Reaching Users on iPhone, Android, d more

2nd Edition
Includes HTML5

X003JSJ3F9

Programming the Mobile Web:...

Used - G

Programming the

Mobile Web



POSTSCRIPT 1016

O'REILLY®

001

Maximiliano Firtman

SECOND EDITION

Programming the Mobile Web

Maximiliano Firtman

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

Programming the Mobile Web, Second Edition

by Maximiliano Firtman

Copyright © 2013 Maximiliano Firtman. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Simon St. Laurent and Meghan Blanchette
Production Editor: Melanie Yarbrough
Copyeditor: Rachel Head
Proofreader: Kiel Van Horn
Indexer: Lucie Haskins
Cover Designer: Randy Comer
Interior Designer: David Futato
Illustrator: Rebecca Demarest

March 2013: Second Edition

Revision History for the Second Edition:

2013-03-15: First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449334970* for release details.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Programming the Mobile Web, Second Edition*, the image of a jerboa, and related trade dress are trademarks of O'Reilly Media, Inc.

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 O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-449-33497-0

[LSI]

For my parents, Stella Maris and Edgardo, my brother, Sebastián, Inés and Antonio, and my lovely wife, Ani, who have supported me during all of my projects

Table of Contents

Preface xvii

1. The Mobile Jungle 1
The Mobile Ecosystem 1
What Is a Mobile Device? 1
Mobile Device Categories 3
Brands, Models, and Platforms 9
Apple iOS 9
Android 13
Windows 17
Nokia 20
BlackBerry 24
Samsung 26
Sony Mobile 27
Motorola Mobililty 27
Amazon 28
LG Mobile 28
HTC 29
HP and Palm 29
Firefox OS 30
Ubuntu for Phones 30
Chinese Platforms 31
Other Platforms 31
Smart TV Platforms 31
Technical Information 32

2. Understanding the Mobile Web 35
Myths of the Mobile Web 35
It's Not the Mobile Web; It's Just the Web! 35

You Don't Need to Do Anything Special About Your Desktop Website 36
One Website Should Work for All Devices (Desktop, Mobile, TV) 36
Just Create an HTML File with a Width of 320 Pixels, and You Have a Mobile Website 37
Native Mobile Applications Will Kill the Mobile Web 37
People Are Not Using Their Mobile Browsers 37
What Is the Mobile Web? 38
Differences 38
Mobile Web Eras 39
WAP 1 39
WAP 2.0 41
The Mobile Browsing Experience 43
Navigation Methods 43
Zoom Experience 45
Reflow Layout Engines 46
Direct Versus Cloud-Based Browsers 47
Multipage Experience 48
Web Engines 49
Fragmentation 50
Display 50
Input Methods 58
Other Features 60
Market Statistics 60

3. Browsers and Web Platforms..... 63
Web Platforms That Are Not Browsers 64
HTML5 Web Apps 64
Web Views 67
Pseudo-Browsers 68
Native Web Apps, Packaged Apps, and Hybrids 69
Ebooks 72
Mobile Browsers 73
Preinstalled Browsers 73
User-Installable Browsers 82
Browser Overview 87

4. Tools for Mobile Web Development..... 89
Working with Code 89
Adobe Dreamweaver 89
Adobe Edge Tools 90
Microsoft Visual Studio and WebMatrix 91
Eclipse 91

Native Web IDEs 91
Testing 91
Emulators and Simulators 92
Real Device Testing 114
Remote Labs 116
Production Environment 124
Web Hosting 125
Domain 125
Error Management 125
Statistics 126

5. Architecture and Design..... 127
Mobile Strategy 127
When to Get Out of the Browser 127
Context 130
Server-Side Adaptation 131
Progressive Enhancement 132
Responsive Web Design 134
RESS 139
Navigation 140
Design and User Experience 141
Touch Design Patterns 145
Tablet Patterns 148
Official UI Guidelines 149
What Not to Do 149

6. Markups and Standards..... 153
First, the Very Old Ones 153
WML 154
Current Standards 158
Politics of the Mobile Web 159
Delivering Markup 160
XHTML Mobile Profile and Basic 164
Available Tags 165
Official Noncompatible Features 166
Creating Our First Compatible Template 167
Markup Additions 169
Mobile HTML5 169
Creating Our First HTML5 Template 170
Syntax Rules 171
New Elements 172
CSS for Mobile 172

WCSS Extensions 172
CSS3 176
HTML5 Compatibility Levels 177
Testing Your Browser 177

7. Basics of Mobile HTML5 181
The Document Head 181
Title 181
Website Icons 183
Home Screen Icons 187
The Viewport 200
Changing the Navigation Method 215
Removing Automatic Links 216
Metadata for Sharing 216
Hiding the URL Address Bar 218
Native App Integration 220
The Document Body 224
Main Structure 225
HTML5 Mobile Boilerplate 227
The Content 228
Block Elements 228
Lists 228
Tables 229
Frames 229
Links 230
Accessibility 232

8. HTML5 Forms 235
Form Design 235
Form Elements 238
Select Lists 238
Radio Buttons and Checkboxes 241
Buttons 242
Hidden Fields 243
Text Input Fields 243
Range Slider Fields 254
Date Input Fields 255
File Selection Fields 257
Noninteractive Form Elements 264
Form Control Attributes 266
Placeholder 266
autofocus 267

autocomplete 267
readonly 268
Input Validation Attributes 268
Safari Extensions 268
Firefox Extensions 268
XHTML Mobile inputmode 269
Additional Form Attributes 270
Form Validation 270
HTML5 Validation 271
WAP CSS Validation 275

9. Feature and Device Detection..... 279
Possible Problems 279
Possible Solutions 280
Informational Websites 281
Can I Use 282
MobileHTML5.org 282
WebPlatform.org 284
Client-Side Detection 284
HTML Fallbacks 284
CSS Fallbacks 285
Vendor Prefixes 286
JavaScript Fallbacks 290
Modernizr 292
Polyfills 297
Platform Detection 298
Server-Side Detection 300
HTTP 300
Detecting the Context 307
Cloud-Based Browsers 309
Mobile Detection 311
Transcoders 312
Device Libraries 313

10. Images and Media..... 331
Images 331
Image Formats 331
Using the img Element 336
Responsive Images 337
Local Pictograms 340
Dealing with Multiple Screen Densities 342
Using Vector-Based Solutions 343

Providing One Single Image 344
Providing Image Alternatives 346
SVG 350
Canvas 357
Adobe Flash 365
Video 368
Containers and Codecs 368
Delivering Video 370
The HTML5 video Element 370
Streaming 376
Embedding with object 378
Video Compatibility 378
Audio 379
Invisible Audio Player 380
Web Audio API 380
Audio Compatibility 381

11. CSS for Mobile Browsers..... 383
Where to Insert the CSS 383
Media Queries 384
CSS3 Media Queries 385
Selectors 392
CSS Techniques 393
Reset CSS Files 393
Text Formatting 395
Common Patterns 404
Display Properties 404
Rounded Corners 408
Border Image 409
Pseudoclasses 413
Backgrounds 413
Scrollable Areas 414
Content 415
Opacity 416
Cursor Management 416
Selection Management 417
Touch Callout 420
Highlight Color 421
Appearance Override 421
CSS Sprites 422
Samples and Compatibility 422
CSS Sprites Alternatives 427

CSS3 Modules 427
Gradients 428
Reflection 431
Masks 432
Transforms 433
Transitions 439
Animations 442
CSS Filter Effects 446
CSS Regions and Exclusions 447
New CSS Values and Units 448

12. JavaScript Mobile..... 451
Coding for Mobile Browsers 452
HTML5 Script Extensions 453
Code Execution 453
Cloud-Based Browsers 454
JavaScript Debugging and Profiling 455
Battery Consumption 455
Background Execution 456
Status Detection 457
Background Tab Notification Trick 460
Background Execution Compatibility 461
Push Notifications 462
Supported Technologies 463
The Document Object Model 463
The Selectors API 463
JSON 464
Binary Data 464
Web Workers 464
HTML5 APIs 466
Native Web App APIs 466
Standard JavaScript Behavior 467
Standard Dialogs 467
History and URL Management 470
Manipulating Windows 471
Focus and Scroll Management 472
Timers 473
Changing the Title 476
Cookie Management 476
Event Handling 477
JavaScript Libraries 484
Mobile Libraries 485

UI Frameworks 487
Sencha Touch 488
jQuery Mobile 489
Enyo 490
Montage 491
iUI 492
jQTouch 493
JavaScript Mobile UI Patterns 495

13. Offline Apps, Storage, and Networks. 501
Offline Web Apps 501
The Manifest File 502
Accessing Online Resources 503
Updating the Package 504
Deleting the Package 505
The JavaScript API 506
Compatibility and Limits 508
Client-Side Storage 511
Web Storage 511
The Web SQL Database API 514
The IndexedDB API 518
The FileSystem API 520
User Intervention 521
Debugging Storage 521
Network Communication 522
Ajax 522
Server Sent Events 525
WebSockets 527

14. Geolocation and Maps. 529
Location Techniques 529
Accuracy 529
Indoor Location 529
Client Techniques 530
Server Techniques 532
Asking the User 533
Detecting the Location 534
The W3C Geolocation API 534
Carrier Network Location APIs 539
IP Geolocation 540
Maps/Navigation App Integration 541
Google Maps for Android 541

iOS Maps 543
Bing Maps 545
Showing a Map 546
Google Maps API v3 546
Google Maps Static API 549
Nokia Here 550

15. Device Interaction..... 553
Mobile-Specific URIs 553
Making a Call 554
Sending Email 556
Sending an SMS 557
Other Communication Apps 558
Adding a Contact to the Phonebook 559
Integrating with Other Applications 560
JavaScript APIs 564
Touch 564
Gestures 572
Sensors 579
Network Information 583
File Management 585
Full Screen 587
Web Notifications 588
Camera 590
Battery 593
Vibration 593
Other APIs 594

16. Native and Installed Web Apps..... 597
Web App Pros and Cons 598
Architecture of a Web App 599
Meta Configuration 599
Platform Access 600
Data Storage 600
Network Access 600
Logic 600
User Interface 601
Packaging 601
Distribution 601
Standards 601
Packaging and Configuration Standards 601
Official Platforms 602

iOS Web Apps 602
Symbian Standalone Web Apps 612
Windows 8 Store Apps 613
Mozilla Open Web Apps 619
Chrome Apps 624
Samsung Web Apps 624
BlackBerry WebWorks Apps 624
Nokia S40 Web Apps 632
Nokia Symbian Web Apps 637
Apache Cordova/PhoneGap Apps 644
Template Creation 646
Cordova Web View 646
CordovaJS 647
Plug-ins 650
Distribution 650
iOS App Store 650
Android Stores 651
BlackBerry AppWorld 651
Microsoft Windows Store 651
Microsoft Windows Phone Store 651
Full-Screen App Patterns 652
Multiple Views 652
Layout 652
Input Method 652
One-View Widget 653
Dynamic Application Engine 653
Reload My Web App 653

17. Content Delivery..... 655
MIME Types 655
Static Definition 655
Dynamic Definition 657
File Delivery 658
Direct Linking 658
Delayed Linking 658
OMA Download 659
Application and Games Delivery 662
iOS Applications 663
Android Applications 665
Windows Applications 666

Java ME 667

18. Debugging and Performance..... 671
Debugging 671
Server-Side Debugging 671
Markup Debugging 676
Client-Side Debugging 678
Performance Optimization 686
Measurement 686
Best Practices 689

19. Distribution and Social Web 2.0..... 695
Mobile SEO 695
Spiders and Discoverability 696
Mobile Sitemaps 696
How Users Find You 698
SMS Invitation 698
Email Invitation 699
Mobile Tiny URL 699
QR Codes 699
NFC Tags 701
User Fidelizing 701
Web Shortcuts 702
RSS 702
Open Search 702
Apple Passbook 703
Mobile Web Statistics 704
Google Analytics for Mobile 704
Yahoo! Web Analytics 705
Mobilytics 705
Monetizing Your Website 705
Mobile Advertisements 705
Mobile Web Social Features 706
Authentication and Sharing APIs 706
Sharing Content 707

A. MIME Types for Mobile Content..... 709

Index..... 713

Preface

In your pocket is a device that has changed the lives of billions of people all over the world. The third personal screen (after the TV and the computer) is the most personal one, and bringing our services to it is one of the key business priorities of this decade.

Mobile development, however, is a more challenging activity than desktop development. Platforms are severely fragmented, and developers have to work with minimal resources. Fortunately, the mobile web makes it easier to deal with this fragmentation, allowing developers to create applications that run on many more platforms than native (or installable) applications. As we will see later, the mobile web and installable applications are not enemies. In fact, they work together very well.

All of that sounds great: billions of devices, web technologies, multiplatform solutions... where's the problem? More than half of your desktop web skills and the tips, hacks, and best practices you already know simply do not apply on the mobile web. The mobile web demands new usability patterns, new programming best practices, and new knowledge and abilities.

This is a second edition, prepared two years after the first one. At the time of the first edition there were almost no books, websites, or training courses focused on concrete mobile web programming. Today it's more common to find such information; however, it's not always good enough. We don't need vague information like "this may not work on some phones"; we need real, fresh, and working data. On which devices does a solution not work? Why? Is there another solution? That is why I've written this book: to help developers in programming mobile websites.

You may feel that you are advanced enough to go directly to the code, but I encourage you to start from the beginning of the book if you are new to the mobile world. This is another universe, and every universe has its own rules.

Who This Book Is For

This book is for experienced web developers who want to learn what's different about designing for the mobile web. We will talk about HTML, CSS, JavaScript, Ajax, and server-side code as if you have experience with all those technologies. If you are a web designer with some basic programming skills, you will also find this book useful.

We will cover HTML5 features, but don't worry if you don't have any experience with this platform yet; we will cover it from the ground up, and your HTML 4 and XHTML 1.0 knowledge will be enough.

If you are an individual freelancer, if you work for a company in the areas of programming or web development, or if you work in a web design studio, this book is for you. Perhaps you need to create a mobile application or client for a current desktop service, you want to add new services to your portfolio, or you need to migrate an old mobile website to newer devices.

You may also be a web entrepreneur with—or looking for—a great idea for mobile devices, and you want to analyze what you can do with current mobile browsers. This book investigates compatibility device by device and discusses advanced features you can implement.

The book will also be useful if you are wondering how to identify devices and deliver proper and compatible content for ad campaigns, to sell content or to deliver free content to mobile users.

Who This Book Is Not For

I don't really want to cut anyone out of the possibility of reading this book, but there are a lot of people who aren't likely to benefit from all of it. If you are a graphic designer, you will not find detailed tips and practices in this book, and you are likely to enjoy only the first four chapters.

If you are a web designer without programming skills, Chapter 1 through Chapter 7 are the ones you should read line by line; the rest will be useful to review so you know the capabilities you can request from a developer.

If you are a native mobile developer (iPhone, Android, Java mobile, Windows Phone), some web knowledge will be required in order to understand and follow all the samples in this book.

This is also not a book for learning basic HTML, CSS, or JavaScript. You will not find detailed samples or step-by-step instructions on how to implement every task. It is assumed that you are experienced enough to create code on your own, or at least know how to find out by searching the Web.

If you are a manager, a CTO, a project leader, or an entrepreneur without any web knowledge, you will find the first four chapters useful: they describe the current state of the art in this market and should help you decide how to organize your team.

What You'll Learn

This book is an advanced reference for the mobile web today, and it is the most complete reference available at this time. This may seem an ambitious claim, but it is the truth. This book draws upon a mix of experience and very detailed research and testing not available in other books, websites, or research papers about the mobile web.

Programming the Mobile Web will teach you how to create effective and rich experiences for mobile web browsers, and also how to create native web applications that will be installed in a device's applications menu.

We will not talk only about the star devices, like the iPhone and Android devices; we will also cover mass-market platforms from Nokia, BlackBerry, Amazon, Microsoft, and other vendors.

Other Options

The main challenge looking for books and online resources is finding solutions that contain information, not ones having plenty of "maybes," "perhapses," and "be carefuls."

If you need to learn web technologies, there are plenty of books and resources available. Take a look at *oreilly.com/css-html* and *oreilly.com/javascript* for some lists.

If you want a complement to this book in the areas of design, performance, and advanced programming, I recommend the following books:

- *Mobile Design and Development* by Brian Fling (O'Reilly)
- *Programming the iPhone User Experience* by Toby Boudreaux (O'Reilly)
- *JavaScript: The Good Parts* by Douglas Crockford (O'Reilly)
- *High Performance JavaScript* by Nicholas Zakas (O'Reilly)
- *High Performance Websites* by Steve Souders (O'Reilly)
- *Even Faster Web Sites* by Steve Souders (O'Reilly)
- *Website Optimization* by Andrew B. King (O'Reilly)
- *Mobile JavaScript Application Development* by Adrian Kosmaczewski (O'Reilly)
- *Mobile Usability* by Jakob Nielsen and Raluca Budiu (New Riders Press)

If You Like (or Don't Like) This Book

If you like—or don't like—this book, by all means, please let people know. Amazon reviews are one popular way to share your happiness (or lack of happiness), and you can leave reviews on this book's website (*http://oreil.ly/program_mobile_web_2e*).

There's also a link to errata there, which readers can use to let us know about typos, errors, and other problems with the book. Reported errors will be visible on the page immediately, and we'll confirm them after checking them out. O'Reilly can also fix errata in future printings of the book and on Safari, making for a better reader experience pretty quickly.

We hope to keep this book updated for future mobile platforms, and will also incorporate suggestions and complaints into future editions.

Conventions Used in This Book

The following font conventions are used in this book:

Italic
: Indicates pathnames, filenames, and program names; Internet addresses, such as domain names and URLs; and new items where they are defined.

`Constant width`
: Indicates command lines and options that should be typed verbatim; names and keywords in programs, including method names, variable names, and class names; and HTML/XHTML element tags.

`Constant width bold`
: Used for emphasis in program code lines.

`Constant width italic`
: Indicates text that should be replaced with user-supplied values.



This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, if this book includes code examples, you may use the code in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Programming the Mobile Web*, Second Edition by Maximiliano Firtman (O'Reilly). Copyright 2013 Maximiliano Firtman, 978-1-449-33497-0."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online



Safari Books Online (*www.safaribooksonline.com*) is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://oreil.ly/program_mobile_web_2e*.

To comment or ask technical questions about this book, send email to *bookques tions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

I want to thank first all the members of my family, including my parents, Stella Maris and Edgardo, my brother, Sebastián, and my lovely wife, Ani, who have supported me during the writing of this book and all of my projects.

Second, thanks to the many people who have helped with comments, reviews, and criticisms on the first edition of this book and on the many training courses and conferences I've held since its publishing. Without them, I could not have learned as much as I have and gained experience in this minefield.

I want to thank my technical reviewers, who helped find some bugs and fill in some information gaps: Tomomi Imura, Carlos Solís, Dion Almaer, Luca Passani, and Edgar Parada.

A special thanks to Vivian Cromwell, Alejandro Villanueva, Nick Bortolotti, Fernando Freytes, John Koch, Robert Burdick, Adam Stanley, and Bryan Tafel for their support.

A special mention to Rachel Head, copyeditor of this book, who did a really great job making this book a perfect read even with my not-so-perfect English.

Finally, to Simon St. Laurent at O'Reilly Media, thanks for all your help and for trusting me when I presented this risky project in 2009, and for your help in continuing its evolution.

Creating a book about the mobile web was really a challenge; updating it after more than three years was hard, but it was worth it. Enjoy!

CHAPTER 1

The Mobile Jungle

Everyone wants to go mobile today. There are several reasons for this, such as more than 70% of the world's population having an active mobile device, and these being really the first personal devices (*really* personal), reading the user's context all the time, always and everywhere.

So what's the problem? Well, as I always say, "Mobile is a minefield." That is, everyone trying to enter into the mobile world feels as if they're in a minefield. Every step might be the right one, or might just be a waste of time.

Should we create native apps? Should we go with jQuery Mobile? Should we create a special version for tablets? Should we target feature phones?

We have lot of questions, and sometimes it's difficult to find answers. No step feels right enough. That's the "minefield sensation."

The mobile web appears to be a solution to some of the problems that the mobile space has. This book is about trying to get an idea of what we can do and how, and what the main problems we face are.

The Mobile Ecosystem

If you are coming from the desktop web world, you are probably not aware of the complete mobile ecosystem. Maybe you have read a lot about mobile development, but I can assure you that it will be more complex than you think. Let's review the current state of affairs, so we can be sure we have all the knowledge we need to create the best solutions.

What Is a Mobile Device?

It's really difficult to categorize every mobile device. Is it a smartphone? Is it a handheld? Is it a netbook? Is it a music player?

First, when is a device considered a mobile one?

For the purposes of this book, a mobile device has the following features:

- It's portable.
- It's personal.
- It's with you almost all the time.
- It's easy and fast to use.
- It has some kind of network connection.

Let's take a closer look at these features.

Portable

A mobile device has to be portable, meaning that we can carry it without any special considerations. We can take it to the gym, to the university, to work; we can carry it with us everywhere, all the time.

Personal

We've all heard it: "Don't touch my phone!" A mobile device is absolutely personal. My mobile is mine; it's not property of the family, nor is it property of my company. I choose the ringtone, the visual theme, the games and applications installed, and which calls I should accept. My wife has her own mobile device, and so do my kids. This personal feature will be very important in our projects. You can browse a desktop website from any computer—your familiar home PC, your computer at work, or even a desktop at a hotel or Internet café—and numerous people may have access to those machines. However, you will almost always browse a mobile website from the same device, and you are likely to be the only person who uses that device.



Do a test: go now and ask some friends or colleagues to allow you to view your email or your Facebook account using their mobile devices. Pay attention to their faces. They don't want to! You will log them out from their accounts, you will use their phone lines, and you will touch their personal devices. It's like a privacy violation.

Companion

Your mobile device can be with you anytime, anywhere. Even in the bathroom, you probably have your mobile phone with you. You may forget to take lots of things with you from your home in the morning, but you won't forget your wallet, your keys, and your mobile. The opportunity to be with the user all the time, everywhere, is really amazing.

Easy to use

A notebook (or even a netbook) is portable; it can be with you at any time and it has a network connection, but if you want to use it, you need to sit down and perhaps find a table. Therefore, it's not a mobile device for the purposes of this book.

A mobile device needs to be easy and quick to use. I don't want to wait for the operating system to start; I don't want to sit down. If I'm walking downtown, I want to be able to find out when the next train will be departing without having to stop.

Connected

A mobile device should be able to connect to the Internet when you need it to. This can be a little difficult sometimes, so we will differentiate between *fully connected devices* that can connect any time in a couple of seconds and *limited connected devices* that usually can connect to the network but sometimes cannot.

A classic iPod (non-touch) doesn't have a network connection, so it's out of our list, too, like the notebooks.



Where do tablets, like the iPad, fit in? They are not so personal (will you have one tablet per member of the family?), and they may not be so portable. But they generally use mobile instead of desktop operating systems, as well as mobile browsers, so they are more mobile than notebooks or netbooks. I don't have the final answer, but they are considered mobile devices in this book.

Mobile Device Categories

When thinking about mobile devices, we need to take the "phone" concept out of our minds. We are not talking about simply a phone for making calls—that's just one possible feature of a mobile device.

With this in mind, we can try to categorize the mobile devices on the market today.

Mobile phones

OK, we still have basic mobile phones in some markets. These are phones with call and SMS support. They don't have web browsers or Internet connectivity, and they don't have any installation possibilities. These phones don't really interest us—we can't do anything for them—and in a couple of years, because of device recycling, such phones will probably not be on the market anymore.

The Nokia 1100 (see Figure 1-1) is currently the most widely distributed device in the world, with over 250 million sold since its launch in 2003. In terms of features, it offers nothing but a built-in flashlight. The problem for us, as developers, is that we can't create

web content for it. Some companies may continue to make very low-end entry-level devices like this in the future, but hopefully Nokia and most other vendors will stop creating this kind of device soon. Even newer, cheaper mobile devices now have built-in browser support. This is because the mobile ecosystem (vendors, carriers, integrators, and developers) wants to offer services to users, and a browser is the entry point.



Figure 1-1. 250 million devices worldwide sounds very attractive, but this device (Nokia 1100) is out of our scope because it doesn't have a web browser.

For example, Nokia offers Nokia Mail, an email service for non-Internet users in emerging markets. Thanks to this service, many, many people who have never before had access to email have been able to gain that access, with a mobile device costing less than $40. This widespread solution meets a real need for many people in emerging markets, like some countries in Africa and Latin America.

Feature phones

Feature phones, also known as low-end mobile devices, have a great advantage: they have web support. They typically have only a very basic browser, but this is the gross market. People who buy these kinds of devices don't tend to be heavy Internet users, although this may change quickly with the advent of social networks and Web 2.0 services. If your friends can post pictures from their mobile devices, you'll probably want to do the same, so you may upgrade your phone when you can.

Nokia, Motorola, Kyocera, LG, Samsung, and Sony Mobile have devices for this market. They do not have touch support, have limited memory, and include only a very basic

camera and a basic music player. We can find phones in this category on sale from $40 all over the world.

Just to give you an idea of how big this market is, during 2011 70% of total phone sales worldwide were in the feature phone category (source: ZDNet (*http://zd.net/Y0vG4I*)). However, these users are usually phone and messaging users only, so they rarely consume web content.

Social phones

Social phones are also known as mid-range and/or high-end mobile devices. The difference is that they are prepared for social users: apps such as Facebook, Twitter, and WhatsApp are always preinstalled and a minimum data plan is also usually included with the contract.

This is the mass-market option for a good mobile web experience. Mid-range devices maintain the balance between a good user experience and moderate cost. From $150, we can find a lot of devices in this market sector. In this category, devices typically offer a medium-sized screen, basic HTML browser support, sometimes 3G connection support, a decent camera, a music player, games, and application support.

One of the key features of mid-range devices is the operating system (OS). They don't have a well-known OS; they have a proprietary one without any portability across vendors. Native applications generally aren't available publicly, and some runtime, like Java ME, is the preferred way to develop installed applications.

The same vendors develop these devices as the low-end devices.

Originally the same category as smartphones, high-end devices are generally non-multitouch but have advanced features (like an accelerometer, a good camera, and Bluetooth) and good web support (but not the best in the market). They are better than mid-range devices but not on a par with smartphones. The enhanced user experience on smartphones is one of the key differences. The other difference is that high-end devices generally are not sold with flat Internet rates. The user can get a flat-rate plan, but he'll have to go out and find it himself.



You will find different mobile categories defined in different sources. There isn't only one de facto categorization. The one used here is based on mobile web compatibility today.

Smartphones

This is the most difficult category to define. Why aren't some social devices considered "smart" enough to be in this category? The definition of *smart* evolves every year. Even

the simplest mobile device on the market today would have been considered very smart 10 years ago.

A device in this category can cost upwards of $400. You can probably get one at half that price from a carrier; the devices are often subsidized because when you buy them you sign up for a one- or two-year contract with a flat-rate data plan (hopefully). This is great for us as users, because we don't have to care too much about the cost of bytes transferred via the Web.

A smartphone, as defined today, has a multitasking operating system, a full desktop browser, Wireless LAN (WLAN, also known as WiFi) and 3G/4G connection support, a music player, and several of the following features:

- GPS (Global Positioning System) or A-GPS (Assisted Global Positioning System)
- Digital compass
- Video-capable camera
- Bluetooth
- Touch support
- 3D video acceleration
- Accelerometer
- Gyroscope
- Magnetometer

Currently, this category includes the Apple iPhone, every Android device (including the Samsung Galaxy series and the Sony Xperia series), the Nokia Lumia 920, and later BlackBerrys.



The discussion about smartphones is always floating in the market. For example, BlackBerry believes that every device it has manufactured is a smartphone, even though they don't all have touch or WiFi connection support. On the other hand, Nokia's Asha series (in the social phone category) do have touch and WiFi support, so they may claim to be smartphones.

If you are still confused about the models, brands, and operating systems, don't worry, it will become clearer. Some confusion is normal, and I will help you to understand the mobile web ecosystem in the following pages.

Tablets

A tablet is a flat device that usually doesn't fit in a pocket and that has a touch screen with a size ranging from 7 to 11 inches. Sometimes it uses a mobile operating system (such as iOS instead of Mac OS), and sometimes it uses a touch-optimized version of a desktop operating system (such as Windows for tablets instead of Windows Phone).

Tablets have the same functionalities as smartphones, including multitasking operating systems, installed applications, and full modern web browsers. Every tablet has WiFi support, and there are some with 3G or 4G connections.

