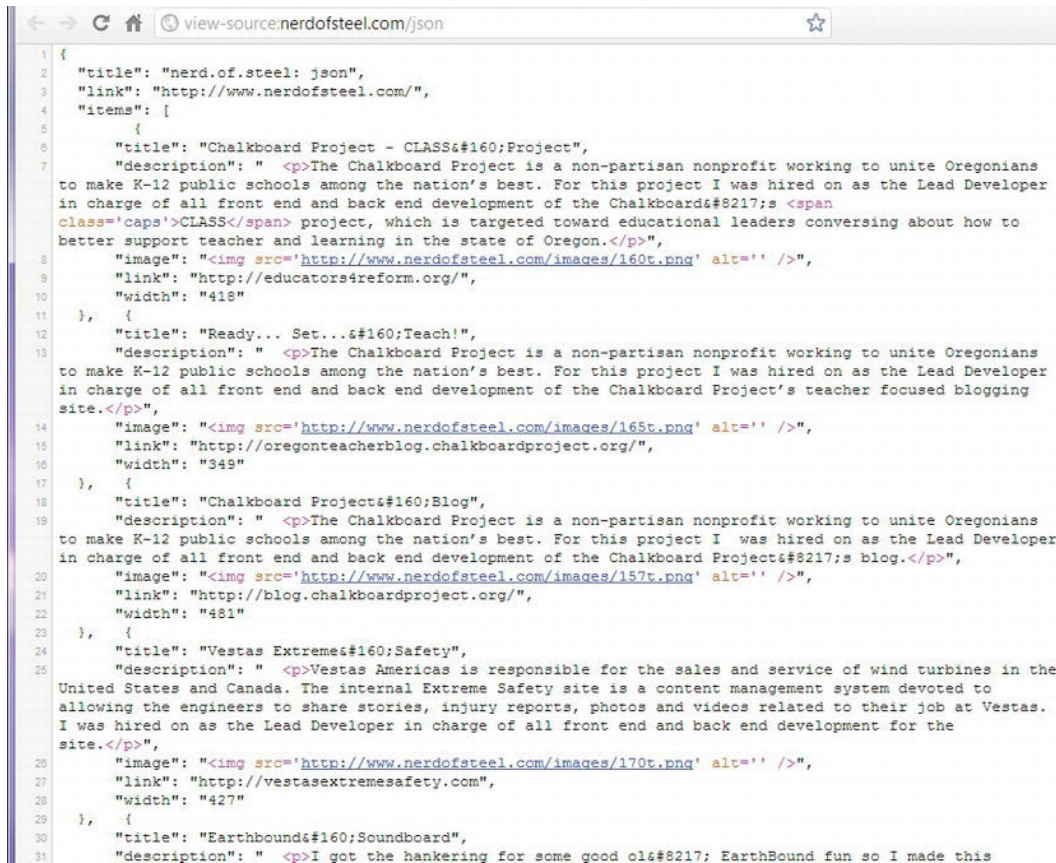
While the example above shows XML as an output, the web application that powers my blog (WordPress) produces the XML so other sites like Google Reader can use it. XML can also be used as an input. For example, I might want to take data (such as sports scores) and display them on my webpage. Most likely, those sports scores will be available in XML, which my web page can then open and **parse**. Parsing is simply a fancy term that means "read, interpret, and display". My webpage will read the scores, interpret them if necessary (i.e., calculate something, aggregate something), and then display them to the user in some meaningful way.

So to recap, we've now seen how to build a basic webpage, how to make it look pretty (easily), and how to make it interact with a user. Finally, we talked about how webpages and programs get data between each other by using XML. As we move through the book, we'll talk in depth about each of these areas and give you plenty of examples of how to use them. In fact, coming up in Chapter 2, we'll discuss how to get data from a very popular web service and display it in the first full application we'll create!

JSON: Human-Readable Data Interchange

If you have a brilliant idea for a mobile web application that relies on the application programming interface, or API, of other services, such as Twitter or Foursquare, then chances are you will be quickly introduced to JSON (JavaScript Object Notation), which is one of my favorite technologies.