Some devices are difficult to categorize—for example, the Samsung Galaxy Note is a 4G phone with a 5.3" pointer- and finger-based touchscreen that can also be included in the tablet category.



There is an informal concept known as the *phablet*—a portmanteau of the words "phone" and "tablet"—describing phones with very large screens (5 to 7 inches), such as the Samsung Galaxy Note phone series.

Nonphone mobile devices

This may sound a bit strange. Nonphone mobile pocket devices? Indeed, there are some mobile devices that have all the features we've mentioned and have a size similar to a phone, but without voice support using the normal carrier services.

For example, Apple's iPod touch is a device in this category. They aren't phones, but they can be personal, are portable and easy to use, can be kept with you most of the time, and have WiFi connections, so they fall into the category of limited connected devices. They can have great mobile browsers, so they will be in our list of devices to be considered for development.

We can also consider some of the new ebook readers. For example, I have an Amazon Kindle ebook reader, like the one shown in Figure 1-2, with data connection support (both WiFi and 3G). The Kindle has a great web browser, if you can get used to e-ink refresh delays. Ebook readers aren't phones, but they conform to all our other guidelines for mobile devices.

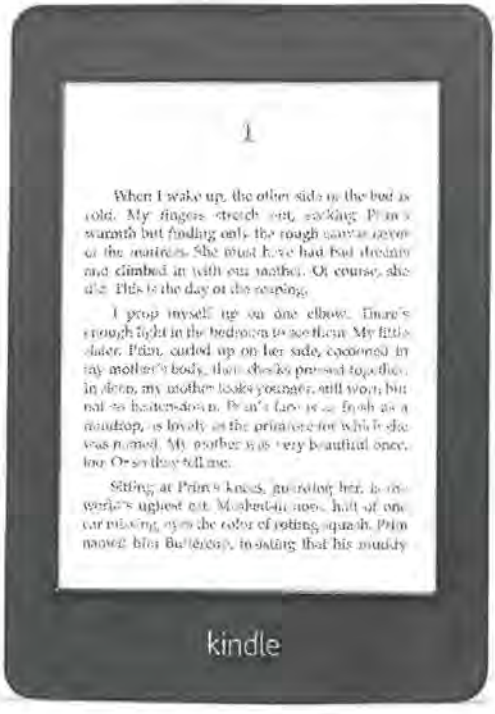


Figure 1-2. The Amazon Kindle can be considered a mobile web device because of its network connection and web browser.

At the time of this writing Google is working on a project called Google Glass, which is a connected pair of augmented reality glasses. We may be delivering services and widgets to our eyes in the near future with these new devices. We can develop solutions for them using a web service–based API.

Netbooks

Netbooks have at minimum a nine-inch display, and they are more like desktops than mobile devices. Some have desktop operating systems and desktop browsers, while others have a reduced web-based operating system (such as the Chromebooks, which include Chrome OS).

If a device has a full operating system, you will need to install antivirus protection and a firewall on it, so it won't meet the easy usage criterion for a mobile device. Also remember that you can't use a netbook while walking, and that's one of the key features

of mobile devices. Because of these limitations, netbooks are not considered mobile devices for the purposes of this book.

Brands, Models, and Platforms

Now that we have established a set of categories in the mobile world, let's talk about the difficult part: the brands and models on the market. We are not going to talk about every model available, and we don't need to know about all of them. We only need to be aware of some information that will be useful for making decisions in the future.

Writing a book about brands and models is very difficult. The market changes a lot every year. In fact, during the months while I was writing this book, I had to update this information several times. So, I'll be general here and aim to give you not only an idea of what's out there now, but also of how to understand new devices that may appear on the market.

Apple iOS

We are going to start with Apple, not because its devices are the best or because it has the greatest market share, but because Apple has caused a revolution in the market. It changed the way users see mobile devices and mobile web browsing, and it is the reason why many developers (web or not) have turned their attention to the mobile world.

Apple, a well-known desktop computer company, entered the mobile world with a revolutionary device: the *iPhone*. The iPod touch continued the iPhone revolution with a similar device without phone capabilities. Luckily for us, all of Apple's small mobile devices are quite similar. They have a multitouch screen, a 3.5" screen with medium or ultra-high resolution or a 4" screen with ultra-high resolution, WiFi connections, and Safari on iOS (formerly Mobile Safari) as the browser. The iPad series added a 9.7" screen size (medium and high resolution), with almost the same other features as the iPod touch and iPhone.



As of June 2012, Apple had sold over 400 million iOS devices, including iPhones, iPod touches, and iPads.

Figure 1-3 shows a few examples of current Apple devices.



Figure 1-3. We can find iPod touch and iPhone devices with a 3.5" or 4" screen; here you can see the iPhone 4S and iPhone 5.

Apple's devices have a great feature: an operating system called *iOS* (formerly iPhone OS) that is based on Mac OS X (a Unix-based desktop OS). Usually, the last hardware versions of the iPhone can be upgraded to the latest operating system version. By default, the iPhone, iPad, and iPod touch are charged using a USB adapter; when you charge your device, iTunes (the Apple software for managing your device's content and music) will detect automatically if an OS update is available for your device, and you can install the update in minutes without any technical skill needed. Starting with iOS 5.0, the device can upgrade itself over the air (OTA), just using your wireless connection.

That is why today, for a mobile web developer, it's more important to know what OS version an Apple device has installed than which device it is. For those of us whose aim is to create great web experiences for the iPhone, it doesn't matter if the device is an iPhone (the original phone), an iPhone 4S (the *S* stands for speed, indicating a device

with more power and speed), an iPhone 5, or an iPod touch (like the iPhone without the phone). Even within each device type, we have many generations. The same idea applies to the iPad.



The Apple iPad is a 9.7" tablet running iOS. It includes the same functionality and browser as the iPhone, with minor differences because of the larger screen, which has a 768 × 1024 resolution in the first two generations and 1536 × 2048 in the third generation. Apple sold more than 84 million iPads in just the first two and a half years after this device's release.

So, the important thing is to know which OS version a device that accesses your website is running. It may be iOS 1.0, 2.2, 3.0, 3.2, 4.0, 5.0, 6.0, or newer (although versions 1.0 to 3.2 are rarely seen on devices in use today, so we can safely work with versions 4.0 and beyond). Up to version 4.0, iOS was called iPhone OS. Every version has upgrades in the browser—for example, the Gmail version for the iPhone is different if you have an iPhone running OS 2.2 or 6.0—and is backward compatible. Table 1-1 lists the iOS versions and the devices that support them.

Table 1-1. List of iOS versions and devices supported per version

iOS version	Devices added	Highest version for	Released in
1.0 to 1.1.5	iPhone 1st gen. iPod touch 1st gen.		2Q 2007 – 3Q 2008
2.0 to 2.2.1	iPhone 3G iPod touch 2nd gen.		3Q 2008 – 1Q 2009
3.0 to 3.1.3	iPhone 3GS iPod touch 3rd gen.	iPhone 1st gen. iPod touch 1st gen.	2Q 2009
3.2 to 3.2.2	iPad 1st gen. (only)		2Q 2010
4.0 to 4.2.1	iPhone 4 iPod touch 4th gen.	iPhone 3G iPod touch 2nd gen.	2Q 2010 – 4Q 2010
4.3 to 4.3.5	iPad 2		1Q 2011 – 3Q 2011
5.0 to 5.1.1	iPhone 4S iPad 3rd gen.	iPod touch 3rd gen. iPad 1st gen.	4Q 2011 – 2Q 2012
6.0	iPhone 5 iPod touch 5th gen. iPad mini 1st gen. iPad 4th gen.		3Q 2012 – 4Q 2012

You can see some sample screens in Figure 1-4.



Figure 1-4. This is the same Gmail account accessed from an iPhone running OS 2.0 and one running 6.0—the latter provides a more rich and contextual experience for the user.

Today, we can develop applications for iOS devices on only two platforms: using mobile web techniques, and using the native *Cocoa Touch* framework built on Objective-C. There are other nonofficial platforms that can also compile iOS native applications, such as Adobe AIR, Corona, or Appcelerator Titanium.



It's not widely known today that Apple, creator of the iPod and iPhone, was really one of the pioneers in the mobile device market. The Apple Newton was on the market from 1990 to 1998, and the ROKR E1, which Apple released in conjunction with Motorola in 2005, was the first phone connected with Apple's services (including iTunes). The phone was not a great success from Apple's perspective, and that was one of the reasons why the company decided to start the iPhone project.

Every iOS device has built-in access to the App Store, Apple's famous store for selling and distributing free native apps for the iPhone, iPod touch, and iPad.



Later in this book, we are going to talk about how to detect the OS and use all the features available only in Safari on iOS. We will also talk about the App Store and how to distribute our mobile web applications via this store.

Android

Android is a platform, not a manufacturer. Therefore, it may not seem to fit in this list. It does, though—if we are developing a website for an Android device, we don't need to bother too much about who the manufacturer is. This is because the Android platform is powerful enough to leave the brand and model in second place when we are talking about developer features.

Android is an open source, Linux-based operating system created and maintained by a group of software and hardware companies and operators called the *Open Handset Alliance*. Google mainly maintains it, so it is sometimes known as the "Google Mobile Operating System." As with any open source software, any manufacturer could theoretically remove all the Google-specific stuff from the operating system before installing it on its devices. However, as of this writing no vendor has done this, which is why every Android device is very "Google friendly."



According to official Google data, as of July 2012 there were 400 million Android devices in use and 1 million new devices activating every day.

Android is a software stack including a Linux-core, multitasking operating system based on the concept of a virtual machine that executes bytecode, similar to .NET or JVM (Java Virtual Machine). Google chose Java as the main language to compile (not compatible with Java ME) with web users in mind. Android includes a bunch of Google services, such as Google Maps, Google Calendar, Gmail, and an email client, and provides connections to many free Google web services. It's not an obligation, but as of today every Android device is touch-based, and many of them have a QWERTY physical keyboard, GPS, a digital compass, and an accelerometer.



Android is the mobile platform with the most choices when talking about mobile browsers. From the default Android browser available before Android 4.1, to Firefox, Google Chrome, Opera Mobile, Opera Mini, UCWeb, and many others, there are multiple options available to download and install for free.

Today, HTC, Motorola, Samsung, LG, Acer, Asus, and Sony Mobile make Android devices. There are also some nonphone devices, such as tablets, that use Android. There are even some personal video-playing glasses that use Android, and Google TV, an interactive television platform, is based on this mobile operating system.

An Android device usually comes with the *Google Play Store* (previously known as Android Market), the official platform store for Android apps, music, books, and other multimedia content, preinstalled. Some Android-based vendors replace this or add their own stores, such as the *Amazon Appstore* or *Samsung Apps*.



Android 3.0 was the first version supporting large screens and tablets. However, you can find lots of 7" tablets—such as the first generation of Samsung Galaxy Tabs—and some 10" tablets from low-budget manufacturers using Android 2.2 and Android 2.3 sold before 2012.

Versions

As of the writing of this book, the Android OS comes in multiple versions. A device usually can update its OS only once or twice, because every manufacturer (and sometimes carriers) needs to create its own version of Android from Google's source code. That means that at any given time we can find on the market devices running at least three main versions of the OS that are stuck on that version and will not receive an update.

Knowing the OS version will be very useful to determine what browser features are available. Unfortunately, the documentation about the Android browser's features is not complete, although (as we will see in the next chapter) Google Chrome has appeared to solve some of these problems for the future.

Every Android version is known by its number and also by a codename that is always a dessert, beginning with successive letters of the alphabet. In Table 1-2 you will see a list of the Android versions that have been published or are scheduled to be published at the time of this writing. Google also maintains a website (*http://bit.ly/X3KDsh*) where you can see every version's market share over the last 14 days.

Table 1-2. List of Android versions and code names

Android version	Codename	Released on	Optimized for
1.0 & 1.1		2008, 2009	Smartphones (deprecated)
1.5	Cupcake	1Q 2009	Smartphones (deprecated)
1.6	Donut	3Q 2009	Smartphones (deprecated)
2.0 & 2.1	Eclair	4Q 2009	Smartphones
2.2	Froyo	2Q 2010	Smartphones
2.3	Gingerbread	4Q 2010	Smartphones
3.0, 3.1 & 3.2	Honeycomb	2011	Tablets
4.0	Ice Cream Sandwich (ICS)	4Q 2011	Smartphones & tablets
4.1 & 4.2	Jelly Bean (JB)	2Q 2012	Smartphones & tablets
Not confirmed	Key Lime Pie (KLP)		Smartphones & tablets

Figure 1-5 shows a few examples of Android devices.



Figure 1-5. The Galaxy from Samsung and RZR from Motorola are the most famous Android device series; here you can see the Galaxy SIII and the RZR M.

Google's Nexus devices

In 2011, Google acquired Motorola Mobility, the well-known mobile device manufacturer. This means that Google is now itself a manufacturer of mobile devices—including the well-known tablet Motorola Xoom and the Motorola Droid series—and at the time of this writing Motorola is still working as a separate company under its own name. While Google has promised that Android will continue its current open source initiative and Motorola will not have special treatment, other competitors working closely with Android, such as Samsung and HTC, are looking for an alternate operating system in case the current status changes in the future.



Rumor has it that Google plans to deliver a new phone under this new ecosystem at some point after this book is published, under the code-name XPhone. To get updates on new devices and platforms after this book's publication, check out *www.mobilexweb.com*.

Despite its link with Motorola, Google has some devices that it identifies as key devices —examples of the best hardware for every Android version. Different vendors, such as HTC, Asus, and Samsung, manufacture these devices without any customization (pure Android), and they are usually very high-end devices, mostly acquired by developers. These Google key devices are under the Nexus series name; they include the Nexus One, Nexus S, Galaxy Nexus, and the multimedia home player Nexus Q. From 2012, Google offers different sizes for a Nexus experience, such as the smartphone Nexus 4 and the tablets Nexus 7 and Nexus 10.

Fragmentation

One of the big problems of Android from a developer's perspective is its fragmentation. By fragmentation, I mean the diversity of manufacturers and devices on the market, from very low-end cheap hardware to very expensive high-end hardware, and from really small screens and resolutions to high-definition devices with large screens. Compared to iOS, where every device is basically the same with a few hardware improvements, the difference is huge.

The openness of Android makes it the perfect platform for customization. That is why you will find lots of Android devices with different user interfaces (UIs), and even with different mobile browsers. For example, Samsung adds its own UI layer to the basic Android interface on some devices, and there are also some tablets with a complete layer over the Android UI. Amazon has released tablets under the series name Kindle Fire that are Android devices with a different user interface; Barnes & Noble did a similar thing with Android and the Nook series.

Windows

Microsoft has been trying to enter the mobile space as a big player for years. Windows Mobile was its main platform, until it decided to start from scratch: Windows Phone was the result, leading to Windows 8 on the desktop and tablets later. A new mobile operating system, developed from the ground up with a unique user interface (originally called Metro), it provides a different experience than the other main platforms. Both Windows Phone and the Windows tablet and desktop versions (from version 8) include an official Windows store for applications and content.

Windows Phone

Microsoft launched its new operating system with companies such as HTC, LG, and Samsung, but it didn't get too much market share. A special agreement with Nokia changed this, providing much wider Windows Phone distribution worldwide starting in 2012. Most independent analysts conclude that in the following years iOS, Android, and Windows Phone are likely to be the main three platforms in the market.

The first version of the Windows Phone OS was 7.0 (following on from Windows Mobile 6.5). This was followed by the first big step: Windows Phone 7.5, also known as Mango, which supported multitasking and HTML5 in Internet Explorer 9. The operating system has the same restrictions as iOS when dealing with the balancing act of providing an easy-to-use platform and hiding some low-level stuff, such as the filesystem, app installation from unknown sources, or real multitasking. Windows Phone devices include Microsoft-related services and applications, such as Office, Internet Explorer, and Bing services. Applications on Windows Phone devices can only be installed through the official Windows store.

The second generation of this platform, known as Windows Phone 8, is not compatible with devices sold with 7.x, although these devices were upgraded to Windows Phone 7.8, which includes some of the new features of the second-generation platform (such as a new Home screen with tiles). Windows Phone 8 includes a new architecture and an optimized UI for better customization, including Kid's Corner, which provides a worry-free way for your kids to play with your phone. All Windows Phone 8 devices include the Windows Phone Store, formerly known as Windows Marketplace, for native app distribution. Figure 1-6 shows an example of a device running Windows Phone 8.



Figure 1-6. Windows 8 and Windows Phone 8 include a new user interface, originally called Metro; here you can see the Nokia Lumia 820 with Windows Phone 8 installed.

Windows 8

Windows Phone is not intended for tablets because Windows 8—the big version—is prepared for bigger touch devices. Windows 8 includes the same user interface that appeared first in the phone version of Microsoft's mobile operating system.



Microsoft originally named the Windows Phone and Windows 8 user interface *Metro*, but a trademark dispute forced it to drop this name. It is now called the "Windows 8–style UI" or "Windows Store UI."

Windows 8 is included with many tablets on the market, including Microsoft's own devices and those from different vendors, and every version supports Windows 8 Store apps—full-screen applications created with HTML5 or .NET. The operating system includes an official Windows Store for the first time, for app distribution.

This operating system is optimized for desktops, notebooks, and tablet devices. For tablets, it is available in two main versions: Windows 8 Pro and Windows 8 RT. The first version is the "classic" Intel chipset–based version, and it supports every Windows Vista or legacy Windows XP application in a "Desktop" mode.

Windows 8 RT is optimized for ARM-based devices and is more common on tablets than the more powerful (Pro) version of the operating system. This version has a limited desktop mode that works with Microsoft apps; it can't run any classic Windows application, just Windows Store certified apps.

Microsoft has also released its own tablets running Windows 8 under the series known as Microsoft Surface. Two main tablet categories are offered, with the first (cheaper) one shipping with Windows 8 RT and the second with Windows 8 Pro. One of the big advantages of the Surface over other tablets on the market today is the ability to use the cover as a keyboard—it's available as a touch surface or a physical keyboard version.

Windows Mobile and Embedded Compact

One of the older mobile operating systems on the market is Windows Mobile (formerly Windows CE for PocketPC and Smartphones). For many years its market included the well-known PocketPCs, personal digital assistants (PDAs) without phone features. The "mobile revolution" pushed Microsoft to create a smartphone version of its mobile operating system, called Windows Mobile, which was available in two flavors: the Professional (formerly Pocket PC) and Smartphone editions.

Today Windows Mobile doesn't have too much market share, and it has been totally replaced by Windows Phone on the consumer side. However, it is still well received in some industries for specific-purposes devices.

Almost every mobile device with Windows Mobile that has launched since 2003 has .NET Compact Framework support. This means we can develop native applications for these devices using C# or Visual Basic with a reduced .NET Framework.

For industrial and corporate users, Windows Mobile continued as a separate mobile operating system with the Windows CE codebase, called *Windows Embedded Compact*. A 7.0 version of this operating system was released in March 2011, at almost the same time as Windows Phone 7. However, it's not intended for the mass market and is only for embedded systems. In early 2013, Microsoft released Windows Embedded 8 in several versions for vertical markets: Standard, Pro, Industry, Handheld, and Automotive. Table 1-3 lists the recent and current Windows versions.

Table 1-3. List of Windows versions and code names for mobile web development

Name	Version and codename	Released on	Optimized for
Windows Phone 7	7.0 (Metro)		Smartphones (deprecated)
Windows Embedded Compact	7.0	1Q 2011	Devices for the enterprise, industrial, and consumer electronics markets
Windows Phone 7.5	7.1 (Mango)	3Q 2011	Smartphones
Windows Phone 7.5 Refresh	7.1.x (Tango)	2Q 2012	Smartphones
Windows Phone 7.8	7.x (Apollo)	4Q 2012	Smartphones
Windows Phone 8	8.0 (Apollo)	4Q 2012	Smartphones
Windows 8	8.0	4Q 2012	Tablets and desktop
Windows Embedded 8	8.0	1Q 2013 – 2Q 2013	Devices for the enterprise, industrial, consumer electronics, and automotive markets

Nokia

Nokia had the largest market share in mobile devices and smartphones worldwide for years (but not necessarily in specific markets, like the US). Nokia has devices in all the mobile categories, from very low-end devices to very high-end smartphones.

I've been one of the Nokia Developer Champions (a worldwide recognition program for top mobile developers) since 2006, and I know that Nokia really cares about the developer community. You can find the Nokia Developer website (*http://developers.nokia.com*).

The bad news for developers is that hundreds of different Nokia devices are available today. The good news is that they are very well organized by platform into different series, making it easier for us to develop, test, and port our web applications to most of them.

All Nokia devices, except the Windows-based ones, support the Nokia Store, formerly Ovi Store, for application distribution for all the supported platforms.

Windows Phone

In late 2011, Nokia began using Windows Phone as the main smartphone platform in devices released under the marketing series name *Lumia*. These devices are replacing Symbian devices as Nokia's main high-end line and will coexist with them for a couple of years.

Nokia Lumia phones have Windows Phone 7.x, 8.0, or later and include Microsoft and Nokia software such as Internet Explorer, Bing, Nokia Maps, Nokia Drive, and the Windows Phone Store.

For most of our web development work we will follow Microsoft directives, tools, and documentation, as there is no specific work to do on Nokia Windows Phone devices in

terms of mobile web development. The only exception may be the Nokia Xpress browser that Nokia offers to its users; we'll cover that later in this book.

Tablets

At the time of this writing Nokia has confirmed that it will soon be creating tablets using Windows 8 RT as the operating system.

Series 40

Nokia's Series 40 (S40) line consists of low-end and mid-range devices (both feature phones and social phones) using a proprietary Nokia OS focused on the mass market. The devices in this series first appeared in 2003, and today they are separated into different editions and even small update packages (called "feature packs") that help us to understand the abilities of each mobile device in this series. The "Lite" suffix on some editions and feature packs indicates low-end limited devices.

Between 2003 and 2007 lots of series were developed, from 1st Edition to 3rd Edition Feature Pack 3 (FP3), that today can be considered legacy platforms. There have been no new devices on the market in these subseries since the end of 2007.

In 2012 this series was renamed *Nokia Asha* for marketing purposes, and it now includes touch devices with 3G and WiFi connection support.

At the time of writing, Series 40 includes all the versions listed in Table 1-4.

Table 1-4. List of Nokia S40 versions, release dates, and device types since 2007

Version name	Release dates	Device type	Features
5th Edition and 5th Edition FP1	2Q 2007–4Q 2009	Medium feature phones	Numeric keypad, 240×320 screen
5th Edition Lite and 5th Edition FP1 Lite	2Q 2007–4Q 2009	Feature phones	Numeric keypad, 128×160 screen
6th Edition and 6th Edition FP1	2Q 2009–3Q 2011	Feature and social phones	Numeric keypad, QWERTY keyboard, and some small touchscreens; some with WiFi support
6th Edition Lite	2Q 2010–2Q 2012	Feature phones	Numeric keypad, 128×160 screen
Developer Platform 1.x	2Q 2010–1Q 2012	Social phones	Numeric keypad or QWERTY keyboard, WiFi
Developer Platform 2.0	From 2Q 2012	Social phones	Full touch devices, WiFi

Every edition has between 5 and 40 devices on the market today, and we can safely work with the fifth edition and beyond. The best part is that Nokia guarantees us that development for each device in one series (edition plus feature packs) is the same.

All the Series 40 devices have a mobile browser and *Java ME* (Micro Edition)—formerly known as J2ME (Java 2 Micro Edition)—support. From the third edition, they also

support different versions of Adobe Flash Lite. Figure 1-7 shows a recent device, the Nokia Asha 306.



Figure 1-7. The latest Series 40 devices in the social phone market, such as this Nokia Asha 306, include a full touchscreen, WiFi access, video streaming, and web browsing support.

The platform now has an application store (Nokia Store) and a modern web browser. We can also create HTML5 native web apps for these devices, as we will see later in this book.

Symbian

Series 60 (S60) began as the smartphone line from Nokia, using Symbian as the operating system. Today these devices are closer to the high-end category, but the limit is not clear. The Symbian company was formed by a group of manufacturers including Nokia, Ericsson, and Motorola. Later, Samsung and Sony Ericsson were added to the member list. For many years Nokia was the leading company using the Symbian platform, although there were also some Samsung, Sony Ericsson, and Motorola Symbian-based devices on the market.

This is history, though. In 2008, after the launching of Android as an open source operating system, Nokia made a decision: it bought 100% of Symbian, Ltd., from all the

other manufacturers and created the Symbian Foundation to migrate the Symbian operating system to open source. At the end of 2010, Nokia decided to close the Foundation and now Symbian is 100% property of Nokia. A few months later, after a lot of criticism of how Symbian was evolving to compete with Android and iOS, Nokia did a 360-degree turn, making an arrangement with Microsoft to use Windows Phone as the main smartphone platform for Nokia's future devices.

In 2012, Nokia delivered both Symbian and Windows Phone devices, and in the following years Symbian may cease to exist. However, there are millions of Symbian devices on the market and in use today, so even if its market share is decreasing, we will have Symbian for a couple of years yet. If you are in the United States this may not seem like a big deal because Symbian was never a massive platform in the US, but you can see a very different perspective in Europe, Asia, and Latin America.

Symbian devices from Nokia were divided into different editions under the name S60, from the 1st to the 5th edition. After that, the S60 platform was renamed the Symbian platform and Nokia did not use the S60 name anymore. In the market today, we can find Symbian devices under the versions listed in Table 1-5.

Table 1-5. List of Symbian smartphone platform versions available since 2008

Version name	Released on	Input type	Update ability
S60 3rd Edition FP2	1Q 2008 – 2Q 2011	Numeric keypad and QWERTY keyboard, no touch devices	No updates
S60 5th Edition	4Q 2008 – 3Q 2011	Full touch devices, no keyboard	No updates
Symbian^3	2Q 2010	Full touch devices with optional QWERTY keyboard	No updates
Symbian Anna	2Q 2011	Full touch devices with optional QWERTY keyboard	From Symbian^3
Symbian Belle	3Q 2011	Full touch devices with optional QWERTY keyboard	From Symbian^3 and Anna
Symbian Belle FP1	1Q 2012	Full touch devices with optional QWERTY keyboard	Only for some devices
Symbian Belle Refresh	3Q 2012	Full touch devices with optional QWERTY keyboard	Only for some devices



There isn't a fourth edition in Nokia's Series 40 or Series 60. Why is this? Nokia has a lot of market share in Asia, and in China 4 is considered bad luck (like the number 13 in the Western world) because it is pronounced "si," similar to "death" in Chinese.

All Symbian devices include a camera, a mobile browser, multitasking support, and a numeric or QWERTY keyboard. Fifth edition and later devices have touch support and the ability to upgrade to newer versions of the operating system. For example, the Nokia N8 originally came with Symbian^3 in 2010, and it has received free over-the-air upgrades to Symbian Anna in 2011 and to Symbian Belle in 2012.

Every Symbian device has a WebKit-based browser that allows it to browse almost any website on the Internet, including Flash-based sites like YouTube, thanks to Flash Lite.

MeeGo

In 2005, Nokia introduced a new platform to replace Symbian, called Maemo. It's a Linux-based operating system designed for small netbooks or devices with full web-browsing support. In 2010, Nokia's Maemo merged with Intel's Moblin OS, creating the MeeGo OS. While Nokia originally intended MeeGo to be the replacement platform for smartphones, it subsequently moved to Windows Phone, so the MeeGo smartphones project was cancelled inside Nokia.

Before that, one MeeGo device was released to the market: the Nokia N9, sold in Europe and Latin America at the end of 2011, was the first and last device of its kind. To be honest, I have a Nokia N9, and the Linux-based operating system looks really great. Nokia has received some criticism about discarding this promising platform.

After Nokia discarded MeeGo, Intel began looking for a new partner to continue the project. Samsung was interested, and together they created a new project called Tizen as the evolution of some parts of MeeGo in conjunction with other projects from Samsung.

At the same time, a new company appeared in Finland, formed by ex-Nokia employees with the goal of continuing production of MeeGo devices. The company is called Jolla, and its version of the MeeGo operating system is called SailFish OS (*http://sailfishos.org*).

BlackBerry

Research in Motion (RIM) was the Canadian manufacturer of the BlackBerry devices, mobile devices focused on being "always connected" with push technologies. The company was renamed "BlackBerry" in early 2013, dropping the RIM name for the future. Early adopters were primarily corporate users who needed to remain connected to intranets and corporate networks; then the devices appeared in new markets, becoming popular with teenagers and instant messaging fans. RIM used to call all its devices "smartphones," but others did not always consider them as falling into that category.



In later chapters we will cover all the tools, SDKs, and emulators available from each manufacturer to make our lives as web developers easier.

For years RIM had few devices aimed at the mass market; most of them had QWERTY keyboards and were not designed for gaming. Many of these devices had proprietary input devices, like a scroll wheel or a touchpad, although some touch-enabled devices

were launched in the last few years, giving users more multimedia and gaming support. Until 2012 all BlackBerry smartphones shipped with the BlackBerry OS, a proprietary operating system compatible with Java ME with extensions, and, of course, a mobile browser. We can categorize the devices by operating system version. The versions available on the market in 2012 were 4.7, 5.0, 6.0, 7.0, 7.1, and the new BB10 platform. Versions 4.7 and 5.0 should be considered legacy platforms; however, there are still some old devices in some markets working with these (more than three-year-old) versions.

All the BlackBerry devices support App World, the BlackBerry application store.

The PlayBook tablet

In 2011, the company released its first tablet, the BlackBerry PlayBook (Figure 1-8), a 7" device with a new operating system, Tablet OS, derived from *QNX*, a Unix-based real-time operating system. This new operating system was prepared from the ground up to compete with iOS and Android and included many new and modern features, such as multitouch support, a modern web browser, multitasking support, and even Flash Player and AIR support.



Figure 1-8. The BlackBerry PlayBook, including the Tablet OS, was the origin of the new BB10 platform, which has now also come to smartphones.

In 2012, Tablet OS 2.0 appeared on the market, and every device is upgradable to the latest BB10 operating system. A 4G tablet version was also released in 2012. The

distribution has never reached iPad or Android tablet levels, though, and the future of the platform is still uncertain.

BlackBerry 10

Starting in 2013, BlackBerry is fully changing its smartphone platform in favor of the same PlayBook QNX-based operating system, adapted to the phone layout. This new platform is called BlackBerry 10 (*BB10*), and both the phones and the tablets from the company include it. BB10 is not compatible with BlackBerry OS versions earlier than version 7.1, and all the native applications must be developed using HTML5 or C++ using the Qt framework and a UI layer called Cascades. Adobe AIR (Flash) and Android Java applications are also compatible. (No, that's not a mistake—Tablet OS 2.0 on the PlayBook and BB10 on both tablets and phones support Android Java applications through an invisible virtual machine.)

The first BlackBerry 10 devices to hit the market are the full-touch Z10 and Q10, the latter of which is a square-screen device with a QWERTY keyboard. At the time of their launch in early 2013, the new BlackBerry 10 devices are among the most powerful HTML5 devices on the market.

Samsung

Samsung has many devices on the market, most of which are divided into three different series: feature phones with a proprietary operating system, social devices with Bada, and smartphones with Android. From 2013, Samsung is also delivering devices with the new Linux-based Tizen operating system (partially based on MeeGo with Intel support).

Samsung's native devices are low-end and mid-range mobile devices with a proprietary OS including a browser and Java ME support, and typically a camera and a music player.

Bada is a mobile operating system managed completely by Android; it was released in 2010 and is optimized for mid-range devices to offer a touch interface without the need of high-end hardware. It includes a modern browser and native application installation support through the Samsung store, Samsung Apps.

In the smartphone category, Samsung is offering Android- and Windows Phone–based devices at the time of this writing, and will be offering Tizen-based devices from 2013.



Samsung Tizen-based devices support first-class-citizen HTML5 apps and Android apps through an application compatibility layer.

Prior to 2010, the smartphones and high-end devices were divided into two categories by operating system—Symbian and Windows Mobile—with each having its own set of features. Subsequently, while delivering some Windows Phone devices (primarily the Omnia series), Samsung committed its smartphone and tablet platform to Android, and it has created different series, including the Galaxy series, with devices such as the Galaxy SIII or Galaxy Tab 10.1. Some of the Android-based devices from Samsung include a UI layer called TouchWiz, providing a different experience from other vendors' Android-based devices.

Android devices from Samsung support both the Google Play Store and the Samsung Apps store for application distribution.

Sony Mobile

Ericsson built many mobile phones in the 1990s, and in 2001 it merged with Sony and created the Sony Ericsson company. Sony Ericsson produced a variety of low-end and mid-range devices and a couple of smartphones, including the *Xperia* series. In 2012, Sony became the sole owner of the company and "Ericsson" was removed from the name, converting the company into Sony Mobile.

Sony Mobile, like Samsung, has decided to offer devices with different operating systems. It offers low-end and mid-range devices using a proprietary operating system, as well as Windows Phone devices and Android devices. Before 2010, Sony Ericsson also delivered some Symbian and Windows Mobile devices to the market.

In the smartphone category, Android, Windows Phone, and Firefox OS devices are supported. The high-end series using Android is called Xperia, and it includes the Xperia Play, a PlayStation-certified phone optimized for gaming. Sony has also entered the tablet market with the Android-based devices Sony Tablet S and Sony Tablet P.

Motorola Mobililty

For many years, Motorola has been a leading manufacturer of low-end and mid-range devices. Motorola's devices were the first mobile devices on the market, and the company pioneered the clamshell design with the classic Motorola StarTac. Motorola's mobile devices have traditionally used either a proprietary operating system (like the well-known Motorola v3), Symbian, Windows Mobile, or a Linux-based operating system the company created for its devices.

This situation created a very fragmented market for developers. Fortunately, today Motorola has changed its vision and has focused on a single solution for phones and tablets: Android. Motorola has also created some inexpensive mid-range devices with Android that might be considered social devices rather than smartphones, because the screen size, the hardware capabilities, and the performance are not high-end enough for them to be compared with other smartphones.

Google acquired Motorola in 2011, but at the time of this writing it continues operations under the Motorola name and as a separate company.

Motorola is the company behind some successful Android-based series, such as the Droid series (known as Milestone outside the US) and the Xoom tablet.

Amazon

It may seem strange to see an online seller such as Amazon in this list. However, Amazon became a mobile device manufacturer with its Kindle ebook reader. The latest versions of the e-ink reader include a web browser, and while it is not one of the most-used browsers in the world, most users will use it as a backup browser.

Actually, Amazon appeared in the mobile market not with its ebook reader but with the Kindle Fire, the first tablet from the company. The first edition was a 7" tablet with a full touchscreen (not e-ink) using a customized version of Android and its own mobile browser. The Kindle Fire hit the market in November 2011 with a new price point for tablets: $199, less than half the cost of the iPad—or even other 7" Android-based tablets —at the time.

While it's difficult to know how many Kindle Fires are out there, Amazon reported millions of sales in the first weeks—more than the sales reported by other non-iPad tablets.

In September 2012, Amazon released the second generation of Kindle Fire tablets, known as the Kindle Fire HD, including WiFi connection support and a 4G-enabled 8.9" screen version.



The bookseller Barnes & Noble has also released its own ereader and tablet, called Nook. The Nook Color and Nook Tablet are Android-based devices with their own UI and browser layer.

LG Mobile

LG Mobile has many feature phones, social devices, and smartphones on the market today. Most are based on a proprietary OS with Java ME, Flash, and web support. However, LG also has devices with Android and Windows Phone on the market.



LG Mobile has released one of the first auto-stereoscopic 3D devices on the market, the LG Optimus 3D: it's an Android device with a 3D screen that does not require the use of glasses. While there is an SDK that works with the Android SDK, there is no special 3D behavior that can be applied on web pages yet.

HTC

HTC has become very popular in the mobile market since it created the first and second Android devices in the world and the first Google phone, the Nexus One. But HTC doesn't create only Android devices; it also produces a lot of Windows Phone devices. We can think of HTC devices as either Android devices or Windows devices; that's the only distinction that's needed.

HP and Palm

My first mobile device was a Palm III, back in 1998. At that time, it was a great device. It was touch-enabled (used with a stylus), black and white, and very small. It was a revolution for me: I could install applications, read newspapers, and even program directly on the device with a Pascal for Palm interpreter. OK, the programming wasn't the best experience, but the concept was really powerful.

USRobotics bought Palm Computing, Inc., in 1995. At the time, it was the pioneer launching PDA devices. USRobotics later merged with 3Com, and as 3Com was dedicated to network cards and accessories, Palm Inc. was created as a subsidiary. Palm Inc. was very successful, and other manufacturers (including IBM) created other devices licensing its Palm OS. In 1998, a couple of Palm's directors left to create another company, HandSpring, which released the Treo devices to the market. Half PDA and half mobile phone, they can be considered the first smartphones on the market.

A few years later, Palm decided to divide the company into a hardware manufacturer, palmOne, and an operating system developer, PalmSource. This idea didn't work out: customers didn't accept the palmOne brand, so the company again acquired the Palm trademark and the operating system became the Garnet OS. In the meantime, Palm acquired HandSpring, so now we have Palm Treo devices.

In 2005, ACCESS (who also had other mobile technologies) acquired PalmSource and the operating system. Suddenly, the new-old Palm company made a difficult decision: it started to manufacture Treo devices with Windows Mobile, killing all hopes for the future of the Garnet OS (formerly Palm OS).

The Treo series was the only type of Palm device that survived in the mobile world, and BlackBerrys, the Nokia E Series, and other devices soon pushed Palm to the bottom of the market. In response, Palm created another operating system for mobile devices, aimed at being a web-oriented platform for iPhone-killer devices. *webOS* came to the market in 2009 with the first device, the Palm Pre. Other devices, such as the Palm Pixi, followed.

The company didn't do so well in the market, and in 2010, HP acquired Palm, promising evolution of webOS. In 2011, it even delivered new smartphones, such as the HP Pre 3, the HP Veer, and a new tablet, the HP TouchPad.

All this is history, to tell you that TouchPad tablet sales in 2011 proved disappointing; after that, HP decided to conclude its phone and tablet production and webOS became an obsolete mobile operating system. In 2012, HP decided to open source webOS; it became *Open webOS*, and at the time of this writing, there is no indication of who is going to use this operating system in the future. webOS is a powerful mobile operating system, so as an open source solution it seems reasonable to believe that some tablets, smartphones, or ebook readers will use this platform in the near future.

Firefox OS

The Mozilla Foundation made a late entry into the mobile world. A Firefox mobile version for Android and MeeGo devices launched in 2011, but as of today it doesn't have too much market share. However, the Mozilla Foundation didn't give up, and in mid-2012 it created a new web-based operating system for smartphones, originally called Boot2Gecko but rebranded as Firefox OS.

This new platform is an open source Android-kernel-based platform utilizing the Gecko engine (the same engine that powers the Firefox browser), optimized for mid-range hardware delivering HTML5 and web experiences with lots of rich APIs. The first agreement that led this software project to a real implementation was with Telefónica.

Telefónica is the company behind many carriers in the world, including Movistar in Spain and dozens of countries in Latin America and O2 in the United Kingdom and Germany. Telefónica has committed to create and deliver devices using Firefox OS, called Open Web Devices (*http://www.openwebdevice.com*).

Following Telefónica, Deutsche Telekom, Etisalat, Smart, Sprint, Telecom Italia, and Telenor have announced their adherence to the Firefox OS project.

The idea is to offer inexpensive devices that are comparable with iOS, Android, and Windows Phone devices, with a rich user interface and an HTML5-based development platform. Sony, as well as other vendors, has announced Firefox OS devices.

Ubuntu for Phones

Ubuntu for phones (*http://www.ubuntu.com/devices/phone*) is a mobile operating system presented at the beginning of 2013, with devices planned for release in 2014. It is based on the popular Ubuntu Linux distribution and was created by Canonical, the company behind Ubuntu support. The platform will support HTML5 and Qt native apps and includes a heavy gesture-based UI.

Chinese Platforms

China has several of its own mobile platforms, typically based on Android. We can mention here *Baidu Yi*, an Android-powered operating system for the Chinese market, and *OPhone*, a Linux-based operating system based on the original Android (even before Google bought it) used by the operator China Mobile.

Other Platforms

We've already covered almost 98% of the market. There are many other manufacturers, like Sanyo, Alcatel, Kyocera, and ZTE, but they don't have visible market share, and many of them produce devices based on platforms we've already discussed, like Windows Phone or Android. With the information I've shared with you in the preceding pages, I think you will be capable of understanding any new platform you can find on the market.

Smart TV Platforms

I know what you're thinking: my TV is not too mobile! Well, you're right. We are not going to talk too much about TVs in this book, but it's important to give them a mention. Why? Because interactive TV systems are using web technologies, and most of them are imported from mobile operating systems. While TVs are not mobile, they usually have web browsers and engines similar to those of mobile devices. And mobile web developers usually are the first group of professionals to be called when a company needs a TV-based application.

At the time of this writing, interactive TV platforms are divided into native platforms and set-top box (STB) platforms. The first group includes companies that are delivering the interactive platform with the TV itself, while the latter are just set-top boxes that you can buy and attach to any HDMI-based TV.

As with smartphones and tablets, we can find low-end TVs and high-end TVs on the market, with radical differences in the engines behind them.

Interactive TV platforms typically include a web browser and native web or widget apps that can be attached to the TV home screen or even to a live channel to follow the video streaming. On the market today we can find Android-based, Opera-based, and custom-based TV platforms.



Some console platforms, such as the PlayStation 3 or Wii, include a web browser, and they should be considered if you are creating a website for TV.

If you want more information about smart TV web platforms, you can look at the following websites for developers:

- Google TV (*http://developers.google.com/tv*)
- Samsung Smart TV (*http://www.samsungdforum.com*)
- Philips NetTV (*http://www.supportforum.philips.com*)
- LG NetCast (*http://netcastdev.lge.com*)
- Panasonic Viera Connect (*http://developer.vieraconnect.com*)
- Opera TV platform (*http://www.opera.com/business/tv/emulator*)

While Apple has the Apple TV platform based on iOS, at the time of this writing there is no web browser and no application installation process on this platform.

Technical Information

After reading the previous section, you may be wondering where you can find information about all the individual devices on the market. What operating system does the Nokia Lumia 900 use? Does the BlackBerry PlayBook use the same browser as BlackBerry's smartphones? Which HTC devices use Android 4.0?

To get you closer to these answers, Table 1-6 lists the developer sites of all the major device manufacturers and platforms. Everyone has one, and almost all of them list the technical specifications of each of their devices. You can usually filter the devices by any characteristic, such as screen size, platform, operating system, or browser version. Sony Mobile's developer site is shown in Figure 1-9.

Table 1-6. Mobile manufacturer and platform developer website URLs

Manufacturer/platform	Developer site URL
Apple iOS	*developer.apple.com/ios*
Android	*developer.android.com*
Nokia (S40, Symbian, MeeGo, Windows Phone)	*developer.nokia.com*
RIM/BlackBerry	*developer.blackberry.com*
Sony Mobile	*developer.sonymobile.com*
Microsoft Windows Phone	*dev.windowsphone.com*
Motorola Mobility	*developer.motorola.com*
Opera Mobile/Mini	*dev.opera.com*
LG	*developer.lgmobile.com*
Samsung	*developer.samsung.com*
Samsung Bada	*developer.bada.com*
Amazon Kindle	*developer.amazon.com*

Manufacturer/platform	Developer site URL
HTC	*htcdev.com*
HP webOS	*developer.palm.com*
Open webOS	*openwebosproject.org*
Barnes & Noble Nook	*nookdeveloper.barnesandnoble.com*
Tizen	*developer.tizen.org*
Firefox OS	*developer.mozilla.org/apps*
Ubuntu for Phones	*developer.ubuntu.com/gomobile*



Figure 1-9. The Sony Mobile Phone Gallery—almost every manufacturer website for developers allows you to filter the devices by features, such as the browser used.



If you are new to the mobile development ecosystem, it's a good idea to register on all the developers' websites—and even the operators', if they have one. You will receive updates about tools, documentation, and news. You will also have access to download tools and emulators.

CHAPTER 2

Understanding the Mobile Web

Isn't the mobile web the same web as the desktop one? It does use the same basic architecture and many of the same technologies, though mobile device screens are smaller and bandwidth and processing resources are more constrained. There's a lot more to it than that, though, with twists and turns that can trip up even the most experienced desktop web developer.

Myths of the Mobile Web

As the Web has moved onto mobile devices, developers have told themselves a lot of stories about what this means for their work. While some of those stories are true, others are misleading, confusing, or even dangerous.

It's Not the Mobile Web; It's Just the Web!

I've heard this said many times in the last few years, and it's true. It's really the same Web. Think about your life. You don't have another email account just for your mobile. (OK, I know some guys that do, but I believe that's not typical!)

You read about the last NBA game on your favorite site, like ESPN; you don't have a desktop news source and a different mobile news source. You really don't want another social network for your mobile; you want to use the same Facebook or Twitter account as the one you use on your desktop PC. It was painful enough creating your friends list on your desktop, you've already ignored too many people...you don't want to have to do all that work again on your mobile.

For all of these purposes, the mobile web uses the same network protocols as the whole Internet: HTTP, HTTPS, POP3, Wireless LAN, and even TCP/IP. OK, you can say that GSM, CDMA, and UMTS are not protocols used in the desktop web environment, but

they are communication protocols operating at lower layers. From our point of view, from a web application approach, we are using the same protocols.

So, yes...it's the same Web. However, when developing for the mobile web we are targeting very, very different devices. The most obvious difference is the screen size, and yes, that will be our first problem. But there are many other not-so-obvious differences. That's why the concept of the mobile web exists as a group of best practices, techniques, and frameworks specially designed for mobile devices. I'm pretty sure that in the near future all these techniques and best practices are going to become must-knows for every web developer, and perhaps the mobile web will disappear; it will just be the Web for different devices.

Don't get me wrong—this doesn't mean that, as developers, we need to create two, three, or dozens of versions duplicating our work. In this book, we are going to analyze all the techniques available for this new world. Our objective will be to make only one product, and we'll analyze the best way to do it.

You Don't Need to Do Anything Special About Your Desktop Website

Almost every smartphone on the market today—for example, the iPhone and Android-based devices—can read and display full desktop websites. Yes, this is true. Users want the same experience on the mobile web as they have on their desktops. Yes, this is also true. Some statistics even indicate that users tend to choose desktop web versions over mobile versions when using smartphones.

However, is this because we really love zooming in and out, scrolling and crawling for the information we want? Or is it because the mobile versions are really awful and don't offer a great user experience? I've seen a lot of mobile sites consisting of nothing but a logo and a couple of text links. My smartphone wants more! In my opinion, for most cases, it is a much better idea to create a device-specific user interface. We are not trying to create fragmentation; we are trying to give the user the best experience her device can support.

One Website Should Work for All Devices (Desktop, Mobile, TV)

As we will see, there are techniques that allow us to create only one file but still provide different experiences on a variety of devices, including desktops, mobiles, TVs, and game consoles. This vision is called "One Web," and there is also a technique called Responsive Web Design that aims to achieve it. This is, to an extent, possible today, but the vision won't fully be realized for years to come. Today, there are a lot of mobile devices with very low connection speeds and limited resources—nonsmartphones—that, in theory, can read and parse any file, but will not provide the best user experience and will have compatibility and performance problems if we deliver the same document as for desktops. Performance is so important for mobile devices that in most situations, if you consider all the pros and cons, the idea of providing the same document to every

device seems like just a utopian ideal. Therefore, One Web remains a goal for the future. A little additional work is still required to provide the right user experience for each mobile device, but there are techniques that can be applied to reduce the work required and avoid code and data duplication.



If you believe that the mobile web is about WML (Wireless Markup Language), a language deprecated several years ago, then you need to fast forward to the present. Today, it's almost an impossible mission to find a live phone supporting only WML as markup.

Just Create an HTML File with a Width of 320 Pixels, and You Have a Mobile Website

This is the other fast-food way to think about the mobile web. Today, there are more than 3,000 mobile devices on the market, with almost 30 different browsers (actually, more than 300 different browsers if we separate them by version number). Creating one HTML file as your mobile website will be a very unsuccessful project. In addition, doing so contributes to the belief that mobile web browsing is painful.

Native Mobile Applications Will Kill the Mobile Web

Every solution has advantages and disadvantages. The mobile web has much to offer native applications, as this book will demonstrate later. The mobile web (and the new concept of the native web app or hybrid application) offers us a great multidevice application platform, including local applications that don't require an always-connected web with URLs and browsers.

People Are Not Using Their Mobile Browsers

How many Internet connections are there in the world?

> 2,267,233,742 (33% of the world's population) at the beginning of 2012 (*http://www.internetworldstats.com*).

How many people have mobile devices?

> 5,981,000,000 (86% of the population) at the end of 2011 (U.N. Telecommunications Agency (*http://www.itu.int*))

Opera Mini is a mobile browser for all ranges of devices. It is free, and as of January 2012 it had 159 million active users. This tells us that 159 million users wanted to have a better mobile web experience, so they went out and got Opera Mini. Do all the nearly 6 billion worldwide mobile device users know about Opera Mini? Perhaps not, so it's

difficult to know how many would be interested in trying this different mobile web experience. However, 159 million unique users for a browser that the user had to install actively is a big number for me. When Opera Mini appeared in Apple's App Store, from which users can download and install applications for the iPhone, iPod, and iPad, 1 million users downloaded the browser on the first day. That's quite impressive.

The percentage of total web browsing from mobile devices is increasing month by month. Mobile browsing may become as popular as desktop browsing in the coming months and years. According to StatCounter (*http://gs.statcounter.com*), in mid-2012, 10% of total web browsing was done from mobile web devices—5 times more than at the same time one year before and 10 times more than two years before.

What Is the Mobile Web?

The mobile web, from a user's perspective, is basically just web content accessed from a mobile device. We can even discuss whether the mobile web exists as a concept from a user's perspective.

From a developer's perspective, however, the mobile web definitely does exist, and it's a group of best practices, design patterns, and even new code that we need to learn. And that's why this book exists.

Differences

If you are an experienced web developer or web designer, you may feel confident about creating mobile web experiences. It seems like the same thing in a smaller package, right? Wrong. The mobile web is really different, and even if you're a senior, experienced web professional you still have lot to learn.

Some of the main differences from the desktop web include:

- Slower networks with higher latency
- Slower hardware and less available memory
- Different browsing experience
- Different user contexts
- Different browser behavior (for example, do you know that usually only the current tab on a mobile browser is active and running effectively?)
- Too many mobile web browsers, with different versions on the market at the same time
- Some browsers are too limited, some browsers are too innovative
- Several browsers without identity (it's just...the browser), documentation, or developer tools

- Differences in testing and debugging (in the mobile web this is a challenge, and it's different from in classic web development)

Mobile Web Eras

Don't panic; this isn't a history class. However, it is useful to be aware of the history of the mobile web. This is recent history: the first mobile web platform was developed less than 15 years ago. Analyzing this history can help us to understand the technologies behind the mobile web, and compatibility issues.

WAP 1

The mobile Internet appeared at the end of the last millennium. I remember all the advertising on the streets and in TV commercials. A wide range of operators started to offer mobile web browsing, with one or two devices with Wireless Application Protocol (WAP) browsers.

What Is WAP?

The Wireless Application Protocol is a standard for application-layer network communication in the mobile world. With the exception of the i-mode protocol used in Japan and briefly in other countries, WAP is the primary protocol used by operators worldwide.

The WAP standard describes a protocol suite that allows the transportation of information between a device and the Internet (via a WAP gateway), and a list of standard recommendations for the content to be transmitted. It was created by the WAP Forum (converted in 2002 to the Open Mobile Alliance, or OMA).

For many years the term "WAP" was used incorrectly, to refer to a document type ("a WAP file") or a website as a whole ("I've developed a WAP").

WAP has two main versions: 1.1, released in 1998, and 2.0, released in 2002 (this is the actual standard). Many users are not even aware of the existence of the newer version, and today it has been replaced by classic HTTP web browsing.

I can remember traveling by train in my home city in 2000, using the first device with a WAP browser (my Nokia 7110, famous because of its similarity to Neo's phone in the film *The Matrix*). My operator had an excellent promotion: free browsing for two months. I was browsing a website with an ICQ client, and nobody in my buddy list could believe that I was chatting from my mobile phone. The free browsing promotion wasn't very popular, because at the time few people understood what the mobile Internet was. When I received a $300 bill for "voice calls," I realized that I was probably the only one

using that promotion (of course, I didn't pay the bill, because it was supposed to be free!).



Some WAP 1.x browsers were so simple that they didn't even have a "back" feature—the developer was responsible for providing a link back to the previous page.

At that time, mobile devices connected to the Internet using a voice call as a modem communication. So, every minute you were connected was charged as a voice call minute. The devices with browsers had black and white screens without image support (or very basic support) and could display only three or four lines of text on the screen. This early version of the mobile Internet was a failure. It was expensive and did not offer any useful services. The overall user experience was very poor.

A few years later, 2.5G technologies such as the General Packet Radio Service (GPRS) appeared on the market. These technologies allowed us to browse the Internet (even WAP 1 sites) and be charged according to the number of kilobytes transferred, no matter how many minutes we were connected to the Internet.

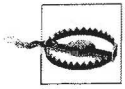
This first mobile web was defined by the WAP 1.0 standard (which, in practice, never existed on the market, having quickly been replaced by WAP 1.1). That standard suggested Wireless Markup Language (WML)—an XML version designed for mobile devices that was not compatible with the HTML standards—as the document type for the web content. The devices communicated with the operators' WAP gateways using WAP protocols, and the gateways translated the communications to HTTP and passed them on to the destination web servers. WAP 1.x is not recommended today, as it has been replaced by new technologies.



The HTML 3.2 subset in the Japanese i-mode service was one of the first languages that shifted the mobile web away from the old Handheld Device Markup Language (HDML) and WML and pushed it toward HTML and, eventually, XHTML.

Mobile browsers in this era were called "WAP browsers," and websites using this standard were called "WAP sites"—"WAP" was used instead of "web." This created a perceived distinction between the two (WAP appeared to be different from the Web).

At this time, the de facto standard for publishing WAP sites on the Internet was the use of the *wap* subdomain. So, for example, we could access Yahoo!'s WAP site using the URL *http://wap.yahoo.com*. Even today, it is not uncommon to find this domain pattern in use for mobile websites.



Most low-end and mid-range devices on the market today still support WAP content, but the browsers on newer smartphones (like the iPhone, and Android and Windows Phone devices) don't support WML content anymore.

WAP 2.0

The last OMA standard, WAP 2.0, was released in 2002. The first WAP 2.0 devices appeared in 2002, and almost every device on the market today is WAP 2.0 compatible (with some exceptions in the last few years). This standard is nearer to the web standards than the previous version and allows HTTP communication between the device and the server. The WAP gateway acts only as a proxy in the operator network.

WAP 2.0 deprecated WML and created XHTML MP (Mobile Profile), along with other companion standards that we will analyze in detail in Chapter 6. Surprisingly, after this new standard was released the word "WAP" dropped out of usage and "mobile web" started to be used. So, if we talk about a "WAP site" today, it will be understood that we are referring to a WAP 1.1 website.

Many sites continued to use the *wap* subdomain for mobile websites, while others started using the other de facto standard for publishing mobile websites, the *m* subdomain ("m" for mobile). For example, today we can access the Google Mobile website using *http://m.google.com*, or the popular Facebook social networking site using *http://m.facebook.com*.

WAP Push

WAP Push is a standard available since WAP 1.2 that allows content to be pushed to a mobile device at any time. A WAP Push is generally an SMS (Short Message Service) message to a special port with a URL to content or a website. When a device receives a WAP Push, it asks the user if he wants to go to that URL. Content portals use this method to push games, ringtones, and other premium content when you ask for that content using SMS. There are also some silent pushes from the operator that the user doesn't receive any feedback about.

WAP Link is a similar solution, but it sends an SMS to the user's inbox. The message contains a URL. Modern devices autodetect URLs inside the text messages and convert them into links that the user can click on.

Modern smartphone platforms support Push messages using an operating system layer that is out of the scope of carriers, working through TCP protocols.

The dotMobi era

dotMobi (*.mobi*) is a top-level domain (TLD) approved by the Internet Corporation for Assigned Names and Numbers (ICANN) in 2005 and made available to the public at the end of 2006. It was approved to be used as the main domain for the mobile Internet and it had the support of many software companies, Internet companies, manufacturers, and operators. The idea was that if you had the domain *yourcompany.com*, you should use *yourcompany.mobi* for your mobile website. The concept was original and some big companies have gotten on board (for example, Nokia uses *nokia.mobi*), but it's not as well supported as everyone wanted. For example, there is no browser with a shortcut to *.mobi* when you are typing in a URL address (there are shortcuts for *.com*, *.net*, and *.org*).

The TLD is managed by mTLD, an organization with many services and web portals centered around the mobile web. Any *.mobi* site should be WAP 2.0 compatible, but there is no technical obligation for this. You can buy a dotMobi domain from any registrar and host it anywhere, without any limitation.