JSON is a human-readable, super-lightweight data interchange technology that is based on JavaScript. Basically, it is a JavaScript object that can be used to transmit simple data structures and arrays of information. When it comes to working with external data in your web application, I have fallen in love with JSON for its ease of use when compared to other technologies such as XML. As with all technologies though, your mileage may vary. Figure 1–10 shows an example of what a JSON document would look like.



```

1 {
2   "title": "nerd.of.steel: json",
3   "link": "http://www.nerdofsteel.com/",
4   "items": [
5     {
6       "title": "Chalkboard Project - CLASS:Project",
7       "description": " <p>The Chalkboard Project is a non-partisan nonprofit working to unite Oregonians
8       to make K-12 public schools among the nation's best. For this project I was hired on as the Lead Developer
9       in charge of all front end and back end development of the Chalkboard's <span
10      class='caps'>CLASS</span> project, which is targeted toward educational leaders conversing about how to
11      better support teacher and learning in the state of Oregon.</p>",
12      "image": "<img src='http://www.nerdofsteel.com/images/160t.png' alt='' />",
13      "link": "http://educators4reform.org/",
14      "width": "418"
15    },
16    {
17      "title": "Ready... Set...Teach!",
18      "description": " <p>The Chalkboard Project is a non-partisan nonprofit working to unite Oregonians
19      to make K-12 public schools among the nation's best. For this project I was hired on as the Lead Developer
20      in charge of all front end and back end development of the Chalkboard Project's teacher focused blogging
21      site.</p>",
22      "image": "<img src='http://www.nerdofsteel.com/images/165t.png' alt='' />",
23      "link": "http://oregonteacherblog.chalkboardproject.org/",
24      "width": "349"
25    },
26    {
27      "title": "Chalkboard Project:Blog",
28      "description": " <p>The Chalkboard Project is a non-partisan nonprofit working to unite Oregonians
29      to make K-12 public schools among the nation's best. For this project I was hired on as the Lead Developer
30      in charge of all front end and back end development of the Chalkboard Project's blog.</p>",
31      "image": "<img src='http://www.nerdofsteel.com/images/157t.png' alt='' />",
32      "link": "http://blog.chalkboardproject.org/",
33      "width": "481"
34    },
35    {
36      "title": "Vestas Extreme:Safety",
37      "description": " <p>Vestas Americas is responsible for the sales and service of wind turbines in the
38      United States and Canada. The internal Extreme Safety site is a content management system devoted to
39      allowing the engineers to share stories, injury reports, photos and videos related to their job at Vestas.
40      I was hired on as the Lead Developer in charge of all front end and back end development for the
41      site.</p>",
42      "image": "<img src='http://www.nerdofsteel.com/images/170t.png' alt='' />",
43      "link": "http://vestasextremesafety.com",
44      "width": "427"
45    },
46    {
47      "title": "Earthbound:Soundboard",
48      "description": " <p>I got the hankering for some good ol' EarthBound fun so I made this

```

Figure 1-10. Rocco Augusto's portfolio in JSON format from nerdofsteel.com

The Mobile Web, Yesterday and Today

Many of us first started using the Internet in the late 1990s or, if you weren't alive in the late 1990s, perhaps you've used it your whole life! While we might be very familiar with the Internet on our desktop, getting it onto a small screen that can fit in our pocket might seem a bit strange, especially with different jargon and marketing-speak that is often heard when it comes to the Web. We'll start by discussing how fast the data arrives on your phone and then what sorts of data can be sent.

Knowing the Speeds (Or "What is 3G anyway?")

Often in commercials for the latest smartphone, you'll hear a number such as "4G" or "faster than 3G". The "G" stands for the generation of the technology. You rarely hear about 2G, second generation, and there is a good reason for it. The onslaught of data onto a smartphone coincided with the emergence of the third generation of cellular network data standards. However, 1G and 2G did exist and, if you owned the first

iPhone (Released in 2007), you only had 2G speed, using protocols including GPRS, EDGE, and 1X. Data coming to you over 2G was just about twice as fast as a dial-up modem, around 115Kbps. So while email and text-based web pages would load reasonably fast, anything with too many images or multimedia would take approximately eternity².

Around 2001, the initial designs for what we consider 3G (or third generation) data networks were drafted and could reach theoretical speeds of over 7Mbps. These networks, which include protocols like UMTS, WCDMA, and EV-DO, can move data much faster than their 2G counterparts. For the first time, this allowed for innovations such as streaming movies and music directly to a phone. The limiting factor in showing complex web pages was no longer the speed of the data connection but the speed of the phone. By 2007, most telecom providers had adopted and “rolled-out” a 3G network and devices, such as mobile broadband cards, became common.

In the past few years (2008-2010), new and improved versions of current 3G technologies have become available. While there is considerable argument about what exactly the differences between the 3G technologies we were privy to before and this newer 3G-based technology that is being dubbed “4G” are, it is obvious that newer protocols, such as HSPA, HSPA+, WiMAX, and LTE, are faster than their 3G predecessors. Unfortunately, while all of the major carriers are gradually moving forward with their plans to increase data speeds and network capacity, those updates and changes will not become immediately apparent to the end user until they purchase a phone with the right internal hardware to take advantage of these numerous changes.

One common trap that many web developers who target mobiles may fall into, at least early on, is the notion that speed is all one needs to consider when developing a mobile app. If I know my users will need a 3G network to use the feature I’m developing, it might be tempting to skip streamlining other areas of the app, given the fact I know I’ll have a faster data connection. However, as mentioned above, speed isn’t the only factor one needs to consider. The actual processing and software power of the device (i.e., the capabilities of its web browser) can also slow down an app. If you want evidence of this, use your own Smartphone or Tablet on Wi-Fi instead of cellular data and observe how certain sites and apps still lag when connecting to the Internet, despite a connection much faster (generally) than 3G or 4G. We’ll discuss how to avoid programming bloated unresponsive apps as we continue through the book.

Languages and Protocols, Yesterday and Today

Now that we know how fast we can go, we should probably talk a bit about how web pages were shown to handheld users over the past 10 years and the current state of the mobile world.