The main criticism of dotMobi is that it created two Internets, a desktop web (*.com*, *.net*, and so on) and a mobile web (*.mobi*), rather than promoting a single Web that can be browsed from different devices.

The iPhone era

In 2007, the iPhone appeared, and it was a big hit in the market because it was the first device with a browser encouraging users to browse the "desktop/full web" instead of the mobile web. While often this is not a good idea, as we are going to see later, Safari on the iPhone completely changed the mobile web space. Almost every mobile browser today has some behavior and UI elements inspired by the iPhone browser.

Safari supported HTML5 stuff even before it had that name; they were called "Apple extensions" and led developers to create iPhone-specific websites.

The iPhone gave us another way to use the Web: applications. Lots of users use the Web through apps, such as Twitter, Facebook, or Gmail. This is not true of all users, but installed apps have become important in the mobile world if we want to be discovered and used frequently.

The mobile HTML5 era

When all the major browser vendors entered the HTML5 world, a new era started. While it is still well received to have an *m.** subdomain, mobile HTML5 is starting to promote the idea of not having a different URL for the mobile and desktop web. The same URL can have different views, served client-side or server-side with different techniques that we are going to see later in this book.

The recommended way to implement mobile websites is using the same URL for every device. However, using a mobile subdomain is still valid.

With mobile HTML5, new abilities appeared, such as the creation of installed apps and native web apps.



Back in 2005, installed apps for using web services used to be called On Device Portals (ODP), and around 2007 native web apps used to be called *widgets*. These names are legacy now and you won't find too many usages in the mobile web space today.

The Mobile Browsing Experience

The mobile browsing experience varies among different devices, and even among different browsers running on the same device. The user interfaces work very differently.

Navigation Methods

A mobile website can be navigated using different techniques. Every mobile browser uses one or many of these modes of navigation. The modes are:

- Focus navigation
- Cursor navigation
- Touch navigation
- Multitouch navigation

Focus navigation, illustrated in Figure 2-1, is the most frequent mechanism used for browsing websites on low-end and mid-range devices. (Smartphones that have hardware cursor keys, a touchpad, or a scroll wheel sometimes use focus navigation as an alternative to touch.) With this mode, a border or a background color is used to show the user where the focus is. In general it is used in nontouch devices: the user uses the cursor keypad to navigate between links and scroll the website. Pressing the down arrow key makes the browser change the focus to the next focusable object (for example, a link, a text field, or a button), or scroll a couple of lines in the content if there is no other focusable object nearby.



Figure 2-1. Focus navigation on a low-end device

Cursor navigation, illustrated in Figure 2-2, emulates a mouse cursor over the screen that can be moved using the arrow keys. A mouse click is emulated with the Fire or Enter key. For a better experience, many browsers jump the cursor to a nearby focusable object to reduce the distance the user has to move the pointer to use a link or a button.



Figure 2-2. Cursor navigation on nontouch devices shows a typical mouse pointer that allows mouseover events and mouse effects in a website

Touch navigation may seem obvious, but we need to be aware of one thing: the user may navigate using a finger or a stylus. The differences in design can be huge; precision is much lower if fingers are used. Touch devices allow the user to use detectable gestures to easily perform some actions. We will cover gesture detection in later chapters.

Some devices are also *multitouch*, allowing the users to select many objects at the same time and incrementing the number of gestures that can be detected.

Zoom Experience

Analyzing how browsers manage zoom options reveals two different types of browser. The first type offers *basic zoom* capabilities: the web page is always rendered at 1:1 scale to the original design, and the user can only change the font size. If the design doesn't fit on the screen, the scroll bar comes in to solve the problem.

The second type offers *smart zoom* capabilities: the web page can be viewed at any zoom scale the user wants, and the zooming action affects the font size, the images, and the web page as a whole. Based on a user gesture or menu option, we can switch from a full-page view to a paragraph view, as shown in Figure 2-3.

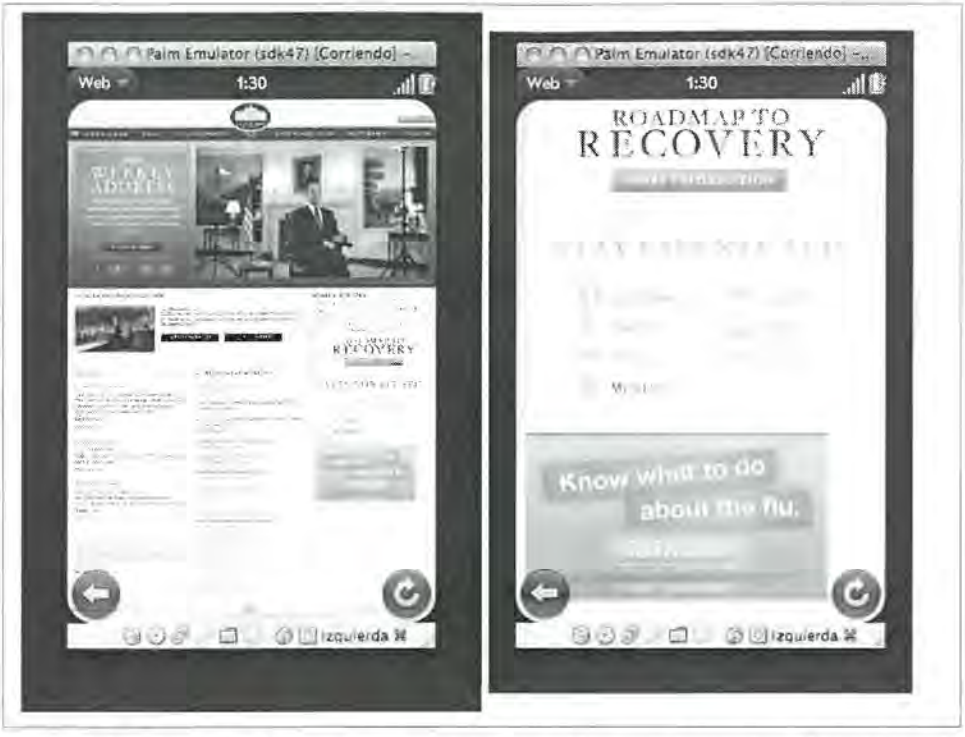


Figure 2-3. The webOS browser, like many others, offers smart zooming: the entire website layout is rendered first, and when the user double-taps on part of the page, the smart zoom focuses in on that area

Some browsers use smart zooming like on a desktop: if a paragraph extends beyond the page width, when you zoom in you need to scroll horizontally (Safari on iOS is one example). Some others reflow the text when zoomed in to fit the page width (the Android browser does this), and still others (such as Opera Mini) reflow the page even when zoomed out.

Reflow Layout Engines

Some mobile browsers aim to offer a better experience to mobile users browsing websites that were not designed for mobile devices by reflowing the pages to a one-column design. The smart zoom option has started to replace this technique, but there are still some browsers that use a reflow algorithm. For example, Figure 2-4 shows the result of using Opera Mini's "Single Column View" on a web page.



Figure 2-4. The same website browsed with Opera Mini in normal mode and in "Single Column View," with a reflow engine that autodetects navigation bars, content zones, and footers and shows us a one-column view of the site

Direct Versus Cloud-Based Browsers

Another difference we will find is between *direct* browsers, which get content directly from the website server, and *cloud* browsers (also known as proxied browsers), which go through a proxy server on the cloud. The proxy server usually does many of the following actions on the fly:

- Reduces the content, eliminating features that are not mobile-compatible
- Compresses the content (images included)
- Pre-renders the content, so it can be displayed in the browser faster
- Converts the content, so we can see Flash video on devices with no Flash support
- Encrypts the content
- Caches the content for quick access to frequently visited sites

Cloud-based browsers, such as Opera Mini and the Nokia Xpress browser, consume less bandwidth, increasing performance and reducing total navigation time at the same time. From a developer's perspective, the main difference is that the device is not directly accessing our web server; it's the proxy server on the cloud that is requesting all the files

from our server, rendering those files on the cloud, and delivering a compressed, proprietary result to the browser.

Multipage Experience

There are very different approaches to multipage browsing (such as opening more than one web page at the same time). This can be initiated by the user, or by the developer opening a pop-up window or a link in a new window. Different browsers take different approaches and may support any of the following:

- Only one page
- Multiple windows (shown in Figure 2-5)
- Window stacks
- Tab navigation



Figure 2-5. The iPhone browser has multiwindow support, allowing the user to maintain up to eight different websites open at the same time.



The iPhone, Android, and Windows Phone browsers all support multiple windows.

Web Engines

The rendering engine is the heart of the browser. It's the native code that parses, understands, and renders the page after reading the HTML and CSS or executing JavaScript code. While some rendering engines come with a default JavaScript execution engine, some browsers upgrade it to a different one.

Rendering engines

With regard to mobile devices, the king of rendering engines is WebKit.

WebKit is an open source layout engine for web browsers. Apple created it based on the open source KHTML engine (used in the Linux Konkeror browser) to be used in its Safari web browser for Mac OS X, and later for Windows and iOS. As an open source project, there are many ports of the rendering engine, and today we can find many applications using it—for example, Google Chrome and Adobe AIR.

The great thing about WebKit is that almost everyone in the mobile world is using it. This means that even on very different mobile devices we can expect very similar web rendering with simple markup and styles, which is good news for developers. However, it isn't perfect—as we'll see in later chapters, many differences do exist between WebKit implementations, as well as in HTML5 compatibility.



WebKit2 is a new project created from the ground up supporting a new internal architecture that splits the web content process from the application UI process. At the time of this writing the only mobile browser supporting this new version is the Nokia Browser for MeeGo.

Firefox on Android, Firefox OS, Windows 8, and MeeGo use Gecko, another open source rendering engine created by the Mozilla Foundation; Opera used a proprietary engine called Presto until 2013; and Microsoft uses its own engine called Trident on Internet Explorer. Other browsers for older devices or feature phones use their own engines.



As of January 2013, according to StatCounter (*http://gs.statcounter.com*), WebKit has 44% of the desktop web browsing market and 66% of the mobile web browsing market. In the mobile world, Opera and the Presto engine have the next biggest share, with 17%, and the UC browser has 10%.

JavaScript engines

JavaScript is a scripting language used by interpreters, not compilers. However, starting in 2010, some new JavaScript engines appeared in the market, increasing execution speeds by two or three times. The main difference in these new engines is a precompilation process that accelerates JavaScript code execution time, known as the JIT (Just in Time) compiler.

The first of the new engines was Carakan, created by Opera for its desktop browser; this was followed by V8, an open-source JavaScript engine developed by Google that is currently used in Google Chrome and other projects. Other modern JavaScript engines include JägerMonkey (Mozilla Foundation), IonMonkey (Mozilla Foundation, since 2013), SquirrelFish Extreme (WebKit, marketed as Nitro for Safari), and Chakra (Internet Explorer).



While WebKit is important in the mobile web world, we need to develop, test, and profile on all the other engines too. The market changes so fast that we need to be multiplatform to be future-proof. Embrace WebKit while ensuring compatibility with standards and the whole browser ecosystem.

Fragmentation

If you are a newcomer to the mobile world, we need to discuss some things affecting web browsing before we do any coding. I know you want to start coding right now, but believe me, your project will be more successful if you know a bit about the environment.

Fragmentation in this context refers to how the market is "fragmented" in pieces supporting different interfaces, hardware, operating systems, and abilities offered to developers.

Display

I know what you're thinking: "You're going to talk to me about the small screen problems." Yes, I was going to start with that. A mobile device has a very small screen compared with a desktop. While in desktop development we talk about 19-, 21-, and

25-inch screen sizes (diagonally), in mobile development we talk about 1.5, 2.5, or 4 inches. It's really a big difference. Similarly, while in desktop development we talk about 1024×768-pixel resolution, in mobile development we talk about a quarter or half of that (or in some cases, double that).

Resolution

Resolution is the primary concern in mobile design. How many pixels (width and height) are available on a given device? This was the only portability problem for many years in the area of mobile development.



Portability refers to the ability of a mobile application to be used on multiple devices with different hardware, software, and platforms.

There are no mobile device standards regarding screen resolution. One phone-sized device may have a resolution of 128×128 pixels, and another 720×1280. The third-generation iPad has a 2048×1536 screen. But if we talk about devices sold from 2009, we can separate most of them into these groups:

- Feature phones: 128×160 or 128×128 pixels
- Social phones (group #1, nontouch): 176×220, 176×208, 240×320, or 320×240 pixels
- Social phones (group #2, touch): 240×320, 320×240, or 240×400 pixels
- Touch-enabled smartphones (group #1, low resolution): 240×480, 320×480, or 360×480 pixels
- Touch-enabled smartphones (group #2, higher resolution): 480×800, 480×854, 540×960, 640×960, 640×1136, 720×1280, or 768×1280 pixels
- Tablets: 1024×768, 1024×600, 1280×800, 1920×1200, or 2048×1536 pixels



Touch-only devices typically have a higher resolution than devices with a keyboard because no space needs to be reserved for the keypad.

Most screen sizes have recognizable names that you will find in technical documentation and inside emulator settings. Table 2-1 shows the most important names you need to understand and their relation to DPI (dots per inch) for typical smartphones on the today.

Table 2-1. List of screen resolutions available on the market for smartphones and tablets, in portrait mode

Short name	Name	Resolution (w×h in portrait mode)	Aspect ratio	Sample devices	Average screen size	PPI
QQVGA	Quarter Quarter VGA	120×160, 128×160	4:3	Nokia C1-01, LG LX150	1.5" – 1.8"	111 – 133
QVGA	Quarter VGA	240×320	4:3	Nokia Asha 300, Sony W910	2.4"	166
WQVGA	Wide Quarter VGA	240×400	5:3	Samsung Wave 2	3"	155
FWQVGA	Full Wide Quarter VGA	240×432	~16:9	Samsung F490	3.2"	155
LQVGA	Landscape Quarter VGA	320×240	4:3 landscape	BlackBerry 8320 (Curve), Sony Aspen	2.5"	160
HVGA	Half VGA	320×480	5:3	Apple iPhone 3GS, BlackBerry 9550 (Storm 2)	3.2" – 3.5"	165 – 180
nHD	Ninth of High Definition	360×640	16:9	Nokia 808 PureView, Nokia N8	3.0" – 3.5"	210 – 250
WVGA	Wide VGA	480×800	5:3	Samsung Galaxy SII, Nokia Lumia 900	3.7" – 4.3"	220 – 252
FWVGA	Full Wide VGA	480×854	16:9	Nokia N9, Motorola Droid X, Sony Xperia Play	3.9" – 4.3"	228 – 251
VGA	Video Graphic Adaptor	640×480	4:3 landscape	Nokia E6	2.5"	320
DVGA	Double-Size VGA	640×960	3:2	Apple iPhone 4S	3.5"	330
WDVGA	Wide DVGA	640×1136	16:9	Apple iPhone 5	4"	330
QHD	Quarter High Definition	540×960	16:9	Motorola Droid RAZR, HTC Sensation	4.3"	256
WSVGA	Wide Super VGA	600×1024	~5:3	BlackBerry PlayBook, Samsung Galaxy Tab 7"	7"	170
HD (also known as FWXGA/ WXGA)	High Definition	720×1280	16:9	Galaxy Nexus, Sony Xperia S, Samsung Galaxy SIII	4.3" – 4.8"	320 – 342
Sq.HD	Square HD	720×720	1:1	BlackBerry Q10	~3.5"	~300
XGA	Extended Graphic Adaptor	768×1024	4:3	iPad 2, iPad mini, HP TouchPad	7.0" – 9.7"	163 – 132
WXGA #1	Wide XGA	768×1280	15:9	Samsung Galaxy Tab 10.1, Motorola Xoom, Nexus 7, BlackBerry Z10	4.2" – 10"	356 – 151

Short name	Name	Resolution (w×h in portrait mode)	Aspect ratio	Sample devices	Average screen size	PPI
WXGA #2	Wide XGA	768×1366	16:9	Microsoft Surface	10.6"	148
WXGA #3	Wide XGA	800×1280	16:10	Nexus 4	4.8"	320
Full HD	Full HD 1080p	1080×1920	16:9	Sony Xperia Z	5"	443
WUXGA	Widescreen Ultra XGA	1200×1920	16:10	Kindle Fire HD 8.9"	8.9"	254
QXGA	Quad XGA	1536×2048	4:3	iPad 3rd gen.	9.7"	256
WQXGA	Wide Quad XGA	1600×2560	16:10	Nexus 10	10"	300

There are also still a lot of devices with custom resolutions. Web technologies will simplify this problem for us, as we'll see later in this book.

3D Stereoscopic Screens

In 2011, 3D screens started to appear on the market. Examples of these devices are the LG Optimus 3D and the HTC Evo 3D.

They use stereoscopic screens (sometimes known as autostereoscopic screens) and they don't require the use of glasses for the immersive 3D experience. Instead, there is a parallax barrier over the LCD screen, allowing us to receive different information in each eye and to create 3D images in our heads.

Usually these platforms provide an SDK for native developers to create 3D immersive experiences, but unfortunately for mobile web developers, at this time there is no 3D support for web browsing or inside a web browser. There are some extensions to known formats, such as 3D MPEG-4 video formats and the JPS image format (a JPEG file with the image for both eyes in the same file), but true 3D web development is not yet possible at the time of this writing.

Physical dimensions

The resolution isn't the only thing we can talk about with regard to a mobile device's screen. One feature as important as the resolution is the physical dimensions of the screen (in inches or centimeters, diagonally or measured as width × height), or the relation between this measure and the resolution, which is known as the *PPI* (pixels per inch) or *DPI* (dots per inch). This is very important, because while our first thought may be that a screen with a resolution of 128×160 is "smaller" than a screen with a resolution of 240×320, that may be a false conclusion. Consider the iPad 2 and the third-generation iPad: they have resolutions of 1024×768 and 2048×1536, respectively, in the

same 9.7" screen. Also, you can find on the market 10" tablets with only as many pixels as a high-resolution 4.5" smartphone.

We can categorize screen sizes as follows:

- Small phone screens: from 1.5" to 3"
- Normal/medium phone screens: from 3" to 4"
- Large phone screens: from 4" to 5"
- Small tablet screens: from 5" to 8"
- Large tablet screens: from 8" to 11"

Every screen size type can have different density options:

- Low density: 100 to 130 PPI
- Medium density: 130 to 180 PPI
- High density: 180 to 270 PPI
- Ultra-high density: more than 270 PPI

That's why, on the market, we can find medium phone screens with medium density, high density, and ultra-high density with resolutions of 320×480, 480×854, and 640×960, all having the same physical screen size (meaning more or fewer pixels in the same area).



Android offers an online statistics site (*http://developer.android.com/about/dashboards/index.html*) that gives information on the market share of devices with different screen sizes and densities. At the time of this writing, 50% of Android devices are normal phone size with high-density screens, 25% are normal phone size with ultra-high-density screens, 4.6% are large tablet screens with medium-density screens, and the rest is shared between all the other combinations.

One of the phones I owned back in 2006, thanks to a gift from Nokia, was an N90. The device was like a brick, but the great (or not so great, as it turned out) feature was its resolution at the time: 352×416. The problem was that the screen size was very similar to those of other devices on the market at the time that used resolutions like 176×208. Therefore, I couldn't use any game or application on the device, or browse the Web; I needed a magnifier to see the normal font size. Every programmer thought that more

available pixels meant a bigger screen, so why bother increasing the font? "Let's use the extra space to fit more elements," everyone thought. Wrong.

Retina Display

In June 2010 Apple presented the iPhone 4, the first device with a "Retina display"—that is, a display with 326 dots per inch (DPI). The human retina can discern up to a limit of 300 DPI at a certain distance, so this device, with 960×640 pixels in landscape mode, has more pixels per inch than we can really perceive distinctly at that distance.

The third-generation iPad (known as the "new iPad") has double the previous iPad 2's resolution—264 PPI—and the same is true of some new MacBook devices with Retina displays.

The Retina display is a trademark from Apple that today means "double the resolution (or four times the available pixels)" and has no direct relationship with a certain PPI value—as we've seen, it can be 326 DPI on the iPhone and 264 DPI on an iPad.

In response, in January 2013, Sony announced the Xperia Z, the first device with 443 DPI, naming it a device with a "Reality Display."

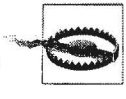
The iPhone 4S has a display size of 4.5" × 2.31" (11.52 cm × 58.6 cm) = 326 PPI (or 0.07 mm dot pitch), in comparison with other devices with a similar screen size, which have between 180 and 220 PPI.



You can find a PPI and DPI calculator (*http://pxcalc.com*) online.

Pixel density ratio

To solve the problem of having different-resolution devices in the same device category, some mobile browsers support a feature called *device pixel ratio*. This is a multiplier that, when available, is automatically applied to our web content.



Be careful to not confuse *device pixel ratio* with *pixel aspect ratio*, which describes how the width of one pixel relates to its height.

When this feature is available inside a browser, instead of working with physical dimensions we are working with something known variously as *device-independent pixels*, *density-independent pixels*, *CSS pixels*, or *virtual pixels*. That means that if we define something to be 100 pixels wide and 20 pixels high, we are not talking about real device

pixels. Instead, our measures will be multiplied by the current device pixel ratio, so it may be 150×30 pixels on a 1.5 pixel ratio device or 200×40 pixels on a 2 pixel ratio device.

The objective of the pixel ratio idea is to simplify our lives as designers and developers: we can design for one virtual, average device and the result will look fine on every screen size. Remember, more pixels doesn't necessarily mean more space to insert content.



Thanks to pixel ratio, we can design one website for the iPhone 3GS and iPhone 5, the iPad 2 and fourth-generation iPad, or every Android smartphone, without worrying about differences in resolution. However, there are some exceptions; for example, the iPad mini exposes the same pixel ratio as the iPad 2, while they have different screen sizes (7.9" versus. 9.7" diagonally). Therefore, everything on the iPad mini looks the same as on the iPad 2, but 19% smaller.

Table 2-2 lists the device pixel ratio values that are most common on mobile devices currently on the market (as of the beginning of 2013).

Table 2-2. Most common device pixel ratio values, and sample devices using these ratios

Device pixel ratio value	Resolution	Platform using this value	Sample devices
0.75	Low	Android	Samsung Galaxy Mini, Motorola Charm, Sony Xperia Mini
1	Medium	iOS, Android, BlackBerry, Symbian	iPhone 3GS, iPad 2, iPad mini 1st gen., Nokia 500, LG Optimus One, HTC Explorer, BB Torch 9810
1.3 to 1.4	Medium	Symbian, Android tablets	Nokia N8 (1.3), Nexus 7 (1.325...)
1.5	High	Android, Symbian, MeeGo	Samsung Galaxy SII, LG Optimus, Nexus One, Sony Xperia Play, Nokia E6, Nokia N9
2	Ultra high	iOS, Android	iPhone 4S, iPhone 5, iPad 3rd & 4th gen., Galaxy Nexus, Galaxy SIII
2.1 to 2.5	Ultra high	Android, BlackBerry	Galaxy Nexus, Galaxy SIII, BlackBerry Z10
3	Ultra high	Android	Sony Xperia Z



Some ultra-high-resolution devices (or browsers inside those devices) report a value of exactly 2, even when the correct pixel ratio value should be greater than 2. Other devices report inexact numbers, such as 2.2437500953674316 on the BlackBerry Z10; therefore, you should always ask about range and not exact values.

Aspect ratio

A device's *aspect ratio* is the ratio between its longer and shorter dimensions. There are vertical (or portrait) devices whose displays are taller than they are wide, there are horizontal (or landscape) devices whose displays are wider than they are tall, and there are also some square screens. To complicate our lives as designers even more, today there are also many devices with rotation capabilities. Such a device can be either 320×240 or 240×320, depending on the orientation. Our websites need to be aware of this and offer a good experience in both orientations.



BlackBerry recently announced a new device (known as the Q10) with a square screen of 720×720 pixels, creating the first modern smartphone with a 1:1 aspect ratio screen. Ten years ago it was common to find square resolutions at a much smaller size, such as 128×128.

The most recognizable aspect ratios are:

- Standard 4:3, used in classic TV and old CRT computer monitors
- Standard 3:2, used in classic 35 mm film
- Wide-screen 16:9, used in standard high-definition TV
- Wide-screen 15:9, used in LCD desktop computer monitors
- European wide-screen 5:3, used in Super 16 mm film

The other important part of aspect ratio in terms of mobile phones is the phone factor. You will encounter all of the following:

- Phones and tablets with portrait (wider than high) and landscape (higher than wide) modes
- Phones with screens wider than they are high, such as most BlackBerry smartphones before BB10
- Phones with screens higher than they are wide with no rotation mechanism, such as most feature phones

While your first thought may be that most mobile devices should be wide-screen because they are "modern," you should check that—for example, every iPad on the market right now is using the standard 4:3 aspect ratio, and the iPhone 4S uses 3:2.



Wide-screen PC monitors used to be 16:10, but since 2008 the HD TV standard has popularized the 16:9 aspect ratio. However, in the tablet world we can find both 16:10 and 16:9 wide-screen tablets, as well as 15:9 and 4:3 devices.

Input Methods

Today, there are many different input methods for mobile devices that change how a mobile web app should be developed. A given device may support only one input method, or many of them. Possibilities include:

- Numeric keypad
- Alphanumeric keypad (ABC or QWERTY)
- Virtual keypad on screen
- Touch
- Multitouch
- External keypad (wireless or not)
- Handwriting recognition
- Voice recognition

And of course any possible combination of these, like a touch device with an optional onscreen keyboard and also a full QWERTY physical keyboard (see Figure 2-6).

If you are thinking that QWERTY sounds like a *Star Trek* Klingon's word, go now to your keyboard and look at the first line of letters below the numbers. That's the reason for the name; it's a keyboard layout organized for the smoothest typing in the English language, created in 1874. This layout is preserved in many onscreen keyboards (see Figure 2-7).



Figure 2-6. The Motorola Droid has a full slider QWERTY keyboard and, when that is closed, an onscreen touch keyboard.



Figure 2-7. The iPhone and iPod touch, like many other touch devices, use an onscreen virtual keyboard when the user needs to type something on a website.

Other Features

We could talk for hours about mobile device features, but we'll focus on the ones that are most useful for us as mobile web programmers. Key features include:

Geolocation
: Many devices can detect the user's geographical location using one or many technologies, like GPS (Global Positioning System), A-GPS (Assisted GPS), WPS (WiFi Positioning System), or cell-based location tracking.

Phone calls
: Yes, mobile devices also make phone calls!

Accelerometer
: An accelerometer is a sensor that measures proper acceleration in three axes—x, y, and z—and can be used for games and orientation purposes.

Gyroscope
: A gyroscope is a sensor that measures orientation based on the principles of angular momentum. It can be used in conjunction with the accelerometer.

Magnetometer
: A magnetometer is a sensor that measures the direction of magnetic fields and can be used as a digital compass.

Application installation
: Many devices allow the user to download and install applications over the air (OTA). This means that we can serve applications to a device from our websites.

Market Statistics

At this point, you may be tempted to close this book and leave the mobile web jungle via a shortcut. However, believe me, the path through the jungle is clearer than you probably think right now.

Let's analyze some market share information. This will help us to make some decisions about how our work will be done.

Sometimes it's difficult to decide what to target. Should we develop for all devices, or only for iOS and Android devices? How can we decide how many versions to create?

The mobile world is very different from the desktop world. If we are developing for desktops, we can assume that the market share of the available browsers will be similar worldwide. In the mobile world, this is not the case. Because of commercial agreements and cultural differences, we find very diverse market shares in different regions of the world (US & Canada, Latin America, Europe, Asia, Oceania). For example, for a long time Nokia had a huge market share in Europe and Asia, but not in the United States.

It's important to define who our targets are. Worldwide users? US users? What about gender and age? Depending on the target demographic, we can define our porting strategy.



Always keep in mind that a multiplatform solution will be preferred. Excluding users because they don't have the platform you like is not a good idea—bear that in mind when you decide about which platforms to target.

Overall mobile device sales statistics aren't the only ones we need to analyze. The market shares are very different if we look only at people using their mobile devices to browse the Web. A device with a very low total worldwide market share, such as the iPhone, can prove to have a big market share if we only analyze devices with high web consumption.

Some websites providing information on mobile web market share are:

- StatCounter Global Stats (*http://gs.statcounter.com*)
- NetMarketShare (*http://www.netmarketshare.com*)
- Akamai Internet Observatory (*http://www.akamai.io*)
- Wikimedia stats (*http://stats.wikimedia.org/wikimedia/squids/SquidReportClients.htm*)
- Chitika Insights (*http://insights.chitika.com*)
- 51Degrees.mobi's Interactive Mobile Analytics (*http://51degrees.mobi/Products/MobileAnalytics.aspx*)

To demonstrate how different the mobile web market can be from country to country, Table 2-3 shows information on the same period (late December 2012/early January 2013) provided by StatCounter Global Stats. Different providers may list different numbers for their market share. You should compare every source of information, understand how it's being measured, and use the information just to give you an idea of the tendencies.