In the beginning, handheld devices, such as the earliest modern personal digital assistants (PDAs), had no direct connection to the Internet. This meant that any content

² At least it felt that way.

the user wanted to read from the web needed to be downloaded first and then stored on the device or cached. A very popular service for this, AvantGo, operated for a little over a decade before closing up shop in 2009. While these services were somewhat annoying (in that you needed to manually synchronize your PDA physically to your computer regularly to get the content you wanted), they generally presented content in a very basic and easy to read manner. Ironically enough, this type of presentation has experienced a bit of a revival as users today find content while online and otherwise busy (i.e. at work) and wish to save it to read later. One popular service, Read It Later (<http://readitlaterlist.com/>), even has a mobile client that shows these saved web pages in a similar format to the old “offline cache” system popular in the late 1990s! Figure 1–11 shows a cached article on Read It later.

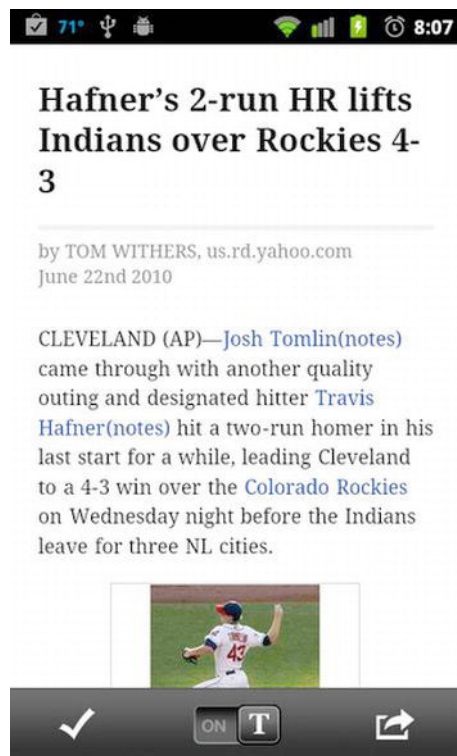


Figure 1–11. Read It Later, on Android, shows the cached version of a Yahoo! Sports article

As PDAs with built-in Wi-Fi radios and smartphones became available, users could connect directly to a web page and view its content. Before direct access to the Internet, however, many telecoms delivered phones with WAP browsers. WAP, or Wireless Application Protocol, was an extremely simple data service that allowed users to navigate simple menus to get to the information they wanted. These menus were typically all text, with perhaps 1–2 images here and there, and were designed to be quick portals to things like web-based email, perhaps movie times, or weather information. Phones with WAP browsers weren't connecting to the Internet, per se, as they could only view what their provider had put on the menu, but at least they could

view it wherever they had cell coverage, as opposed to downloading it and reading it offline.

A similar concept to WAP was the Java Platform, Micro Edition, and often abbreviated j2me. j2me allowed users to load small java applets on their phone that could connect to specific services in the same way WAP did, providing a similar experience. While j2me (Java Platform, Micro Edition) was available on phones 2-3 years ago, along with the ever-popular feature phone-friendly Brew MP operating system from Qualcomm, it was limited by odd security settings and precautions that providers might put on a phone. Oddly enough, it was available on phones that already had working web browsers that could go anywhere. This made one wonder why you might load up a special Gmail j2me applet when you could simply visit the mobile version of Gmail.

Finally, by about 2005, most smartphones on the market contained a fairly decent web browser that one could open, type in a URL, and view an actual web page. These web pages were typically one of the following two varieties: normal web pages crammed onto a smaller screen (see Figure 1-12) or specially created mobile versions of a website (see Figure 1-13). Both had their advantages and disadvantages. The normal page usually looked horrible on a small screen, with information flowing off the page, unreadable, or in technical terms, unrenderable (rendering is the process by which a web page is shown in a browser). While the information was usually horribly displayed, if one had enough patience and skill, one could usually find what they needed.

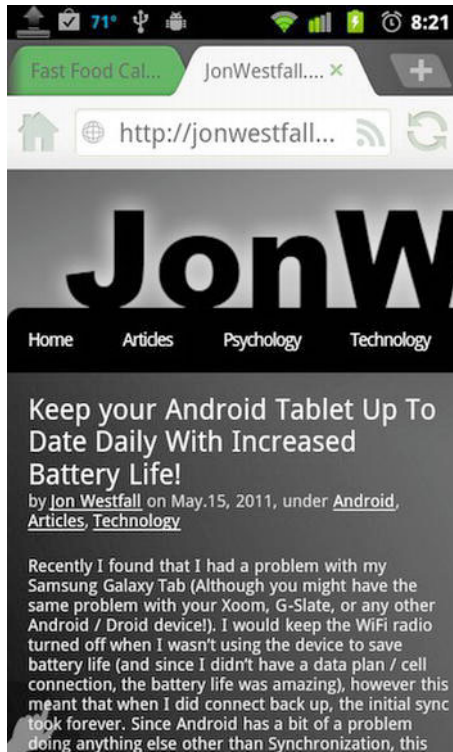


Figure 1-12. My personal blog, desktop-view, displayed in a mobile browser