Table 2-3. Comparison chart of mobile web market share by browser (source: gs.statcounter.com, January 2013)

Top	Global market	US	Germany	Argentina	India
#1	Android – 28%	Safari on iOS – 50%	Android – 50%	Android – 53%	Opera – 29%
#2	Safari on iOS – 23%	Android – 39%	iOS – 35%	Opera – 17%	UC Browser – 26%
#3	Opera – 17%	UC Browser – 3%	Opera – 5%	Nokia – 11%	Nokia – 17%
#4	UC Browser – 10%	BlackBerry – 2%	Internet Explorer – 2%	Safari iOS – 5%	NetFront – 11%
#5	Nokia – 9%	Nokia – 1%	Chrome for Android – 2%	BlackBerry – 3%	Android – 7%

Top	Global market	US	Germany	Argentina	India
#6	NetFront – 4%	Internet Explorer – 1%	Firefox – 1%	Internet Explorer – 2%	Dolfin – 2%
#7	BlackBerry – 3%			Chrome Android – 2%	Samsung – 2%
#8	Dolfin – 1%			Firefox – 1%	Jasmine – 1%



You can find a list of updated mobile browser and device market share statistics in the statistics section of this book's blog (*http://www.mobilexweb.com/go/stats*).

CHAPTER 3
Browsers and Web Platforms

Understanding the big picture about platforms, operating systems, brands, and models is important for getting started in the mobile market, but the most important information for us will be which mobile browser or platform is used. Browsers will guide the rest of this book and most of our work as mobile web developers.

Many web developers curse desktop browsers and compatibility issues between them. Maybe you are one of them. But compared with the mobile world, in the desktop world the browser war is really simple: we have Google Chrome, Internet Explorer, Firefox, Safari, and Opera. That's about it. In the mobile world, there are more than 5,000 devices on the market. The good news (compared with this number) is that there are fewer than 25 mobile browsers in common usage—every smartphone OS has its own mobile browser, but the proprietary operating systems for the low-end and mid-range devices mostly use similar browsers. Still, the situation is far more complex than in the desktop world!

All mobile devices come with one preinstalled mobile browser, and very few of them can be upgraded or uninstalled. There are some exceptions: the browsers included with iOS, Windows Phone, Symbian, and Android (up to 4.0) are upgraded when you update the operating system firmware.

To complicate the situation, almost every device on the market allows users to add an alternative web browser, and some carriers, like Vodafone in Europe, include a copy of an alternative web browser customized for that operator, such as Opera Mini or Mobile, along with the factory-installed browser.

To add more complexity to the ecosystem, now we can also install browsers that are not complete browsers but rather different UIs for the same default browser (what we might call "pseudo-browsers"), and we can also execute mobile web code in native apps.

Web Platforms That Are Not Browsers

I can guess what you're thinking: "Platforms that aren't browsers? I thought this book was all about creating content for mobile web browsers!" Well, that's true, but it's also true that we will deal with other platform types when creating mobile HTML5 content, whether we want to or not. That is, in some situations our content will be executed not in a browser, but inside some other native app or platform that can take web development to a different level.

While the market doesn't have concrete names for all of these things yet, in this book we are going to talk about the following:

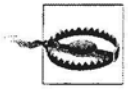
- Browser-based websites or web apps
- HTML5 apps
- Web views
- Native web apps
- Ebooks

While browser-based websites or web apps is the easiest category to understand, with mobile web technologies including HTML5 we can deliver experiences that can get users out of the browser.

HTML5 Web Apps

Some platforms allow us to create application-like experiences by using the browser as the installation platform, and providing from then on a full-screen experience based on the browser. While the name "HTML5 apps" can also be used for any web apps that can run on a browser—and even hybrids, which we'll cover shortly—in this case we are talking about browser platforms that are offering something special to a web developer to upgrade the user experience to a new level, in terms of the user interface and/or device access.

Usually this category involves installation through the browser itself or a store accessed via the browser. We don't need to compile or sign anything, and usually it involves a way to define the package of files to use and the metadata.



There may be some confusion between these HTML5 web apps and hybrid or native apps. In the latter case we are creating HTML5 apps but wrapping them inside a native app, including compilation, signing, and native store distribution. HTML5 web apps are hosted-based (must be hosted on a web server) and distributed through the browser, and there is no way to call native code outside of the HTML5 APIs available.

iOS web apps

On iOS devices we can upgrade a mobile website to a full-screen HTML5 app with a series of meta tags and techniques that we'll cover later in this book. With this technique (Apple calls it a web app), the user can install an icon in the Home screen, sharing the space with the native apps. When opened, instead of being just a shortcut to our website, the user will have a full-screen experience and no browser UI will be provided.

Mixing this technique with HTML5 APIs, we can create an offline experience like that the user gets with any installed application. Testing your app in this new environment is a must, as some behaviors are different from in the browser, and this can frustrate you if you are not aware of them (for example, the page being reloaded every time the user opens the app). Figure 3-1 shows an example of an iOS web app.



Figure 3-1. The Financial Times *offers an iOS web app through app.ft.com—as you can see, once it's installed, there is no Safari UI (including the address bar and toolbar buttons).*

Chrome apps

Google Chrome started with the idea of HTML5 apps on the desktop and notebook side, even delivering a store for app distribution, the Chrome Web Store. Today the Chrome app platform is available on Windows, Mac, Linux, and Chrome OS, and in mid-2012 Google announced that Android would be supported soon. Chrome apps are also known as *packaged apps*.

Once an app is installed in the system, it can be launched from Chrome or a system icon, and using HTML5 and Chrome APIs it can leverage a full-screen and offline experience.

At the time of this writing, Chrome apps for Android are not available, so we don't know if they will be hosted-based (as on the other platforms) or packaged as native web apps (as covered later).

Firefox open web apps

Firefox supports on all of its platforms—desktops, tablets, and smartphones—the ability to create apps and distribute them through the Mozilla Marketplace. Every application is based on HTML5, with an app manifest defining all the information needed to install the application and run it out of the scope of the browser.

We can expect these apps to work on Firefox OS, Firefox for Android, and desktop Firefox.

Symbian standalone web apps

Following iOS web apps syntax, the latest version of the Symbian browser supports the installation of Home screen icons that open full-screen web apps without the browser UI.

S40 web apps

Nokia provides a web app platform for its mid-range device series, called Series 40. This platform allows us to create cloud-based apps with web technologies that can be distributed (and sold) through the Nokia Store. Once installed, an S40 web app can be launched from the applications menu via an icon and from inside the Nokia Browser.

Samsung web API

In early 2013 Samsung released an SDK to create web apps for smartphones and smart TVs using web technologies and a JavaScript API. At the time of this writing it's compatible with devices such as the Galaxy SIII and Galaxy Note II, and it can also communicate with a Samsung smart TV. The platform supports mobile web apps, TV web apps, and convergence web apps, which interwork between a Samsung smartphone and a TV.

Web Views

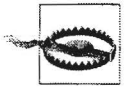
A native application is compiled and signed using official SDKs and nonweb programming languages, such as C, Objective-C, Java, or C#. Almost every native mobile platform includes a control or component that allows web content to be embedded inside a native application. This component is generically known as a *web view*, while having specific class names on every platform (such as *UIWebView* on iOS).

While sometimes the rendering engine in a web view is exactly the same as the one used in the browser, in some situations the behavior (in terms of markup, APIs, and performance) can be quite different. In later chapters, we'll see how to manage the special behaviors we need to be careful of.



Even if you are not creating a native application with a web view inside, your website may be rendered in a web view: some social media–related native apps for iOS, Android, and other mobile operating systems use this mechanism to show web content to users instead of opening the default web browser.

Examples of web views in action include most social networking applications for smartphones and tablets. When you find a post or tweet with a link to the Web, these applications open a web view for you to see this web page inside the social app, not in the browser. Usually there is a way to open the link in the web browser, but the default initial behavior is to open the URL inside a web view. In Figure 3-2 we can see this in action in the Twitter application for iPod.



From a developer's point of view a web view acts like the native default browser, but with some limitations and differences that can change expected behaviors. For example, on iOS, Safari (the native web browser) supports a JIT JavaScript engine for high-performance execution, while the web view for iOS doesn't support it for security reasons. Some HTML5 APIs can have limitations, too, as we'll see later in this book.



Figure 3-2. When a user clicks on a post including a link in a social networking app, the web page is loaded in a web view, not in the browser.

Pseudo-Browsers

I'm pretty sure you've never heard about *pseudo-browsers*. It's really a new category that I'd like to create. A pseudo-browser is a native application marketed as a web browser that, instead of providing its own rendering and execution engines, uses the native web view.

From a user's point of view, it's a browser. From a developer's point of view, it's just the web view with a particular UI. Therefore, we have the same rendering engine as in the preinstalled browser, but with a different UI. These pseudo-browsers are mostly available for iOS and Android, and they offer the same service as the native browser but with different services on top of them.

This can lead to some philosophical questions about what a browser is. From a developer's perspective, it's important to understand that pseudo-browsers are not adding fragmentation—they're not new engines, but simply the web view from the operating system.

Google Chrome for iOS (Figure 3-3) is a key example of a pseudo-browser. Because of an App Store licensing limitation, Google cannot deliver its own rendering and execution engine with a native app. Therefore, Google decided to deliver a Chrome experience application with the default iOS web view engine. This means Chrome for iOS does not have the same HTML5 compatibility or features as Chrome for Android or desktop Chrome, but it does have the same features as the iOS web view.



Figure 3-3. Chrome for iOS is an example of a pseudo-browser: the rendering and execution engines are provided by the iOS web view, not by the Chrome team, so only updates to iOS will enable more developer features for this "browser."

Native Web Apps, Packaged Apps, and Hybrids

Also known as a *hybrid* application, a *native web* application is a compiled native application that usually uses a full-screen web view as the whole application container. In one approach, the whole application is developed using web technologies (HTML, CSS, JavaScript), but it is packaged and compiled as a native application so we can distribute it in native application stores. Another approach is to use some native layers and some

web layers at the same time, such as native UI components with only some parts in a web view, with JavaScript logic.

Some web view platforms, as we'll see later in this book, allow us to communicate between JavaScript and native code, either one way or bidirectionally, opening the possibility to extend the browser's limits with native code (such as C, Java, Objective-C, or C#).



Apache Cordova is an open source native web application framework for multiple platforms that is well known in the market because of one of its implementations: PhoneGap from Adobe. We need to remember that these multiplatform frameworks are based on the web view components from each platform, so we will be dealing with different engines.

We can also use web views ourselves if we have native know-how, such as Objective-C on iOS or Java on Android. We just need to add the web view and connect the component to some local or generated HTML code.

If we don't have any native know-how or just want to make it work fast and across multiple platforms, we can use a framework that will help us with the process. Some frameworks add a layer of JavaScript and native code, offering new APIs for web developers to use. We will cover native web frameworks later in this book.

Facebook and the HTML5 Problem

For a while, the native Facebook apps for iOS and Android were hybrid apps using HTML5 for the main browsing experience. After a couple of months of using this technique and encountering some performance issues, Facebook decided to migrate some of the web view–based screens (such as the main timeline) to full native code, increasing the performance by a factor of two.

Mark Zuckerberg, CEO of the company, said in September 2012: "When I'm introspective about the last few years I think the biggest mistake that we made, as a company, is betting too much on HTML5 as opposed to native...because it just wasn't there. And it's not that HTML5 is bad. I'm actually, on long-term, really excited about it. One of the things that's interesting is we actually have more people on a daily basis using mobile web Facebook than we have using our iOS or Android apps combined. So mobile web is a big thing for us."

That said, Facebook has more mobile browser–based users than native users, and on the browser it's always HTML5. Facebook also has other HTML5-based apps, such as for Firefox OS and other platforms, and even the updated native apps are still using HTML5 in some situations.

Three months after Facebook made its changes, the Sencha (*http://sencha.com*) team—believing that the Facebook team was wrong and that HTML5 was ready—created an HTML5-redesigned Facebook app called *Fastbook* to prove the point. You can try it yourself by accessing *fb.html5isready.com* from your mobile phone. The Fastbook web app has reached the same levels of performance and UI responsiveness as the native app, using optimization techniques.

If you want to know more about Facebook's difficulties with HTML5, check out *mobilexweb.com/go/html5fb*.

Some platforms don't need any real compilation or signing process, but just need to be packaged in a ZIP or other format with a configuration file inside. The operating system will create the web view for us and load the main HTML defined in the configuration file. The main difference from HTML5 web apps, discussed earlier in this chapter, is that in a packaged app we don't use a web server to host the files.

Let's briefly review some of the most important native web frameworks on the market.

Platform-specific solutions

The following operating systems offer a way to compile and sign a package that can be distributed as a native app while being developed with HTML5:

BlackBerry WebWorks
: Available for BlackBerry 5.x, 6.x, 7.x, and 10 (including the PlayBook).

Windows 8 HTML5 apps (previously known as Metro apps)
: Available for Windows 8 (only the desktop and tablet versions, no support yet on Windows Phone up to version 8).

Tizen apps
: Available from Tizen 1.0 devices.

Symbian WRT
: Available from Series 60 3rd edition. This is a legacy platform; while it's still working, Nokia has not added any new features to the platform since 2011.

QtWebKit
: Available for Symbian and MeeGo devices from Nokia; allows developers to compile an HTML5 application as a native Qt-based package with a WebKit engine integrated inside.

Ubuntu for mobile apps
: HTML5 apps with native integration.

Apache Cordova and Adobe PhoneGap

PhoneGap was the pioneer in offering a multiplatform solution for native web development. After a couple of years working as a small open source project, Adobe acquired Nitobi (the company behind PhoneGap), donated the project to the Apache Foundation, where it was renamed Apache Cordova, and continued evolving the project with the power of Adobe.



Adobe has a free and commercial service called PhoneGap Build that is a cloud-based Cordova compiler, so we don't need to deal with the native SDK on our computers. You can find more information on this service at *build.phonegap.com*.

We'll look at Apache Cordova in more detail later in this book, but for now it's important to say that we can compile native web apps and distribute them via official stores using this open source framework.

Sencha Touch

While Sencha Touch is a UI framework, from version 2.0 it includes a native packager for iOS and Android available to Windows and Mac developers. The packager is included in the developer tools of the UI framework that can be downloaded from *sencha.com/products/touch*.

Appcelerator Titanium

This framework allows the creation of iOS, BlackBerry (7.x/10), and Android native web apps, providing a bridge that enables developers to use native UI components from JavaScript. That means that your JavaScript will be converted to native code when your app is compiled, so your app can feel more "native" than if it were using web views directly (because in most situations there is no HTML involved). You can download the free Appcelerator Titanium Studio IDE (*http://appcelerator.com*).

Ebooks

The latest ebook readers support some kind of HTML5 and web content based on different formats such as EPUB 3 or Kindle Format 8. These formats are based on HTML5, meaning that we can create interactive ebook experiences with HTML, CSS, and JavaScript.

While ebook creation is outside the scope of this book, most of the code and techniques that we are going to see can be applied easily to ebooks and digital magazines.

Mobile Browsers

When browsing the Web on our mobile devices, we can use the preinstalled browser available by default on every device or we can install new browsers through the application stores. Let's review the most important browsers in both categories.

Preinstalled Browsers

Practically every mobile device on the market today has a preinstalled browser. One of the big features of these browsers is that the average user typically doesn't install a new web browser; therefore, on each device the preinstalled browser is the most-used one. One main disadvantage of preinstalled browsers is that usually there is no way to update the browser independently from the operating system. If your device doesn't get operating system updates, usually you will not get browser updates.

Safari on iOS

Safari is a WebKit-based browser bundled with iOS that offers a great browsing experience and smart zoom options. It is updated with every operating system change to include new features that allow us, as developers, to create better user experiences.

Safari on iOS (formerly known as Mobile Safari) was the first mobile browser to support a range of new features, including those that allow us to create animations, transitions, 3D effects, and Flash-like experiences using HTML, JavaScript, and CSS (but without Flash—what we currently know as HTML5). We will cover these topics in Chapters 7, 11, and 12.



For many years, Safari on iOS was the best mobile browser in terms of HTML5 API compatibility, performance, and distribution. As of September 2012, it had more than 90% of the web browsing market share on tablets.

This browser is designed for touch and multitouch navigation. It can support focus navigation if the user is attaching an external keyboard, and it was also the first browser to support accessibility features such as a screen reader for people with visual disabilities.

Most of the HTML5 APIs were first implemented in Safari on iOS, and even most of the nonstandard code that we will see later in this book was invented in Safari and then cloned in other mobile browsers.

While the iPad/iPad Mini and iPhone/iPod touch versions are pretty similar, the tablet version additionally supports tab navigation.



Safari on iOS was the first mobile browser to support *chromeless* Home screen applications, which fall somewhere between browser-based apps and native web apps.

The official documentation for Safari on iOS can be found at *www.mobilexweb.com/go/safaridocs*. Safari on iOS doesn't follow the same versioning as desktop Safari (for Mac or Windows); usually it is named with the iOS version number, so we are talking about Safari on iOS 5.x or 6.0. This browser can't be updated using the App Store; it's only updated when the operating system is.

The latest versions of the mobile and desktop versions of Safari are starting to match up, so it's possible that in the near future Safari will be exactly the same on desktops, notebooks, tablets, and smaller devices.

Android browser

Up to and including Android 4.0 (Ice Cream Sandwich), the Android OS came with its own browser, based on WebKit. It is called the Android browser, sometimes referred to as Android WebKit. While many developers believe that it is similar to Google Chrome or even Safari on iOS, the truth is that the Android browser has always lagged behind other mobile browsers in terms of performance, HTML5 compatibility, and even bugs found.

The browser has changed a lot between different versions of the operating system, to the extent that versions 2.2 and 4.0 can be considered two distinct browsers. Even in the same version of the operating system, we can find customized versions of the browser on devices from different vendors, such as Samsung, Sony, or Barnes & Noble (with its Nook tablet).

Google realized that the Android browser was not good enough for its platform, so it started a completely separate project, Google Chrome for Android, to replace this browser.



The Android browser is considered a legacy browser after Android 4.1 (from late 2012) and has been replaced by Google Chrome. However, the web view still executes the Android browser engine.

Google Chrome for Android is now the default browser preinstalled on most new devices coming to the market with Android 4.1. However, devices that were upgraded to 4.1 from 4.0 still have the Android browser.

Google Chrome

In 2012, Chrome appeared in the mobile world as a downloadable browser for Android 4.0, and since Android 4.1 (Jelly Bean) it has begun to replace the default browser. It shares most of its code with the desktop version of Google Chrome and is one of the most modern HTML5 browsers on the market.

Unfortunately, only Android 4.0 or newer devices can install Chrome. However, one of the key features is that the browser is upgradable through the Google Play Store without the need to wait for an operating system upgrade. That means that Chrome users will have new HTML5 features available sooner than users of other platforms.



Google has released a new operating system called *Chrome OS* that is totally based on HTML5. Currently you can find this operating system on netbooks, known as *Chromebooks*, and on desktop computers, known as *Chromeboxes*. This version is not yet available for mobile devices—phones or tablets—so it's outside the scope of this book.

Chrome for Android will follow as quickly as possible updates on the desktop and Chrome OS side. The desktop version has Beta and Canary channels, where we can follow future versions of the browser closely. On Android, only the Beta channel is available. From version 26, it supports a cloud-based network compression feature.

In mid-2012, Google also released *Google Chrome for iOS*, a pseudo-browser for iPhone, iPod, and iPad users that delivers the Chrome experience—UI, omnibox URL search box, tab and info synchronization—but uses the default web view engine on iOS devices. That means that Chrome for iOS is not really Chrome. If you don't believe me, it's enough to say that Chrome for iOS does not even identify itself to servers as Chrome—it uses the same user agent name as Safari, adding a string called *CriOS* (presumably for "Chrome for iOS") so developers looking for the Chrome string inside that name will not be confused and decide to deliver Chrome-only code.



Chrome for iOS, like other pseudo-browsers, uses the iOS web view's rendering and execution engines. The same Chrome for iOS version may have different compatibility on different devices, depending on the current iOS version.

While the rendering and execution engines are from iOS and not from Chrome, this pseudo-browser adds a network layer that is different from that of Safari or the default web view, and will prefetch content and accelerate the downloading of resources.

Amazon Silk

Tablets from Amazon—the Kindle Fire family—include a new browser created by Amazon based on the Android browser, called Amazon Silk. The main difference of Silk is that it has an "accelerated mode" that proxies the rendering in the Amazon cloud servers. That means that when you are browsing in accelerated mode, part of your HTML, CSS, or JavaScript can be parsed in some Amazon servers, and a compressed result is delivered to the browser. From a user's perspective there is no real difference in the UI experience while browsing the Web.

Silk had a major upgrade in 2012 with the release of the second generation of Kindle Fire devices, including better support for HTML5 and modern techniques.

Internet Explorer

Microsoft's browser can be considered to have been one of the first mobile browsers on the market. The first version was released in 1996, for Windows CE 1.0: it was first known as Pocket Internet Explorer (PIE), then Internet Explorer Mobile, and is now just called Internet Explorer. While maintaining the same name, the truth is that the browser has changed from the ground up many times. Up to version 6.0, it had its own rendering engine (based on IE4), but today these can be considered legacy versions. With Windows Mobile 6.5, the browser was upgraded to an IE6-based rendering engine.

The new operating system from Microsoft, launched in 2010 as Windows Phone 7, came with a new version of Internet Explorer Mobile that was based on the IE7 engine, with some IE8 features mixed in (some have called it an IE 7.5 engine). It offers similar behavior to Internet Explorer 7, and multitouch support.



Older Windows Mobile devices supported an IE4 or IE6 engine.

The free upgrade to Windows Phone 7.5 came with the first HTML5-compatible browser from Microsoft: Internet Explorer 9, using the same codebase as IE9 for desktops with some special mobile additions. This was the start of a new way of Microsoft seeing the mobile web browser. Now, every desktop version is followed by a mobile version using the same rendering engine.

Windows 8 and Windows Phone 8 come with Internet Explorer 10 (IE10), which represents a big step forward in terms of mobile HTML5 API support.

Nokia Browser

Nokia Browser is a generic name for different real products available for different platforms inside the Nokia world.

Nokia Browser for Series 40. Every Nokia Series 40 device comes with a built-in web browser created by Nokia. Up to S40 5th Edition, it was a simple browser without smart zoom capabilities, designed with feature phones in mind. It was basically a focus navigation browser, based on Nokia's own rendering engine.

Beginning in the 6th Edition, the browser was updated to WebKit (similar to Nokia's Symbian browser), creating a new browsing experience for low-end and mid-range devices. The main problem with this browser was that the low-end and mid-range devices are not created with high-quality hardware, which can lead to some performance problems.

In 2010, Nokia acquired a browser company called Novarra that offers cloud-based web support. Nokia has since created a new Java-based browser called the Nokia Xpress browser (formerly Ovi Browser) that offers a new experience for feature phones and social devices. This browser was originally available as a free download from the Nokia Store and is now being preinstalled in newer Series 40 devices, such as the Asha series devices.



Nokia has also released Nokia Xpress for Lumia, a cloud-based browser for Windows Phone, based on the same Xpress browser.

The new browser renders websites on the server and delivers a compressed, already rendered version to the phone, giving users a fast browsing experience. This browser works with touch- and focus-based devices, and it adds the ability to create installed web apps for Nokia Store distribution with HTML5 support—all rendered from the Nokia servers.

Nokia Browser for Symbian. In 2005, Nokia created the first open source WebKit-based mobile browser for Symbian devices (known for many years as the S60 OSS browser). Depending on the device, it supports focus, cursor, and multitouch navigation. Many devices support more than one navigation type; for instance, the Nokia E7 supports touch (finger and stylus) navigation, and cursor navigation when the keyboard is opened.

From Symbian Anna, the browser has support for HTML5; however, it tends to lag behind other browsers even for the same operating system, such as Opera Mobile for Symbian.

Nokia Browser for MeeGo. Nokia has also created the first WebKit2 browser in the mobile market: the Nokia Browser for MeeGo. It has great HTML5 support, but unfortunately there aren't many devices on the market today supporting this operating system.

webOS browser

The HP operating system comes with a WebKit-based browser that supports the latest web technologies. It supports touch navigation and a card concept that allows the user to open many websites at the same time and flip between them using a finger.



Older Palm devices using Garnet OS (Palm OS) 3.1 and later shipped with a browser known as Blazer. Future devices will include the Open webOS operating system and an updated, open source browser named Isis (*http://isis-project.org*).

BlackBerry browser

Every BlackBerry device comes with a mobile web browser with focus navigation and, more recently, touch navigation support. Many versions of the browser are available, depending on the device. There are devices with trackball and cursor navigation, older devices with focus navigation, and newer smartphones with touch support.

The first generation of the BlackBerry browser was included with Device Software version 4.5 and earlier. The second generation, available from versions 4.6 to 5.0, had a redesigned rendering engine but was still far behind other mobile browsers in terms of performance and compatibility.

BlackBerry devices running OS 6.0, 7.0, and 7.1 have a WebKit-based browser that was completely redesigned for the tablet PlayBook and the new BlackBerry 10 platform.



The browser on the BlackBerry 10 platform was totally created using web technologies—that means the whole browser was engineered using JavaScript, CSS, and web views.

UC Browser

The UC Browser (formerly known as UCWEB) is the #1 browser in the Chinese market (and #2 in the Indian market) and is now available in English for other markets as a downloadable browser. It is a cloud-based browser supporting full HTML and JavaScript, multiple windows, and many advanced features.

It works on some 3,000 different mobile device models, including Android, Symbian, BlackBerry, Windows Phone, Java-based devices, and the iPhone.



According to StatCounter (*http://gs.statcounter.com*), in December 2012 the UC Browser had 10% usage share of the global mobile browser market and 43% usage share of the Chinese market, with some 300 million active users (85% from China).

You can download the browser from www.uc.cn/English.

Samsung Dolfin & Jasmine

The WebKit-based browser installed with the Bada OS is called *Dolfin*. This browser has included HTML5 support from Bada 2.0 and is installed on social devices.



Don't confuse Samsung's Dolfin browser with Dolphin, another option available for Android and iOS as a user-installable browser.

On low-end devices featuring a proprietary operating system, Samsung also has another browser called *Jasmine*.

LG Phantom

This is the browser included on some social devices from LG, based on WebKit. It doesn't have much market share, and there is not much information available for developers.

NetFront

NetFront is a mobile browser created by the Japanese company ACCESS, targeting low-end and mid-range devices including phones, digital TVs, printers, and game consoles. It is licensed by the manufacturer, and that's why we can find devices of many different brands using the same browser engine. NetFront is installed on thousands of Sony Ericsson, LG, Samsung, and ZTE devices, as well as on Amazon Kindle ebook readers. It is also included with the PlayStation 3, PSP, and Nintendo 3DS.

The browser has many different versions, and it uses its own rendering engine. From NetFront 3.5, it supports cursor navigation and a feature called Smart-Fit that reorganizes websites to fit into a single column without horizontal scrolling. NetFront 4.2, released in late 2011, includes some basic HTML5 support, enhanced JavaScript performance, and an optional Flash Lite Player.



While it's probable that most web developers and designers have never heard of NetFront as a browser, as of July 2012 it had 3% usage share of the global mobile browsing market, according to StatCounter (*http://gs.statcounter.com*).

ACCESS has also released two WebKit-based spin-off browsers: NetFront Life Browser for smartphones and tablets and NetFront Browser NX for other devices, such as set-top boxes. The first is available as a free download for Android devices via the Google Play Store and the Amazon Appstore for Android. As a "different" feature, the NetFront Life Browser offers a "Tilt" browsing mode that allows you to view a web page diagonally, giving you the widest possible view of the page (see Figure 3-4).

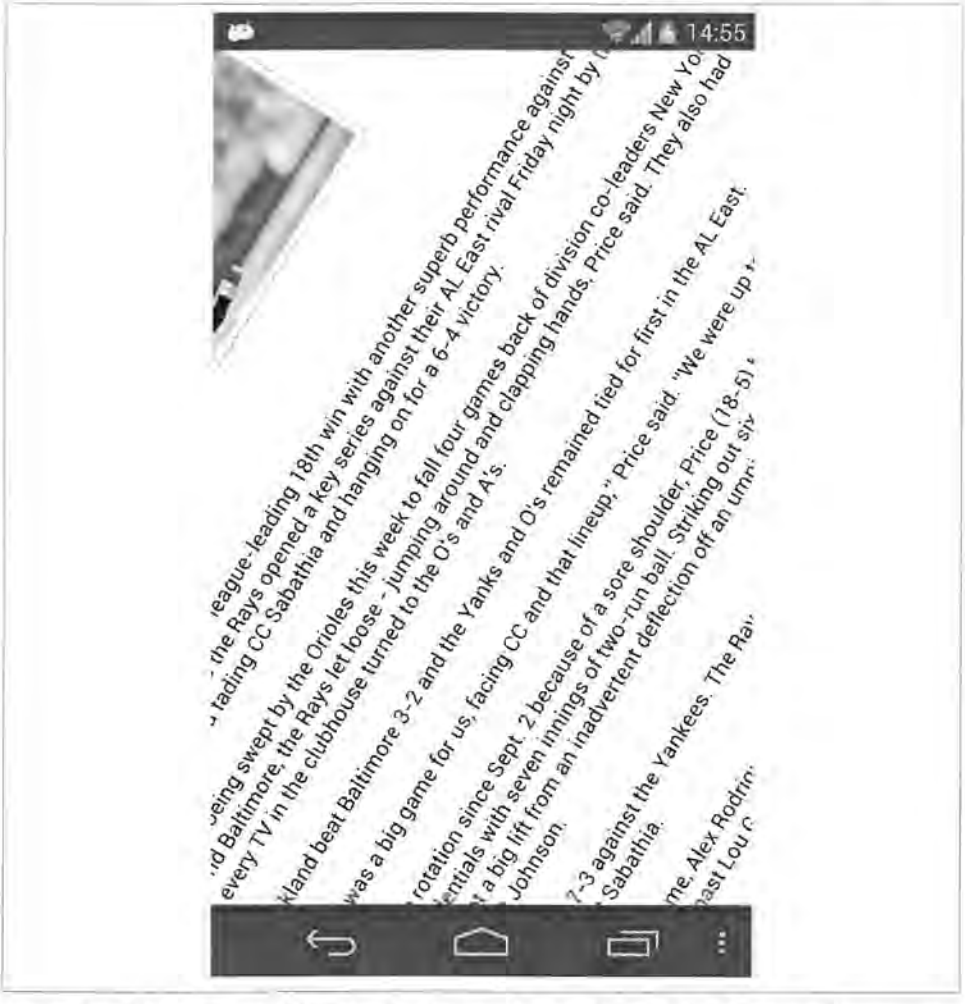


Figure 3-4. It's not a problem in the image: the NetFront Life Browser allows you to browse the Web in "Tilt mode," where you can read text diagonally, getting the widest view for a paragraph.

Myriad browser

The Openwave browser was, for many years, one of the preferred mobile browsers to be preinstalled on low-end devices. In conjunction with NetFront, it was used for the majority of low-end and mid-range browsing. Openwave was acquired by Myriad in 2008, and since that time it has been known as the Myriad browser. Like NetFront, it has been used by many vendors, including Motorola, LG, Sharp, and Kyocera.

Obigo browser

The Obigo mobile browser from Obigo/Teleca claimed 14% browser market share in 2007 (before the smartphone revolution). It was found in Samsung, LG, Motorola, and Sony Ericsson devices and in many CDMA devices from some operators, like Verizon. Obigo also offers a widget solution implemented by LG Mobile.

In this era, it's still being used in some LG and BREW devices, and the latest versions are WebKit-based.

Legacy mobile browsers

There are also a few legacy mobile browsers that are worth mentioning.

MicroB for Maemo. Maemo, the platform that merged with Intel's Moblin OS in 2010 to form MeeGo, shipped on one Nokia device: the N900. This device included a Gecko-based browser known as MicroB. Maemo and MicroB were subsequently replaced by MeeGo and the Nokia Browser.

Sony Mobile (formerly Sony Ericsson) browsers. If we analyze Sony Mobile's non-smartphone devices—that is, those not based on an operating system like Android or Windows Phone—we can find three primary browsers in use, depending on the device's release date:

- Sony Ericsson WAP browser before 2004
- Sony Ericsson web browser from 2004 to 2006
- NetFront browser (version 3.3, 3.4, or 3.5) from 2006

Other browsers, such as Opera and even Openwave, were also preinstalled on some devices.

Motorola Mobile Internet Browser (MIB). Motorola devices based on the Motorola proprietary OS (excluding the company's Linux, Windows Mobile, Android, and Symbian devices) came with a simple proprietary browser that allows focus navigation and page scrolling. The last version was 2.2. As an indication of its limits, it can render only documents up to 10 KB.

Some other older devices came with the Openwave, Obigo, or Opera browser preinstalled. The same device model shipped at different dates and in different markets did not necessarily come with the same browser.

Japanese Mobile Web

In the mobile web world, the Japanese market is quite an exception. The three main carriers in Japan (DoCoMo, Softbank, and Au) each had their own mobile web standards for years regarding markup, Emoji, geolocation, and so on, that browsers preinstalled on their devices were required to support. They didn't actually make their own browsers.

ACCESS's i-mode browser was the most common on DoCoMo devices, while Openwave was more common on Softbank and Au devices. Today, popular platforms such as iOS, Android, and other major brands are on the market, using their own browsers and operating systems.

User-Installable Browsers

The market complicates the situation further: we also need to think about *user-installable browsers*. These are free and commercial web browsers that you can install after you buy a device. Sometimes (mostly on feature phones and social devices) they are included on the device by the vendor or the operator in a particular country or region.



On iOS there is no way to replace the default browser—Apple's policies don't allow anything but the default rendering or execution engine, and the OS does not support the ability to change the default browser. Even if we install apps using the web view, such as Chrome for iOS, the operating system will always open Safari when we click on links in any app.

We need to remember that for iOS and Android there are plenty of pseudo-browsers (native apps marketed as browsers but using the default web view). Besides that, on Android we can find several browsers with their own rendering engines that can replace the default browser (see Figure 3-5).



Figure 3-5. There are a lot of browsers that can be installed on Android devices; these can be used on a per-usage basis or as the default browser.

Opera Mobile

I was an addict of Opera for the desktop for many, many years. Opera has lost the desktop browser war, but it took its experience in browser creation and entered the mobile world in 2000.

Opera Mobile is a full browser supporting tab and cursor navigation that comes factory-preinstalled on some devices and is sometimes preinstalled by the carrier using an OEM license, replacing the default device browser.

Opera Mobile is also available for download by users of Android, MeeGo, and Symbian devices. Usually, users can download the latest version and a beta of the next version, known as the Opera Mobile Labs build.

Opera Mini

My Opera addiction continues: Opera Mini remains one of the best Java ME applications ever produced for feature phones. It is a free browser that works on almost any device, including feature phones from Nokia, LG, Samsung, and Sony, and smartphones such as Android and iOS devices (iPhone, iPod, iPad). It supports "the full web" as a proxied browser. This means that if you browse using Opera Mini, you won't be accessing websites directly. Instead, the application will contact an Opera Mini server that will compress and pre-render the websites. This allows very quick full web navigation for every device, whether low-end or smartphone.

From version 4, it supports video playback, Ajax, offline reading, and smart zooming, even in low-end devices (see Figure 3-6). From version 5, it also supports tabbed browsing, a password manager, and touch navigation in devices with touch support. Versions 6 and 7 include new rendering engines, more CSS3 support, and performance improvements. Opera offers a beta version of future versions of the browser known as Opera Mini Next.



In early 2013, Opera Mobile and Opera Mini were merged into one browser on Android, known as "Opera for Android". It has the option to enable optionally cloud-based compression using Mini servers. Version 14 of this browser is the first edition using a Chromium (WebKit) engine instead of the previous Presto engine.



Figure 3-6. Opera Mini is an excellent option for low-end and mid-range devices, offering a proxied browser with smart zooming for almost any mobile phone with Java ME support. The same browser is available for smartphones.



You can download Opera Mini for free by browsing to *m.opera.com* from your mobile device. Opera Mini had 194 million users in October 2012, compared to 21 million Opera Mobile users. For Android and iOS devices, you can find Opera Mini in the Google Play Store and the App Store.

Firefox

The Mozilla Foundation arrived a bit late to the mobile browser world. Mozilla offers a downloadable Firefox version for Android and MeeGo (the Nokia N9) and a working version for the Windows 8 Metro UI (optimized for tablets), and it has been updated in sync with the desktop version.

You can download the mobile version of Firefox from stores and from *m.firefox.com*. It uses the same Gecko engine as the Firefox desktop browser and it works on smartphones and tablets.

As we saw in Chapter 1, in 2013, Mozilla has worked on its own operating system, Firefox OS: the browser is preinstalled in the OS as the whole base of the platform.



Three versions of Firefox for Android are available: the latest version, a beta of the next version (both available from Google Play Store), and Aurora, an alpha version of a future release that can be downloaded only from www.mozilla.org/firefox/channel/#aurora.

Dolphin

Dolphin (not to be confused with Samsung's Dolfin) started as a pseudo-browser for Android, adding features to the default engine in the Android web view; this was soon followed by a similar iOS version. The latest versions of this application for Android replace the web view with their own WebKit-based engine, which has one of the top scores in terms of HTML5 compatibility and performance. You can download Dolphin from Apple's App Store, the Google Play Store, or the Dolphin website (*http://dolphin-browser.com*).

Baidu Browser

Baidu, the most-used search engine in China, has released a browser for Android- and Windows Phone–based devices that can be downloaded from *shouji.baidu.com/browser*. The Baidu Browser uses a WebKit-based engine called *T5*.

SkyFire

SkyFire is a pseudo-browser available for iOS and Android with the particularity of delivering Flash and video content from its own servers, which transcode nonmobile content to be compatible with these platforms. SkyFire is available for download from the App Store and the Google Play Store.



As of December 2010, SkyFire was rebranded as a proxied browser for Android and iOS. Before that, it was a Gecko-based browser for Symbian, BlackBerry, and Windows Mobile. SkyFire was acquired by Opera in 2013.

Browser Overview

That list of browsers is a lot to digest. Table 3-1 compares key features of the most commonly used browsers on the market.

Table 3-1. Features of mobile browsers available on the market as of 2012

Browser	Current platforms	Engine	Proxied	Navigation
Safari	iOS	WebKit	No	Multitouch
Android browser	Android up to 4.1	WebKit	No	Multitouch and focus
Chrome	Android > 4.0 (iOS as a pseudo-browser)	WebKit	No	Multitouch and focus
Nokia Browser	S40 6th Edition	WebKit	No	Cursor, touch, and focus
	S40 new platform (Nokia Xpress browser)	Gecko	Yes	Focus, touch, and multitouch
	Symbian	WebKit	No	Cursor, multitouch, and focus
	MeeGo	WebKit2	No	Multitouch
webOS browser	webOS, Open webOS	WebKit	No	Multitouch and focus
BlackBerry browser	BlackBerry OS 5.x	Custom	In some situations	Cursor and multitouch
	BlackBerry OS 6.0, 7.0	WebKit	No	Cursor and multitouch
	Tablet OS / BB10	WebKit	No	Multitouch and cursor
Internet Explorer	Windows Phone, Windows	Trident	No	Multitouch
Firefox	Android, MeeGo, Firefox OS	Gecko	No	Multitouch
NetFront	Low-end devices	Custom	No	Focus or touch[a]
Opera Mobile	Android, Symbian	Presto/Webkit	Yes/No[b]	Focus
Opera Mini	Android, iOS, Symbian, Java, BlackBerry	Presto/Webkit[c]	Yes	Cursor or touch[d]
Bada Browser	Bada	WebKit	No	Touch
UC Browser	Android, iOS, Java, BlackBerry	Custom	Yes	Multiple

[a] Depending on the device.

[b] Depending on the usage of the optional compression option.

[c] Presto was used before 2013. From version 14, Opera is based on Chromium/WebKit.

[d] Depending on the device.

CHAPTER 4

Tools for Mobile Web Development

Unlike desktop web development, where you're likely to create and test your work on the same device, mobile development generally requires setting up and managing several development environments.

Working with Code

For coding our markup, JavaScript, and CSS, we can use almost any web tool available on the market, including Adobe Dreamweaver, Microsoft Visual Studio, Eclipse, Aptana Studio, and of course any good text editor, such as Sublime Text, Textmate, WebStorm, or Notepad++. In mobile web development, it is often easier and cleaner to work directly with the code rather than using a visual tool or IDE.

Adobe Dreamweaver

Since the CS5.5 version, Dreamweaver has worked better with mobile markup and allows us to validate against mobile web standards. In this editor, when we create a new document we can choose HTML5 as the document type, as shown in Figure 4-1.

Version CS6 includes several enhancements that support mobile web design and development, such as:

- HTML5 support and code hinting
- Multiple screen preview
- jQuery Mobile integration
- PhoneGap Build integration for native web app compilation from the IDE



Figure 4-1. Dreamweaver allows us to define new files as HTML5 or XHTML Mobile Profile documents, as well as giving us the ability to start with a jQuery Mobile template.

Adobe Edge Tools

Adobe offers a group of tools under the name of Edge that help designers and developers to create HTML5 applications. They include:

Edge Code
: A complete HTML5 editor based on web technologies. This tool is based on the open source editor Brackets and it includes several interesting ideas on how to code HTML, CSS, and JavaScript quickly and easily.

Edge Reflow
: A tool that helps designers to create responsive web design solutions.

Edge Inspect
: A tool for mobile HTML5 testing. We'll cover this tool later in this chapter.

Edge Animate
: A tool to design HTML5 animations visually.

You can download these tools from Adobe's website (*http://html.adobe.com*).

Microsoft Visual Studio and WebMatrix

Microsoft IDEs have supported HTML5 syntax and IntelliSense since version 2010 SP1. You can also use WebMatrix (*http://web.ms/WebMatrix*) for mobile web development; it's available for free.

WebMatrix has supported mobile websites since version 2, including:

- Mobile-friendly templates
- Connection with the Windows Phone emulator and iOS simulation through partners, such as Electrium Plumb
- Code completion for HTML5 and the jQuery Mobile UI framework

Eclipse

If you would like to use Eclipse as your development environment, there are several plug-ins you can use to create mobile HTML5 apps. I suggest Aptana from Titanium, a free Eclipse-based IDE for HTML5 and mobile development. You can download a free version from Aptana's website (*http://aptana.com*).

Native Web IDEs

If you are going to target native web or hybrid apps, some platforms offer tools and IDEs you can use to develop, test, and build your final packages. The most important products include:

Nokia Web Tools (http://mobilexweb.com/go/nokiawebtools)
: For testing and compilation of S40 web apps. A legacy 1.2 version that will help with the legacy WRT Symbian format is still available.

Tizen IDE (https://developer.tizen.org/sdk)
: For the creation of Tizen apps based on HTML5.

Intel XDK (http://dev.appmobi.com/?q=node/153)
: A nonofficial for creating tool for creating Apache Cordova HTML5 native apps.

Titanium Studio (http://www.appcelerator.com/platform/titanium-studio/)
: An Eclipse plug-in to create Appcelerator Titanium JavaScript mobile apps.

Testing

Testing with a desktop browser is not good enough. Mobile browsers are really different, and we need to test our mobile apps using tools that are as accurate as possible.

Emulators are very useful and provide a simple, fast, and fairly accurate testing solution. If it doesn't work in the emulator, it probably will not work on the real device, and if it works in the emulator, it probably will work on the real device (yes, again "probably"!).

There are some problems with this testing approach, though. For one thing, there are hundreds of differences between real devices, and hundreds of bugs. Furthermore, there are several platforms without emulation. That is why real device testing is mandatory.

But how can we get access to multiple real devices? Here are a few suggestions:

- Acquire as many friends as you can (with different devices, if possible).
- Buy or rent devices. Some vendors offer promotions for buying or renting devices for developers and their partners.
- Use a testing house company. This is an expensive solution and is not recommended for mobile web developers; we need to be as close as possible to the devices.
- Create a beta tester program, for receiving feedback.
- Access a community mobile lab in your city, see www.opendevicelab.com (*http://opendevicelab.com*) for more information.
- Use a remote device lab.

In Chapter 18, we will review tools and services that will help us to test and measure performance on mobile websites.

Let's first review emulators and simulators before talking about the other solutions we can use.



Mob4Hire (*http://www.mob4hire.com*) is a mobile social network aimed at joining testers with mobile devices around the world and developers who want to test applications or websites using a payment service. You can find a similar solution from UserTesting (*http://www.usertesting.com/mobile*) (at a cost of $39 per tester).

Emulators and Simulators

The most useful tools for our work will be emulators and simulators. Generally speaking, an *emulator* is a piece of software that translates compiled code from an original architecture to the platform where it is running. It allows us to run an operating system and its native applications on another operating system. In the mobile development world, an emulator is a desktop application that emulates mobile device hardware and a mobile operating system, allowing us to test and debug our applications and see how they are working. The browser, and even the operating system, is not aware that it is running on an emulator, so we can execute the same code that we would execute on the real device.



We should also add to our mobile development environments classic tools for project and configuration management, like bug tracking, version control, and project management tools.

Emulators are created by manufacturers and offered to developers for free, either standalone or bundled with the Software Development Kit (SDK) for native development.

There are also operating system emulators that don't represent any real device hardware, but rather the operating system as a whole. These exist for Windows Phone and Android.

On the other hand, a *simulator* is a less complex application that simulates some of the behavior of a device, but does not emulate hardware and does not work over the real operating system. These tools are simpler and less useful than emulators. A simulator may be created by the device manufacturer or by some other company offering a simulation environment for developers. As the simulator does not simulate all the device features, we can also find tools that will be helpful not for mobile web development but rather for other technologies, like Java ME. In mobile browsing, there are simulators with pixel-level simulation, and others that neither create a skin over a typical desktop browser (such as Firefox, Chrome, or Safari) with real typography nor simulate these browsers' rendering engines.

For mobile web development, we will find emulators from Nokia, BlackBerry, Android, webOS, and Windows Phone, and simulators from Apple for iOS (though only for Mac OS X). A multiple mobile browser simulator is available from Adobe, called Device Central, but we will not find any help from Sony Mobile, LG, Motorola, or Samsung with their proprietary OSs (used on their low-end and mid-range devices).



Emulators and simulators don't replace real device testing, but they are useful for UI testing, JavaScript debugging, and testing different scenarios. These tools are useless to test performance, touch interaction, and some hardware scenarios, such as the accelerometer and lighting conditions.

Some browser-based emulators, like the Opera Mini emulator, are also available. An up-to-date list of emulator download URLs can be found at *emulato.rs*.



As the emulators have the same operating system and applications as the real devices, you'll need to wait for the OS to load before opening a web page.

Android emulator

The Android emulator is available in conjunction with the SDK to create native Java applications for Android. You can download it for free from the Android Developer page (*http://developer.android.com*); the base SDK and the different Android OS versions are available separately. The Android emulator is available for Windows, Mac OS X, and Linux. Once you've downloaded it, create a folder for the contents on your hard drive and unzip the package. On Windows, there is an installer version that will do the work for you.

In the folder where you extracted the package, there is an `android` terminal command on Mac OS X/Linux and an *SDK Setup.exe* application for Windows that opens the *Android SDK Manager* shown in Figure 4-2, where you can download and configure Android platforms (known as packages or targets) after installing the base SDK.

You can download as many packages as you want, one per operating system version; you can even download vendor-specific emulators, such as for the Motorola Xoom 2, LG Optimus 3D, or Galaxy Tab. Try to download the latest releases of every Android version, such as Android 2.3.3, Android 4.0, and Android 4.1.

Android on Intel Computers

Most of the Android version SDKs execute slowly on Intel computers, as they are emulating a whole different hardware. Fortunately, now we have an Intel-based porting of Android that will work much faster on our development computers. You can download it from the Android SDK Manager, searching for the package named "Intel Atom x86 Image."

You can also find a different project at www.android-x86.org that was ported to Android to work as a whole operating system on Intel machines (remember, Android is Linux-based). Therefore, you can run this Android x86 operating system as a virtual machine, like VMWare or VirtualBox.



Figure 4-2. After downloading the Android SDK, open the SDK Manager and download the platforms you want—the Google APIs are needed for native development using Google's services.

Opening the Android emulator can be a little tricky the first time. You can open it from an IDE such as Eclipse, but first you need to install the Android plug-in and create a native empty application. Alternatively, you can open the emulator from a console window (Terminal or the command prompt, depending on the operating system) or from the AVD (Android virtual device) Manager. The AVD Manager can be opened from the SDK Manager, using the Tools→Manage AVDs menu option.

Once you've installed a platform, you need to create a new virtual device using the AVD Manager. Creating a new device involves selecting the target (of the installed packages), defining a name, and specifying the size of the SD card, the screen size, and other optional hardware features, as you can see in Figure 4-3. To understand the screen size names, refer back to Table 2-1.



Figure 4-3. After installing the SDK and the platform, you must create virtual devices for each platform and screen combination you need.



If you add external sites in the Android SDK Manager, you can install third-party Android emulators. For example, you can install Kindle Fire emulators by adding *kindle-sdk.s3.amazonaws.com/addon.xml*.

One you've created the device, you can select it and click Start to reach a result like the one shown in Figure 4-4.



Figure 4-4. At this point, you can open the browser as if you were on a touch-enabled Android device and use the mouse over the emulator's screen to emulate the user's gestures.

When you start the Android virtual device (AVD), you will be prompted with an opening configuration window, as seen in Figure 4-5. In this Launch Options window you can scale the emulator if it's bigger than your own computer screen—a possible situation when opening tablet emulators—using the "Scale display to real size" option. If you want to delete all the settings and applications installed on that emulator, you can use the option "Wipe user data."



Figure 4-5. When opening an AVD we need to select some configuration attributes, such as display scaling.



When using the Android emulator, you can use the shortcuts Ctrl-F11 and Ctrl-F12 to change the emulator's orientation.

With the emulator opened, you can open a mobile website by finding the browser using your mouse (remember that almost all Android devices are touch-capable) and typing the URL in its location bar. The emulator doesn't support opening local files directly using the *file://* protocol, so you'll need to set up a local web server (for example, Apache) or upload your files to a web server on the Internet.



If you want to load a local web server in the Android emulator, you can't use *localhost* or 127.0.0.1 because the browser will point the request to Android itself. There is a special IP address available to point to the host computer: 10.0.2.2.

Up to version 4.0, the Android emulator comes with only the Android browser. You can install other browsers if you find the installation packages at the Google Play Store. Inside the emulator, you can download other browsers from the URL www.mobilex-web.com/go/android-browsers.



The Android Emulator doesn't officially include the Google Play Store. Therefore, there is no way to download or buy native apps there. If you want to install other apps—such as an alternative browser—you need to look for the installation package (*.apk* file) on a different source.

iOS Simulator

Available for only Mac OS, the iOS Simulator (shown in Figure 4-6) offers a free simulation environment for the iPhone and iPad, including the mobile browser Safari. It is not an emulator, so it does not really provide a hardware emulation experience and is not a true performance indicator. However, it is perfectly suitable for seeing how your website is rendering and how your code is working. It's especially convenient for loading local or remote files by typing in the URL field using your desktop keyboard.

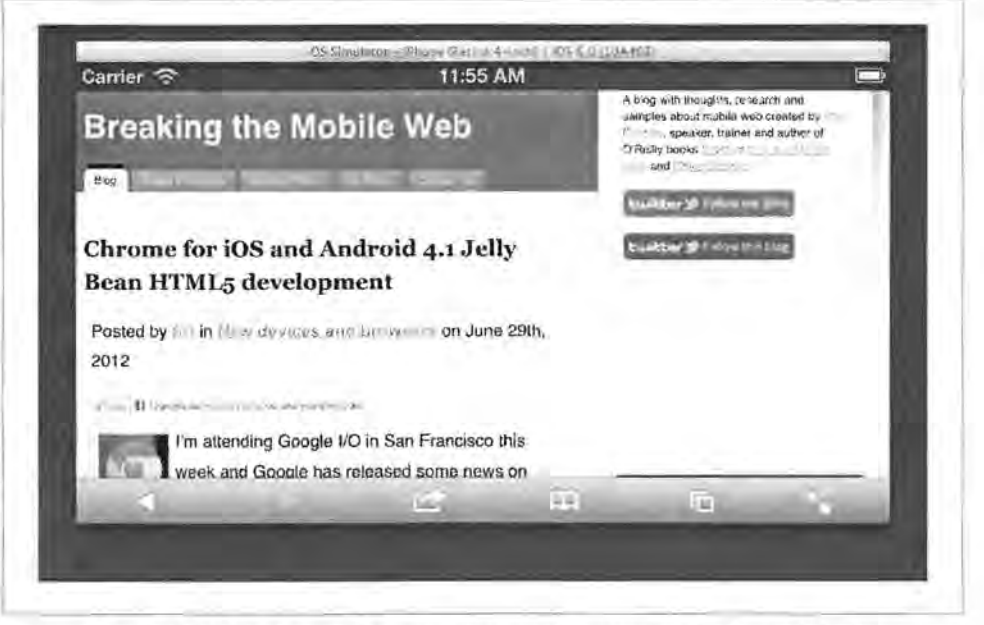


Figure 4-6. The iOS Simulator allows us to rotate the screen as in the real device.



Some websites, such as www.TestiPhone.com and www.iPhoneTester.com, try to simulate the iPhone browser, but the experience isn't the real thing; they are just iframes with the skin of the iPhone.

The iOS Simulator is included with the SDK for native development, available for free at the Mac App Store (search for Xcode) or at from Apple's website (*http://developer.apple.com/ios*). The SDK may take a while to download, because it's about 1.5 GB.

You will always download the latest version of the operating system and can then add previous versions (such as 6.0), in which case you can switch between versions using the Hardware→Version menu option.

To download a previous version of the operating system to the simulator, you need to open the Xcode app, open Preferences, and select Downloads, as seen in Figure 4-7.



Figure 4-7. Opening Preferences in Xcode allows us to download previous iOS versions that will be available later on the simulator.



When simulating high-resolution devices, we can change the scale of the simulator using the Window→Scale menu or using the hot keys with ⌘-1, with ⌘-2, and with ⌘-3.

Within the Simulator, you can also select what device you want to simulate using the Hardware→Device menu option. Options include:

iPhone
: Emulates low-resolution iPhone and iPod Touch devices, such as the iPhone 3GS

iPhone Retina 3.5-inch
: Emulates high-resolution 3.5"-screen iPhone and iPod touch devices, such as the iPhone 4S

iPhone Retina 4-inch
: Emulates 4"-screen iPhone and iPod touch devices, such as the iPhone 5

iPad
: Emulates low-resolution iPads, such as the iPad 2 and iPad Mini first generation

iPad Retina
: Emulates high-resolution iPads, such as the third and fourth generation iPads



At the time of this writing, there is no way to emulate the real iPhone browser on Windows or Linux machines. There are some free and commercial tools that will help you simulate some behaviors on Windows, however, such as the Electric Mobile Simulator (*http://electricplum.com*), the MobiOne emulator (*http://www.genuitec.com/mobile*), and BrowseEmAll (*http://browseemall.com*).

Once the emulator is open, you can open the Safari application and type a URL in the address bar. To open a local file, use the *file:///* protocol in the address field (for example, *file:///Users/myUser/Desktop/test.html* to open an HTML file on the desktop of the *myUser* user). You can also drag and drop an HTML file from the desktop to the Simulator while it's opened to browse it.

Pasting a URL from the clipboard can be a little tricky. When you paste text using the keyboard or the Edit menu, the text will be pasted into the iPhone's internal clipboard. You then need to paste it again using the iPhone's gesture, tapping once over the text input and selecting Paste from the contextual menu (as shown in Figure 4-8), or use the Edit→Paste Text menu option.



Figure 4-8. You can use your desktop keyboard or Edit→Paste to paste text to the iPhone's clipboard, and then tap once on the text input and press Paste on the screen to paste it where you want it to go.



There is no App Store available in the iOS Simulator, meaning that you can't download alternative browsers or pseudo-browsers such as Opera Mini or Google Chrome for iOS.

Nokia emulators

Nokia has always had the better emulators, since the beginning of mobile web development. Instead of one emulator per device, you'll find one emulator for each version of each platform. You can download emulators for Series 40 devices (feature phones and social devices) and for Symbian smartphones at the Nokia forum (*http://www.forum.nokia.com*).



Nokia also has a tool called the Nokia Mobile Browser Simulator, developed in 2003 to test mobile websites for old WAP 1.0 devices and the first WAP 2.0 ones. Today, this tool is still available but deprecated; we don't need it.

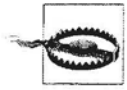
Unfortunately, Nokia emulators, like that shown in Figure 4-9, are available only for the Windows operating system.



Figure 4-9. A touch-based browser running in a Nokia S40 emulator—if you use File→Open, you must type http:// first.

If you need to emulate a Nokia device, first find the correct platform version for that device at Nokia's website (*http://developer.nokia.com/devices*) and then download the emulator for that platform. Nokia guarantees (and it works almost all the time) that every device based on the same platform version has the same browser and rendering engine and even the same hardware features.

The Nokia emulators will add shortcut icons to your Start menu, so it will be easy to find them. Once you've launched the emulator, you can open the browser and type in the URL or use the shortcut File→Open, which allows you to type or paste a URL or browse for a file in your local filesystem. The emulator will open the browser automatically.



Some of the latest S40 emulators have predictive text input active by default, and this will deactivate the usage of your desktop QWERTY keyboard to type. Before using them, you'll need to disable predictive input.

Nokia S40 emulators support the use of *localhost* or 127.0.0.1 to connect with your desktop host computer.

Running Mac OS or Linux?

If you are taking mobile web programming for multiple devices seriously, it will be very useful to have at least one development desktop with Windows, even if it is on a virtual PC. Some emulators work only in Windows environments. Hopefully this will change with time; emulators for Mac OS X and Linux are already available for some platforms.

BlackBerry simulators

Research in Motion (RIM), vendor of the popular BlackBerry devices, has two different tools available for web developers: emulators and a simulator for web apps known as Ripple.

RIM has done a great job with emulators, with only one problem: it is very difficult to decide which one to download and use. Dozens of different installers are available at BlackBerry's developer site (*http://www.blackberry.com/developers*); you can download the proxy server and the emulators. The BlackBerry Smartphone Simulators (for BlackBerry OS versions up to 7.1) are compatible only with the Windows operating system, but the emulators for BB10 and PlayBook are also available for the Mac and Linux platforms.

Ripple. Ripple (Figure 4-10) is a free tool available as a Google Chrome for desktop plug-in that helps us test HTML5 web content and WebWorks native web applications in a simulation environment. It's available for free (*http://mobilexweb.com/go/ripple*) and it's compatible with Mac and Windows. There is also a standalone version that may be deprecated in the future, based on Chromium.



Figure 4-10. Ripple is a free plug-in for Google Chrome for desktops that allows us to simulate some mobile devices, such as BlackBerrys, and native web platforms, such as Apache Cordova/PhoneGap or WebWorks.

You can simulate different scenarios, from BlackBerry 7 to PlayBook and the newest BlackBerry 10 platform, and it includes mobile web support and WebWorks support (adding support for native web API testing). While Ripple is good for a first testing, remember that it is really the Chrome engine, not the real web engine running on BlackBerry devices. Also, Ripple requires an HTTP connection (local or external), so you cannot just open files from the local filesystem.

BlackBerry smartphones. The first requirement for older emulators is to download the BlackBerry Email and MDS Services Simulator Package. This proxy allows any simulator to access the network and emulates email services and an enterprise server. Before opening a browser, you need to start this service on your computer.

The BlackBerry Smartphone Simulators (*http://na.blackberry.com/eng/developers/resources/simulators.jsp*) are available. The first step is to select the smartphone you want to emulate (for example, the BlackBerry Tour 9630) and choose either the carrier you want (or Generic), or the OS version. One example of a BlackBerry simulator is shown in Figure 4-11.



Figure 4-11. Some BlackBerry simulators are pointer-based, so you need to use the onscreen keys or the arrow keys on your desktop keyboard; others are touch-based, so you can use your mouse on the screen.

Once you've installed your emulator, you can launch it, open the browser, and type the URL you want to access (if it's an older version, remember to open the BlackBerry MDS Services Simulator before launching the emulator!). These emulators don't support local files or accessing them through *localhost*; you can use the local IP address of your

desktop if you're on a network or the public IP address if you are connected directly to the Internet.

PlayBook and BB10. The BlackBerry PlayBook tablet and BlackBerry 10 emulators are available for different operating systems. Before installing them, you need to have installed VMWare Player (*http://www.vmware.com/products/player*) on Windows or VMWare Fusion on Mac (*http://www.vmware.com/products/fusion*).

Both the PlayBook and BB10 emulators are virtual machines, and you can use the whole operating system, including the BlackBerry browser to test your web applications. If you want to open local files, you need to set up a server and verify on your VMWare which IP address is your host machine.

webOS emulator

Palm has been in the emulator market for more than 10 years and has always had great support for these tools. We have already talked about the history of Palm and Palm OS; in this book we will cover only webOS, the operating system available since Palm Pre. You can download the Palm SDK (*http://developer.palm.com*), which includes the Palm emulator. It is available for Windows, Mac OS X, and Linux. To use it, you must have VirtualBox (*http://www.virtualbox.org*), a free virtualization tool, installed on your machine. If everything goes OK, you can open the webOS emulator from the Start menu, the command line/Terminal, or your applications list.



For the future, all the webOS emulation tools will be available as an open source project from Open WebOS Project (*http://openwebosproject.org*).

Windows emulators

If you want to test your applications on Windows Phone, you can download the free Windows Phone SDK or buy a license of Visual Studio. The Windows Phone emulator (Figure 4-12) comes with the SDK (*http://mobilexweb.com/go/wpsdk*) and includes the current version of Internet Explorer to test web content.



Figure 4-12. The Windows Phone 7.x emulator needs a Windows 7 desktop environment and the Windows Phone 8 emulator needs a Windows 8 desktop environment.



The Windows Phone emulator is compatible with only Windows Vista SP2, Windows 7, or Windows 8 and requires a graphic driver with WDDM 1.1 support. You can check your hardware specifications to verify whether your graphic driver is compatible. If not, you will see the emulator, but you will see only a white page when trying to load Internet Explorer.

If you want to emulate Windows 8 for tablets, you have two options:

- Use your own Windows 8 for desktop environment.
- Use the Windows Simulator included with Visual Studio for Windows 8 (even with Express, a free version of the IDE). It includes Internet Explorer 10.

The Windows Simulator works only on Windows 8 desktop machines; it emulates a tablet touch environment, where you can emulate touch gestures, geolocation, and different screen sizes and orientations.



If you are looking for the legacy Windows Mobile emulation, you can find a guide from the first edition of this book (*http://mobilexweb.com/go/winmobemu*).

Opera Mobile Emulator

In 2010 Opera released the first emulator for its Opera Mobile browser, available for Mac OS X, Linux, and Windows. The emulator runs the exact same code as the mobile version, so it is accurate. With this emulator you can also debug your mobile web applications using Dragonfly, a debugging service for Opera that we will cover in Chapter 18.

As you can see in Figure 4-13, this emulator comes with different real mobile device profiles, and you can create your own combinations of screen resolution, pixel density, and type (touch, keypad, or tablet).



Figure 4-13. With the Opera Mobile Emulator, we can select a device profile or create our own profile by defining each property's values.

You can download the emulator for free at *www.opera.com/developer/tools*.

Opera Mini Simulator. You can enjoy a full Opera Mini simulation (*http://www.opera.com/mini/demo*) in a Java applet (see Figure 4-14). This URL is for the latest version of the software (at the time of this writing, 7.0).



Remember that Opera Mini and other user-installable browsers are available as normal native or Java ME applications, so you can use any emulator to download them. The Nokia, Android, and BlackBerry emulators are great for this purpose.



Figure 4-14. The Opera Mini Simulator is an online free service running the same Java browser as the one on real devices.

Other official emulation platforms

Other platforms have their own emulators, but they are often not so simple to use or are not optimized for web development:

- The Tizen emulator (*http://developer.tizen.org*) is part of the SDK.
- The Firefox OS emulator (*https://addons.mozilla.org/es/firefox/addon/firefox-os-simulator/*) is available as a Firefox plug-in
- The Bada emulator (*http://developer.bada.com/devtools*), part of the native SDK, is not prepared for web testing.
- The MeeGo emulator (*http://developer.meego.com/meego-sdk*) is prepared for netbooks and smartphones.

Adobe Device Central

This tool was included with Adobe Dreamweaver, Adobe Flash Professional, and some of the suites up to CS5.5, and it had an updated list of devices, including their screen sizes and Flash Lite capabilities. However, for browser emulation it is just a miniature

WebKit browser on the desktop. It doesn't provide real (or even almost-similar) simulation in terms of typography, browser bars, and markup rendering. From CS6, Adobe has abandoned the project.

Keynote MITE

The Mobile Interactive Testing Environment (MITE) is a piece of software from Keynote for testing, validating, and monitoring mobile websites using thousands of simulated devices. You can download it from *mite.keynote.com*.

Comparison

Table 4-1 shows how the different platform emulators and simulators allow us to access files and the clipboard on our host machines.

Table 4-1. Comparison of available emulators and simulators

Platform	Able to open local files	Accesses host's local server via	Devices	OS compatibility
Android	No	10.0.2.2	Smartphones and tablets	Windows, Mac, and Linux
iOS	Yes	*localhost*	Smartphones and tablets	Mac
Nokia S40	Yes	*localhost*	Phones	Windows
Windows Phone	Yes	*localhost*	Smartphones	Windows
BlackBerry	No	Network IP address	Smartphones	Windows
BlackBerry PlayBook/ BB10	No	VMWare IP address	Smartphones and tablets	Windows, Mac, and Linux
webOS	No	Virtual box IP address	Smartphones and tablets	Windows, Mac, and Linux
Opera Mobile	Yes	*localhost*	Smartphones and tablets	Windows and Mac



For emulators without URL pasting abilities, you can generate a free mobile-optimized short URL for easy typing on a mobile device or in an emulator at Mobile Tiny URL (*http://www.mobiletinyurl.com*).

Remote emulation services

The cloud has come to us to solve some problems: now, we don't need to have everything installed on our own computers. In this case of mobile web development, there are some cloud-based solutions that we can use for mobile testing.

BrowserStack. BrowserStack (*http://browserstack.com*) is a cloud-based service optimized for cross-browser testing. While it's been useful for desktop web testing for a while, it recently added mobile web browsers. Instead of having emulators installed on

our own development machines, we can use this web-based solution and use a remote emulator or simulator.



With BrowserStack we can use the official iOS Simulator on Windows and Linux machines. You can request a trial account at BrowserStack's website (*http://browserstack.com*).

BrowserStack includes emulation of Safari on iOS, the Android browser, and Opera through different device profiles, such as the iPhone 3GS, iPhone 4S, iPad, Samsung Galaxy SII, HTC Evo 3D, and Motorola Xoom (see Figure 4-15).

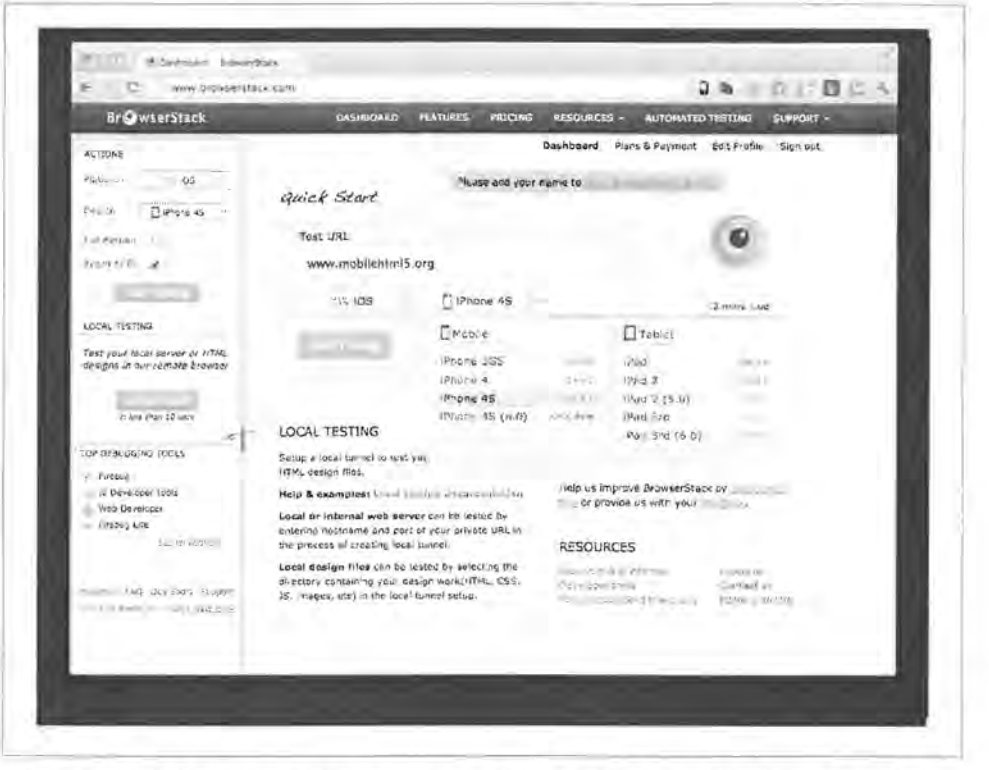


Figure 4-15. With BrowserStack we can emulate Android and iOS devices remotely—even the iOS Simulator on Windows computers.

Using a Java-based tunnel, we can test local files on our development machines, or on a private server that is not accessible through the public Internet.

Browshot. Browshot allows us to take screenshots of our mobile websites with iOS, Android, and BlackBerry devices. While we can't interact with the emulations, sometimes a screenshot is enough. You can request an account at Browshot's website (*http://browshot.com*).

Real Device Testing

There is nothing like real devices when testing mobile websites. You will find differences not only in performance but also in behavior, like when you use your fingers to navigate and not a precise mouse pointer. And while creating your own testing lab is ideal, it's also expensive and needs to be updated frequently. At the time of this writing, I currently have around 45 devices for testing.

If you have a limited budget, you should try to buy one key device per platform, and if you are targeting tablets you should get one—don't rely on smartphones for tablet testing as the browsers don't act the same way.



While you can potentially copy your web files to the memory of your phone and open them locally, it's not a good idea; using an HTTP server—local or remote—is the way to do it.

When you have a real device, the first question is how to easily test your web development. The answer is through a web server. If your device supports WiFi (almost every social device, smartphone, and tablet supports WLAN at the time of this writing), you can run a web server on your computer—such as Apache—and access it from your device using your local IP address. Remember that your computer must be connected to the same network and you should have a firewall or a router allowing internal port connections.

On Windows hosts you can get your IP address by opening a command line/terminal (from the Start menu, type `cmd`) and using the `ipconfig` command. On Mac hosts you can get your IP address from System Preferences→Network.



Typing IP addresses on mobile devices can be a little complicated. You can create a shorter version of your URL and even generate a QR code for camera scanning using the free service Mobile Tiny URL (*http://mobiletinyurl.com*).

If you want to test your mobile website on cellular network or you have no possibility to run a local web server on your development environment, you can always use a normal shared web service to host your files.

Adobe Edge Inspect

When you have real iOS and Android devices, you can use the tool Adobe Edge Inspect —formerly known as Adobe Shadow—to help with your testing and debugging. Adobe Edge Inspect is a solution involving different applications that work together and help you carry out multidevice testing with almost no effort. All the information and download links are available at Adobe Edge Inspect's website (*http://adobe.com/go/edgeinspect*).

The tool is available as part of the Adobe Creative Cloud services. If you have a free account, you can connect with one device at a time. If you have a premium account, you can connect multiple devices simultaneously.

To make it work, you need to download the following parts:

- Google Chrome for Windows or Mac and the Edge Inspect extension, available at the Chrome Web Store
- The Edge Inspect server for Windows or Mac, connected to your Adobe Creative Cloud account
- The Edge Inspect client, available for iOS via the App Store, for Android via the Google Play Store, and for the Kindle Fire via the Amazon Appstore

When you have everything installed, you can open the Edge Inspect app (the server) and Chrome on your desktop computer. Then, on your mobile devices, open the Edge Inspect mobile app (the client), which will automatically try to find the server on your local network.



It's important to understand that both the desktop computer (the server) and the mobile device (the client) must be connected to the same local network in order for Edge Inspect to work.

If your device can't connect to your computer, you can add it manually. Once connected, Chrome for desktop and all your connected devices will be synchronized, as seen in Figure 4-16. That means that browsing to a website in Chrome on your Windows or Mac machine will automatically fire a browsing to the same URL on every device connected to Edge Inspect.



Figure 4-16. Adobe Edge Inspect allows us to keep our desktop browsing experience in sync with a real Android or iOS device, take screenshots, and do basic HTML and CSS debugging.

Edge Inspect supports some advanced features, such as:

- Mobile screenshots
- Debugging your HTML and CSS dynamically through a Weinre session (covered in Chapter 18)
- HTTP authentication
- Clearing the cache on your devices



Adobe Edge Inspect does not use the browser on your iOS and Android devices, but rather the default web view engine. That means that you can't use Edge Inspect to test on a different browser on your mobile device, such as Chrome or Firefox. While mostly similar, some behaviors on the web view differ from the browser (for example, the iOS web view does not support the Nitro JavaScript engine).

Remote Labs

"Any sufficiently advanced technology is indistinguishable from magic," said sci-fi writer Arthur C. Clarke in 1961. When I demonstrate some of these remote labs in my classes, I see a lot of astonished faces.

A remote lab is a web service that allows us to use a real device remotely without being physically in the same place. It is a simple but very powerful solution that gives us access to thousands of real devices, connected to real networks all over the world, with a single click. You can think of it as a remote desktop for mobile phones.

There are three kinds of remote lab solutions for mobile devices:

- Software-based solutions, using a resident application on the device that captures the screen, sends it to the server, and emulates keyboard input or touches on the screen
- Hardware-based solutions, using some technology (magic, I believe) to connect the server to the hardware components of the device (screen, touch screen, keypad, lights, audio, and so on)
- Mixed solutions, having some hardware connection, some software additions, and maybe a video camera for screen recording



As these are real devices, only one user can make use of them at any given time. As such, the devices are a limited resource.

In terms of services provided, we have two kinds of solutions:

- Free usage, where we can use the remote device freely. That is, we can install apps, browse the Web, and use the whole operating system.
- Closed usage, where we can just run a series of tests available on our website or app and receive a report. We can't interact with the device.

Let's take a look at some of the remote lab solutions currently on the market.

Nokia Remote Device Access

Nokia offers a free (yes, free!) remote lab solution for Symbian, MeeGo, and S40 devices called Remote Device Access (RDA), shown in Figures 4-17 and 4-18. To use the service, go to www.mobilexweb.com/go/rda (you'll need to have already created a free Nokia account). You will need Java Runtime 5.0 or newer, because RDA is a WebStart Java application.



Figure 4-17. Remote Device Access is a free and simple way to test on real Symbian, S40, MeeGo, and Windows Phone devices.

At present, usage is limited to eight hours per day. The main features are:

- Complete usage of the device
- 3G and WiFi connection support
- Application installation
- Device rebooting
- Changing screen orientation
- Browser and app support
- Reservation of devices for future usage
- Usage of devices with SIM cards connected in Europe
- Saving screenshot images
- Incoming calls and SMS available



Figure 4-18. The devices are connected to real 3G networks (you can even call them), so you can accurately test speeds and transfers.

At the time of this writing, there are more than 200 devices available. There is no audio or accelerometer support, and depending on your network bandwidth you can select the video quality you want.



AppThwack (*https://appthwack.com*) is a commercial service that uses the virtual test lab idea to work with real Android and iOS devices. It has an option for mobile web testing (available only on Android at the time of this writing): give it a URL, and it will generate dozens of screenshots of that website in different browsers—the Android browser, Opera Mobile, Opera Mini, Firefox, Chrome, and Dolphin—on different real devices.

Samsung Remote Test Lab

Samsung also offers a free remote lab web service, similar to Nokia's RDA, called Remote Test Lab (RTL). RTL, shown in Figure 4-19, includes Android smartphones, Android

tablets, and Bada phones at the time of this writing. The devices don't have SIM cards, though, so you can test only WiFi connections (not 3G).



Figure 4-19. Samsung's Remote Test Lab is a free service where you can use real Android and Bada devices remotely.

You can access this remote lab from www.mobilexweb.com/go/labdev.

Keynote DeviceAnywhere

Keynote DeviceAnywhere is the leader and pioneer in remote lab solutions for mobile testing. It offers a hardware solution that allows any device (low-end, mid-range, or smartphone, from any vendor) to plug into the architecture.

The company offers a product called *Test Center Developer*, with different price models depending on the package. DeviceAnywhere Test Center offers more than 2,000 devices (iOS, Android, Nokia, Motorola, Sony Ericsson, Samsung, BlackBerry, LG, Sanyo, Sharp, HTC, and more) connected to more than 30 live networks all over the world.

You can apply for a free trial at the DeviceAnywhere website (*http://www.keynotedeviceanywhere.com*). The IDE (DeviceAnywhere Studio) is a Java application, so it should work on any OS. Easy-to-install packages are available for Windows and Mac OS X. The company offers a special free plan prepared for mobile web testing of only the 10 most

popular devices. The free service doesn't require the IDE; it uses a web environment for the testing.

The solution includes:

- Access to the registered packages and devices from DA Studio
- Access to all hardware features (lock/unlock, close and open, change orientation, power off and on)
- Ability to place calls, send and receive SMS messages, access the carrier's portal, and buy premium content (as the devices are on live networks)
- Pixel-based perfect image rendering, so you can save screenshots and videos of your testing for offline review (audio is also supported as an optional feature)
- Ability to manage multiple devices at the same time
- Virtual onscreen keyboard, and shortcuts to use your own desktop keyboard for testing
- Team management for testing a device and sharing the screen with other users
- DOM Inspector and HTTP headers viewer using an included proxy

Many manufacturers and carriers have selected DeviceAnywhere as the official testing solution for their Virtual Developer Labs (VDLs). Some of the Virtual Lab solutions include:

- Nokia VDL (Series 40 and Symbian)
- Sony Ericsson VDL
- Palm VDL (Palm OS, Windows Mobile, and webOS)
- Motorola VDL (Motorola OS, Windows Mobile, and Android)
- BlackBerry VDL

To use the full DeviceAnywhere service, you'll need to subscribe to one or more packages. At the time of this writing, a yearly contract is needed, and on top of the monthly subscription fee (starting at $100) you will either pay on a per-hour basis or subscribe to a prepaid plan. On a per-hour basis, the maximum price is $16/hr. There are also other promotions available on the website, and different manufacturers' VDLs can have different pricing models.

The time spent on the system is calculated beginning from when you open a device and finishing when you release it, in six minute minimum time slots.

For web development testing purposes, Keynote offers a free option that allows you to test websites on real browsers in an unlimited quantity of 10-minute sessions, on the most popular devices. It's a good service to start working with the system, and if you

decide you need more devices or more time, you can upgrade your account to a professional or corporate plan.



If you apply for a free trial, you will get three hours of free usage and you will have to enter valid credit card details. It is safe to add this information, and it is a requirement because this is a live network where you can buy premium content.

Usage for mobile web testing. As DeviceAnywhere uses real devices from different manufacturers, you will need to learn to use every operating system interface to access the web browsers. You will generally find an icon in the Home screen or applications menu labeled "Browser," "Internet," or even the name of the carrier's online service (for example, "MediaNET," the AT&T Wireless service).

When in the mobile browser, you will need to type your URL using the phone's features: a numeric keyboard, a QWERTY keyboard, or an onscreen touch keyboard. DeviceAnywhere also offers a feature where you can type or paste any URL and then press a button to automatically generate all the keypresses required on the hardware to type the URL.

In numeric keypad devices, the URL typing process can be slow, so it's better if you first minimize the URL using a shortener service, like www.mobiletinyurl.com.

For mobile web debugging purposes, DeviceAnywhere includes an excellent proxy-based browsing solution that brings into the IDE an HTTP sniffer and a DOM Inspector so you can see what markup is actually rendered on the device.



Remember that these are real devices on real networks. If you want to test an application or installable widget, you will need to first upload it to a web server (DA offers a solution) and then access the URL from the browser, typing it or sending it by SMS to the device.

Testing automation. DeviceAnywhere offers many advanced features. One of them is testing automation, a premium service that allows you to create testing scripts and schedule them to be tested on several devices automatically. You can then access the results via a web report.

Perfecto Mobile

Perfecto Mobile (*http://www.perfectomobile.com*) is a company offering a software/hardware hybrid solution for mobile testing, shown in Figure 4-20. Perfecto Mobile uses

a video camera for screen recording on some devices. A good point for Perfecto Mobile is that the whole environment is built on top of the Adobe Flash Player, so you don't need to install anything, and it works from any desktop browser. You can try the system by registering for a free trial; it will be activated in minutes.

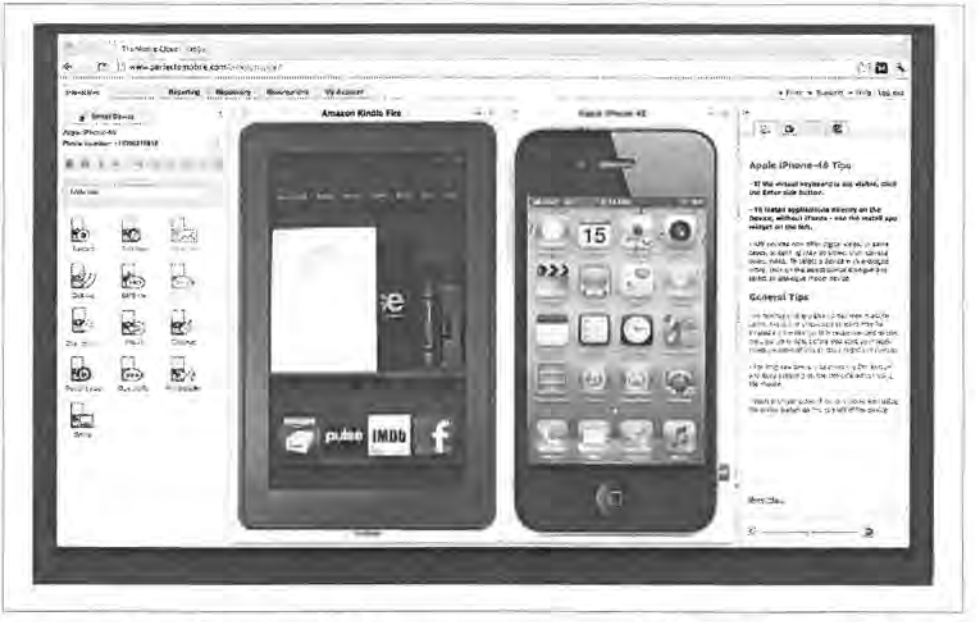


Figure 4-20. With Perfecto Mobile you can manage real phones (here, a Kindle Fire and an iPhone 4S) with a Flash-compatible desktop browser.

With this service, you have access to the whole list of devices and carriers from the same pricing policy. The devices are on real networks in Canada, Israel, the US, the UK, and France.



If you are using nontouch devices, for website scrolling it is better to have a key pressed down for a long time. You can emulate this using the Control key on your desktop keyboard.

Pricing structure. Perfecto Mobile has a simple pricing model. You can access the full cloud of devices with a prepaid monthly plan starting at $17/hr with a minimum 10 hours, or buy a flexible per-project solution at $250 per 10 hours. There is also a flat-rate plan where you can use the full cloud for as many hours as you need.

Sometimes Perfecto Mobile runs special promotions per platform, such as BlackBerry-only or Android-only devices at very low prices.



In both DeviceAnywhere and Perfecto Mobile, you can use two or more devices at the same time. However, your per-minute charges will be counted separately, so you will be charged for two or more minutes at a time.

Main features. The main features of Perfecto Mobile for mobile web testing are:

- When you take screenshots it uses the real screen image, not the camera image.
- You can record videos and share or embed them easily.
- You can send an SMS or invoke a call to the device from the UI.
- You can transfer files to the device (if file transfer is supported).
- There is an OTA mechanism where you can upload your app or widget and the device will receive an SMS link to download it within a 15-minute time slot.
- You can easily share a URL, so customers and coworkers can see what you are doing with the device via live streaming. The only requirement for the other parties is that they use a browser with Adobe Flash Player support.
- You can request Automation, a macro-like recording feature that supports advanced actions and wait conditions using screen recognition and OCR (for example, "go to this URL, wait for the word 'Hello' to appear on the screen, then take a snapshot").
- In Automation, there is a ScriptOnce technology that includes multiplatform templates for common actions.
- You can test how your mobile website is rendering on multiple devices at the same time without your intervention. This feature, called Website Validation, is available on premium plans.

Production Environment

The mobile production environment, surprisingly, doesn't differ too much from a classic web environment. Although many web hosting companies used to offer a "premium WAP hosting" option (obviously, more expensive than the nonmobile options), there is no need for any such distinction.

Web Hosting

To get started, you will need a web server with your favorite platform installed. It should support either static or dynamic files on all platforms you plan to work with (PHP, ASP.NET, Java, Ruby, Python, and so on). Cloud hosting (via a service like Amazon EC2, Google App Engine, Aptana Cloud, or Microsoft Azure) will work well, too.

You will need to have permissions to manage MIME types on the server. We will talk about this in Chapter 17, but for now, just remember that it will allow you to make compatible mobile websites more easily.



There is no special need to use HTTPS (secure connections) for mobile devices. If you want to, just remember that the most widely accepted certificates are from Thawte and VeriSign.

Domain

Which domain alternative should you use? I have no answer for this; you will have to decide for yourself. You can create a subdomain of your desktop website (if you have one), like *m.mydomain.com*, or you can use the main entry point (*mydomain.com* or *www.mydomain.com*. My only recommendation is that, whatever decision you make, you should try to have the other options available and set up a 301 HTTP Redirect to the domain you've chosen. I've tried myself many times to guess a mobile URL using *m.<anysite>.com* or *<anysite>.mobi*, and you should support that user behavior.



No matter which mobile domain you will be using, remember to create a 301 HTTP Redirect to the chosen one from all the possibilities (*m.yourdomain*, *wap.yourdomain*, *mobile.yourdomain*, and, if possible, *yourdomain.mobi*). You don't want to lose visitors because they couldn't guess your mobile address.

Error Management

You'll need to ensure that your error pages will be mobile compatible. You should be able to configure the default error pages for most common HTTP error codes, like 404 (Page Not Found) and 500 (Internal Server Error), on your server. These files must be mobile compatible; we don't want to waste traffic for the user on a server error or deliver pages that aren't compatible with low-end devices. If you're not sure how to configure the default error pages, ask your server provider.

If you are providing both the desktop and mobile versions of your website from the same domain, you should create dynamic code to detect whether or not the device

accessing your site is a mobile device. In the case of a 500 error, deliver a very simple HTML page for both desktop and mobile users; you won't know whether the problem was in your dynamic platform. You can use responsive web design techniques on the error pages.

Statistics

Statistics about mobile website usage are typically the same as those for desktop usage, but a mobile-friendly tool will be very helpful in understanding mobile-specific features. You can log requests on the server for later processing with a log analysis tool, or you can use a third-party tool for help in your statistics management. We will cover mobile-friendly statistical tools in Chapter 19.

CHAPTER 5

Architecture and Design

While this is not a book about design, understanding some architectural and usability concepts is critical to creating useful mobile services. Many common desktop web design patterns and usability concepts do not apply in a mobile environment.



If you want to inspire yourself, check out Mobile Awesomeness (*http://www.mobileawesomeness.com*) to see some of the best mobile websites available.

Mobile Strategy

When creating mobile web applications, we need to remember that we can create browser-based apps, full-screen web apps, or native web apps. All the types have some architectural rules in common, but there are also some practices that are useful only for a particular type.

My first piece of advice is, "Be responsible." Don't discriminate against your users just because they don't have the mobile browser or device that you prefer. I've seen plenty of websites (even from big companies) showing messages like "You can only browse this site with iOS" or providing a good experience only for WebKit-based browsers. If you browse such websites with modern non-WebKit browsers, such as IE10, Firefox, or Opera, you get an awful outdated version, that may even be not touch optimized when browsing with touch devices.

When to Get Out of the Browser

We know that with mobile web we can create that with the mobile web we can create full-screen web apps and native apps. Determining when to get out of the browser depends on your particular case.

With full-screen web apps you gain more space, as well as an icon in the device's Home screen or applications menu. That can help the user remember your web app in the future; he won't need to type in the URL again.

You get the same visibility advantage with a native app, but you also have the ability to run native code (and extend what the browser can give you in terms of API support), and you win some discoverability, as the user will find your app when searching or browsing inside the application store.

It's up to your project what solution is the best. Usually you will offer at least a browser-based website, and possibly more advanced features through a full-screen or native web app. Never force the user to follow your idea; if you have a website and the user wants to use the browser to access your service, let her do that. Don't remind the user on every entry that she can install the app; do it a couple of times and hide the invitation after that.

Just to give you some examples, the *Financial Times* (*http://app.ft.com*) provides a website for every platform, a full-screen optional experience for some devices, and a native web application for other devices, as you can see in Figure 5-1.



Figure 5-1. The Financial Times offers a mobile website for some (first image), a full-screen web app for iOS, and a native web application through the stores for Android and other platforms.

Similarly, Wikipedia (Figure 5-2) has a mobile website for all platforms and a native web app—created with Apache Cordova—available for iOS through the App Store, BlackBerry through App World, and Android through the Google Play Store. This app received more than 6 million downloads in the first few months after its release.



Figure 5-2. Wikipedia has an open source native web application using Apache Cordova as the web view wrapper, available for iOS, Android, BlackBerry PlayBook, and other platforms.



If you are providing a full-screen web app or a native app, every platform will have a way to invite the user to install it while browsing your website. On some platforms you can even send parameters from the website to the native app so the user can continue the experience on the native side using the same context as the website (for example, continuing to read the same article).

Context

Remember that a mobile user has a different context than a desktop user. You should think about and define your users' possible contexts:

- Where is the user?
- Why is the user accessing your mobile website?
- What is the user looking for?
- What can you offer from a mobile perspective to help solve the user's problem?
- Where will the user be when accessing your website? Walking on the street, using public transportation, at the office, traveling as a tourist, sitting on the couch watching TV?

The context will tell you many things about your navigation, use cases, and the usability needs for your mobile site.

A Bad Example of Navigation

I have imprinted in my mind a bad example for navigation in a mobile website. In my city, years ago (but still online today), there was a free mobile web service to get public bus schedules. This is a great service for mobile web users; you are on the move, you need to take the bus, but you're not sure when it will arrive. Should you go for a coffee first?

When you first enter the site, you see a welcome page with a Begin link. After that, you have to choose from a list which bus line you want to query. Then you see a list of final stops for that line, to select your orientation. The first problem is whether you know the name of the final stop in the direction you want to travel. After selecting your orientation, you have to choose the stop or station where you want to get on the bus. There is a list of around 50 addresses, ordered alphabetically. If you don't know the street name for the stop you want, you will need to make an average of 25 clicks to find the right one.

Once you've found your stop, you need to select whether you want a normal bus or a bus with accessibility support. Finally, the service informs you when the next two buses will be arriving. You have to navigate through six pages and choose from a 50-item list to get the result.

In my city, on weekdays during the day buses run very frequently (approximately every five to eight minutes), so what will be the most common context for the service? Probably night and weekend services. The first time I really needed the service was at 1 a.m., coming back from an event. I completed the six-step query only to receive a "There is no information" message. The service did not work at night!

What could be done to improve navigation? Avoid the welcome screen, the accessibility support selection page, and maybe also the direction selection (we can usually guess it from the stop, and even if the stop has service in two directions, information for both can be shown). The stop selection could be improved with features such as a search box, a list filtered by neighborhood, a list of nearby points of interest (museums, cinemas), a location query, a history of stops used before (perhaps using cookies), and so on.

Server-Side Adaptation

A different approach is to create *n* different versions of your site and redirect the user to the appropriate one depending on the device detected. The main problem with this approach is that you need to maintain *n* different versions of the same documents.

If this will be your strategy, expect to need a minimum of four versions for a successful mobile website, with an optional fifth. If you create fewer versions, some users will probably have a bad experience with your site.

Using a server-side adaptation mechanism, you can reduce the number of required versions to two: one for low-end and mid-range devices and one for high-end devices and smartphones. In the high-end and smartphone world it will be better to use an adaptation strategy for the many features that are not compatible with all devices. Broadly, here are the features you will need to consider for each device category:

Low-end devices
: Basic XHTML markup, maximum screen width of 176 pixels, basic CSS support (text color, background color, font size), no JavaScript

Mid-range devices
: Basic XHTML markup, average screen width of 320 pixels, medium CSS support (box model, images), basic JavaScript support (validation, redirection, dialog windows)

High-end devices
: XHTML or HTML 4 markup, average screen width of 240 pixels, advanced CSS support (similar to desktops), Ajax and DOM support, optional touch support, optional orientation change support (for an average screen width of 320 pixels)

Smartphones
: HTML5, large screen size and high resolution, touch support, support for CSS3 (animations, effects) and Ajax, local storage, geolocation

Web app for smartphones
: Same as smartphones, plus offline support, full-screen and icon installation, native integration, and device APIs



I've seen a lot of browser grouping techniques to determine which versions or features we can use in each browser group. The reality is that browsers are so different that there is no way to group them that guarantees all members will have the exact same features. In the next chapters, we will analyze compatibility browser by browser and feature by feature. I suggest you create your own groups based on your code features and the website versions you will design.

Progressive Enhancement

Progressive enhancement is a simple but very powerful technique used in web design that defines layers of compatibility that allow any user to access the basic content, services, and functionality of a website and provide an enhanced experience for browsers with better support of standards.

The term was coined by Steven Champeon (*http://www.hesketh.com*) in 2003, and while this approach wasn't defined specifically for the mobile web, it really is perfect for mobile web design. The concept subverts the typical web design strategy, known as "graceful degradation," where designers develop for the latest technologies and browsers and their designs automatically work with the lesser functions available on older browsers. This technique is not useful for mobile browsers because, as we will see, there are serious compatibility and performance issues in the mobile world. If we develop a website for the latest device (for example, the iPhone), it may not automatically work on other, less advanced or older devices.

Progressive enhancement has the following core principles:

- Basic content is accessible to all browsers.
- Basic functionality is accessible to all browsers.
- Semantic markup contains all content.
- Enhanced layout is provided by externally linked CSS.
- Enhanced behavior is provided by unobtrusive, externally linked JavaScript.
- End user browser preferences are respected.

We will add some other ingredients to this recipe when talking about mobile devices. The objective is to have only one set of code that is compatible with all devices. And, as we are going to discuss later, we must provide the right user experience on every device. We shouldn't create lowest common denominator websites just so that they will be

compatible with all devices, and we shouldn't create overly complex mobile websites that will only work on high-end smartphones.

In the mobile web, a progressive enhancement approach will also include some server-side detection and adaptation that will be mandatory for some specific mobile markup (sending an SMS, for example).



Regressive enhancement is the inverse concept: start by supporting high-end platforms and then add polyfills—libraries that fill gaps—to support older or less-capable platforms. In the mobile ecosystem regressive enhancement is not usually a good idea, because adding more overhead to less-capable platforms impacts directly on already bad performance.

From my point of view, a mobile web design approach should have the following layers, using a progressive enhancement strategy:

1. Create valid and semantic markup containing only the content—no CSS, no frames or iframes, no JavaScript, and no Ajax. All the content and services on the website (with the exception of some nonstandard features, like geolocation) should work with this simple version.
2. Insert in the document any special tags or classes required for device-specific functionality, such as call-to links or a file upload form control.
3. Optionally, from the server, decide which MIME type you will be using (this will be covered in Chapter 17) and recognize the device.
4. Optionally, from the server, replace the special tags inserted in step 2 with real markup depending on the device's capabilities.
5. Add one CSS layer for basic devices, one for high-end devices, and one for some specific smartphones (Android and iPhone devices, for example). You can insert all the markup at the same time using CSS media queries (to be covered in Chapter 11), or use a server-side mechanism to decide which CSS file to apply.
6. Add an unobtrusive basic JavaScript layer for form validation and other basic features.
7. Add an unobtrusive Ajax layer for content updating, capturing the `onclick` event of every link.
8. Add an unobtrusive JavaScript layer and a CSS layer for advanced features (animations, effects, geolocation, offline storage, and so on).
9. Optionally add full-screen installation support using a new layer.
10. Optionally add native web support using a new layer.

We will cover most of these technologies in the next few chapters. The most important part to understand right now is that using this strategy, all devices receive similar markup (with minor changes if we use a server-side adaptation engine); using CSS and JavaScript, we add layers of behavior and design adapted to each device.

Responsive Web Design

The term *Responsive Web Design*, or RWD, was coined by Ethan Marcotte in 2010 in his article "A List Apart" (*http://www.alistapart.com/articles/responsive-web-design*), which was followed by his own short book *Responsive Web Design* (published by A Book Apart). It follows the progressive enhancement idea, as the same document will provide the best experience on every device using the same URL.

It's a simple and powerful idea to provide one HTML document that will automatically adapt (respond) on the client side to different scenarios, usually meaning available screen size or current orientation (landscape versus portrait). RWD is implemented using CSS3 media queries (covered in Chapter 11), which allow the same HTML to automatically change the layout and design in different conditions, so it can be used to separate between:

- Feature phones
- Smartphones
- Tablets
- Desktop browsers
- Smart TVs

A different CSS stylesheet (or a portion of one) will be executed in every scenario, so the layout and full design can be different on each type of device. In Figures 5-3 and 5-4, we can see two examples of the same website serving Responsive Web Design solutions.

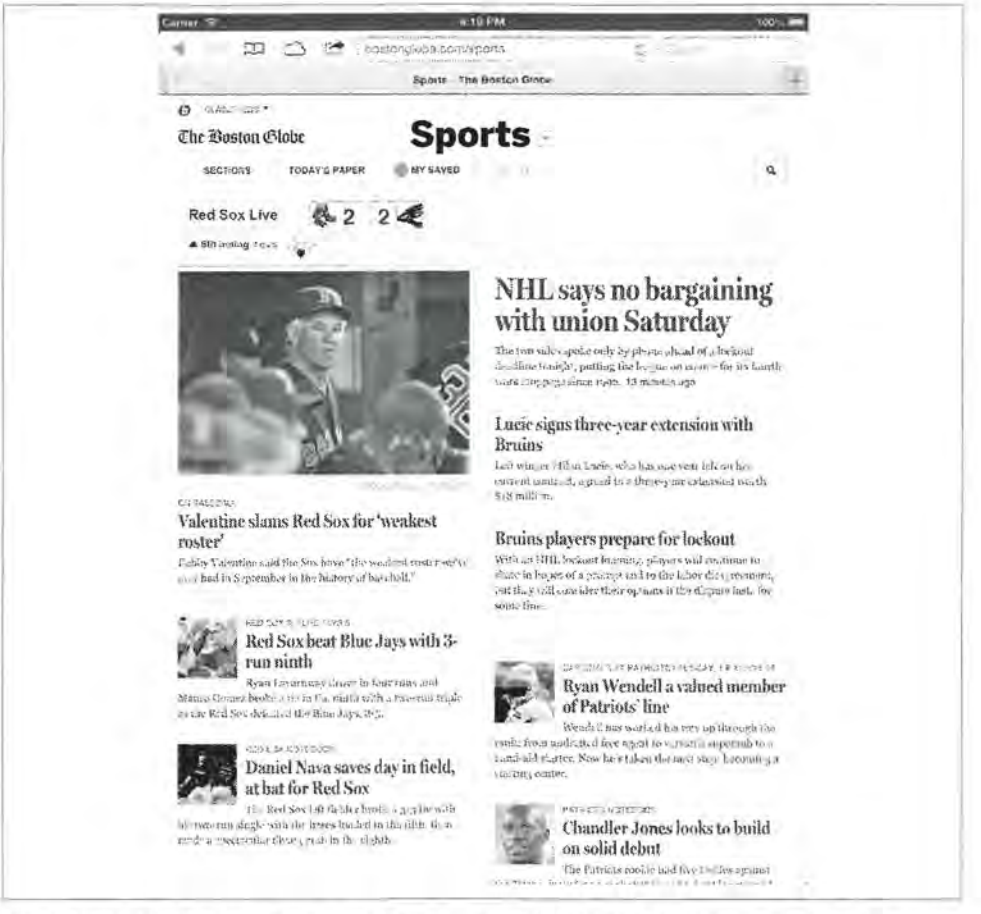


Figure 5-3. The Boston Globe uses Responsive Web Design to deliver the best design for each device and orientation—on a portrait iPad, it has a two-column design and a collapsed menu.

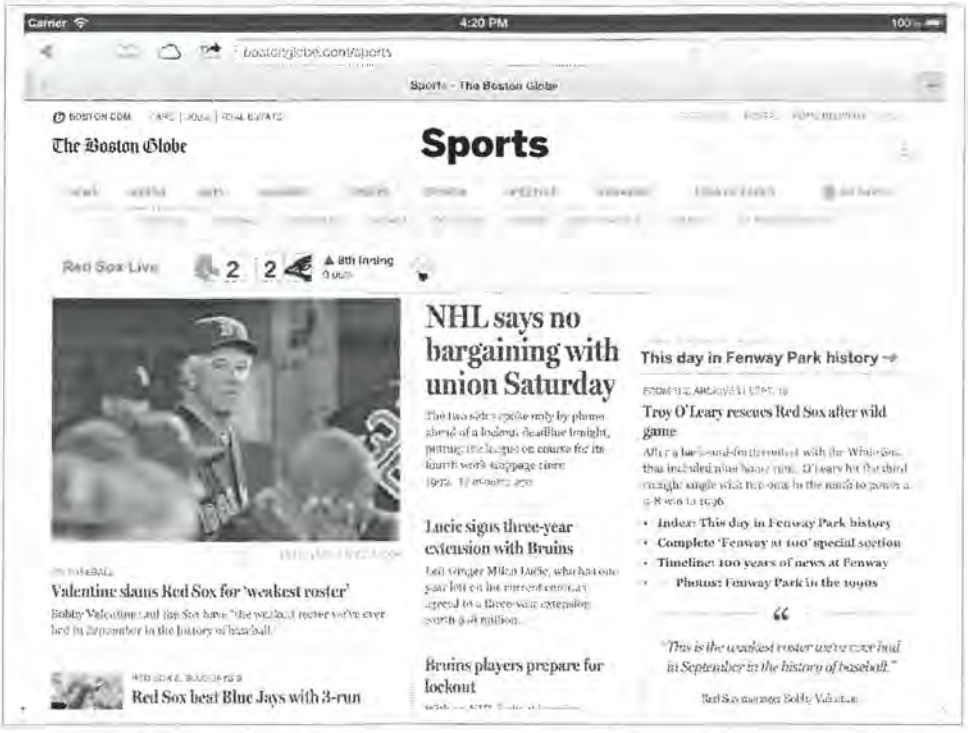
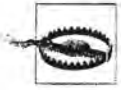


Figure 5-4. When you change the orientation of an iPad to landscape, the Boston Globe website automatically changes to a three-column design with a visible top menu.



While Responsive Web Design can be extremely useful, we need to be careful with this approach in some situations. Not every website is an ideal candidate for a full RWD experience at this time, sharing the same HTML document for desktops, tablets, smartphones, feature phones, and smart TVs. Performance is the key issue—when providing the same document across device types, we can fail in creating a high-performance experience.

While in a desktop browser we can resize a window, that is not possible on mobile devices—or even on smart TVs—so the magic of RWD is only useful for different orientations (portrait or landscape) or some special behaviors, such as the full-screen versus in-browser experience.



In RWD, the HTML file served is exactly the same, while the CSS may vary based on the device specifics (for example, screen width, screen height, pixel density, or orientation). Unfortunately, there is no way yet to recognize touch versus nontouch devices or other mobile-specific features using CSS3 standards.

I have to be honest with you: I like and promote the use of RWD techniques in mobile or tablet web versions as part of the progressive enhancement idea. However, using RWD to serve the same HTML document for all devices, including desktops, may not be a good idea for most websites. Problems include:

- The desktop HTML may include old browser hacks that are not necessary in mobile browsers.
- Unless images are inserted with a responsive technique—to be covered later in this book—the images served may be larger than the visible pixels on the screen.
- Desktop websites may contain large JavaScript frameworks, such as jQuery and its plug-ins, that, as we will see later, may not be a good idea on mobile websites.
- Ads and other third-party content usually are not responsive to the current context or device, so we may find problems if we use these solutions.
- In a content-based website, such as a newspaper, usually the desktop website will contain more text on the home page and inner pages, so we will be delivering content (consuming bandwidth and creating delays) just to hide that content from the CSS side.
- Mobile websites and mobile apps can only respond to orientation changes and, on some platforms, full-screen versus in-browser experiences. Mobile users will not receive changes in window dimensions; therefore, we determine the user experience and the design on the server side or at the `load` event instead of having RWD mechanisms waiting for a scenario change.
- Mobile apps sometimes take advantage of nondesktop features that can't be discriminated using CSS and RWD, such as geolocation, the accelerometer, and call-to-action links.

Therefore, unless you have a simple website in terms of content provided and external framework used, you should think of having at least two versions: desktop/classic and mobile. You can still use RWD inside the desktop version and inside the mobile version to make adaptations to different contexts. This may change in the following years if mobile networks and browsers become more powerful, but today performance is a key problem in mobile web browsing and mobile web apps.



If you want to browse hundreds of Responsive Web Design examples on the Web, you can check out *mediaqueri.es*. You can start accessing websites using a desktop browser and resize the window until you see that layout changes are being applied.

Responsive layouts

RWD is usually accompanied by a fluid grid layout allowing a site to adapt easily to different screen sizes. On the desktop side, sometimes it's common to have one or more fixed-width layouts (such as for 800px wide, 1024px wide, or 1280px wide).



Later we will cover best practices and free frameworks that will help us in creating Responsive Web Design websites and apps.

As you can see in Figure 5-5, Starbucks Coffee's US website has a fixed-width layout for large screen sizes, moving to a fluid-width layout for medium and small screens.

Responsive images

While we can change dimensions dynamically using CSS3, a problem appears with regard to images and how to make them react to different screen sizes. *Responsive images*, also known as *flexible images* or *fluid images*, make use of a group of techniques that may involve CSS resizing (using the same original file), supplying different image sizes on the server side, or cropping the images using CSS to adapt to different screen sizes and orientations.

We will cover responsive image techniques in Chapter 10.



Figure 5-5. Starbucks Coffee's US website uses a responsive layout, having a fixed width for large screens (white bars on both sides) and a fluid-width layout for smaller screens.

RESS

Responsive Web Design + Server Side Components (RESS) incorporates the best of two worlds: RWD and server-side adaptation. With this technique, we can decide on the best version and/or compatible experience on the server side, and then RWD on the client side will make the final adjustments, such as having a fluid layout and reacting to different orientations.

While the concept has been around for a while—it was even published in the fist edition of this book as a possible implementation of progressive enhancement—the RESS name was first suggested by Luke Wroblewski (*http://www.lukew.com*) in 2011 and involves the idea of having one version of the HTML on the server and changing some portions on the server side before delivering it to the client.



RESS solves some of the problems of Responsive Web Design, such as ensuring high-performance HTML for mobile devices and serving mobile-specific code, responsive images, and compatible ads.

For example, we can have a template that will change the header and the footer for feature phones, smartphones, tablets, and desktops, while the main section of the HTML remains the same. The same URL will deliver the current device version, which RWD will then adapt as the context changes.

Mobile First

"Mobile First" refers to the idea of creating a new web-based project first on mobile devices and then adapting it to bigger screens. Luke Wroblewski, author of the book *Mobile First* (A Book Apart), created this idea, too. Some big companies, including Google, Adobe, and Facebook, are already applying this practice in their own products.

Mobile devices' ubiquity, their ability to force us to focus on the product because of the small screen, the explosion in mobile web browsing figures, and the ability to read the user's context on a mobile device (such as through geolocation) are some of the reasons that Luke uses to promote Mobile First.

The idea behind this approach is that if you start with a large desktop website and then try to "mobilize" it (or worse, minimize it), you will end up with a not-so-optimized desktop website and probably a terrible mobile app.

If you start with your mobile website first, however, you will be forced to think about the best user interface, and the use cases; you will focus on the main business areas, and your mobile website will be better as a result. From that version you can then start on your larger-screen versions, which will end up being more optimized and usable.

Navigation

When creating your mobile web concept, before you do any coding you should define what will be in the navigation tree for the user. To do that, you need to understand what services and information will be available for the mobile user. Always remember the *80/20 law*: 80% of your desktop site will not be useful to mobile users. Therefore, you need to research the 20% you should be focusing on.



You can decide that you won't have a mobile website and just want to allow access to your desktop site to full HTML smartphones. If you're sure you will have mobile users I don't recommend you leave the desktop website as-is, but if you do decide not to create a separate mobile site, you will see later in this book how to optimize your desktop website for better visualization in smartphones.

Here are some tips you will need to follow:

- Define the use cases (for example, find a product price, find a store near you, call us, or perform a search).
- Order the use cases by the most frequent for a mobile user. Use your best guess, statistical information, and usability tests to keep this order updated.
- If you have a desktop website, try to maintain visual consistency with it.
- Do your best to make every use case successful in no more than three clicks or at a page depth of no more than three.
- Define no more than five main sections below the home page. If you need more, you should separate your service into more mobile pages.
- Always offer a link to the desktop website, sometimes called the "classic version."
- Determine whether locating the user is useful for your services.
- Reduce the form pages for user text input to the minimum. While mobile users are used to typing a lot—just look at SMS and the adoption of messaging apps!—typing in a long form usually results in a bad experience, as there is not much available space.
- Avoid startup or welcome screens in browser-based apps.
- Do your best to predict users' input based on the context and their browsing history, to reduce the number of page selections and clicks required.
- In full-screen web apps or native web apps, always provide navigation mechanisms such as back buttons. Remember that in those cases there will be no browser toolbar or UI.

Design and User Experience

Designing a mobile website or mobile app can be a challenge at the beginning. Unless you are working on a website for only one device, forget about creating an exact pixel-by-pixel web design. Your mobile website will look different on every device it's viewed on; you need to accept this and, keeping it in mind, develop a strategy to create the best web design you can.



The best advice I can give you about mobile web design is: Keep It Simple! However, that doesn't mean Keep It Ugly.

A mobile website ideally consists of vertically scrollable documents. The typical two- or three-column design is not suitable for mobile web pages, unless you are targeting tablets or devices with a landscape orientation mode. Every mobile web document has a few identified zones:

- Header
- Main navigation
- Content modules
- Second-level navigation
- Footer

These sections will be created one after the other in a vertical scope. Only for devices with a landscape orientation and smartphones is it suitable to create an alternative organization, where you can move the main navigation section to a right-side column.



On high-end smartphones, your main navigation can become a top or bottom tab bar, and the content modules can shrink with an accordion or master-detail design.

When you are creating a mobile version of an existing desktop website, you need to understand that you are *mobilizing* the website, not *minimizing* it. Minimizing (or miniaturizing) a desktop website simply involves displaying the same content on a smaller screen. Mobilizing is more than that; it requires understanding the context and offering your services and content in a manner that is useful and allows for quick access by the user.



If you are designing a mobile app or a website using Ajax, you should always insert in the UI a background operation icon to alert the user when a background connection is in progress. An offline button could be useful if the user is not on WiFi or is in roaming mode and doesn't want to get updates for a while.

Some best practices include:

- Avoid horizontal scrolling (unless you know what you are doing).
- Use Responsive Web Design to provide the best possible design in different scenarios, such as orientation changes.
- Maintain visual consistency with your desktop site, if you have one.
- Reduce the amount of text.
- Use fonts that will be legible on every screen; don't rely on the resolution.
- Use background colors to separate sections.
- Keep the main navigation to four or five links.
- Maintain the total link count at no more than 15 per page.
- For low-end and mid-range devices, don't insert more than one link per line.
- Use all the available width (not columns) for links, list elements, text inputs, and all possible focusable elements. Use HTML by-default focusable elements for clickable areas (for example, avoid using `divs`; use an anchor with a `div`).
- Provide a "Go to Top" link in the footer.
- Provide a Back button in the footer (some browsers don't have a Back button visible all the time).



If your navigation requires going back frequently, you should check whether the browser maintains the scroll position after going back. If not, you should probably create back links with anchors to scroll to directly where the user was.

- Provide the most-used features at the top.
- Group large lists by categories, each with no more than 15 items (for example, country selection by selecting the continent first).
- Use standard dialog windows.
- Use standard drop-down menus, such as HTML `selects`. Don't customize them or create them using JavaScript.
- Minimize the amount of user text input required if you have multiple fields.
- Save the user's history and settings for future predictive usage.
- Split large text articles into pages for feature phones (with page size depending on the richness of the browser).

- Try your color palette in different environments. Users may be in a place with poor lighting, on public transport, at the beach in bright sunlight, or in an office with fluorescent lighting.
- Provide different styling for touch devices.
- Think about fluid (liquid) designs for best adaptation.
- Use lists rather than tables.
- Embrace accessibility support. For example, use HTML5 ARIA or be careful when disabling zooming.
- Don't discriminate against your users; provide the best possible experience to every mobile device.
- For touch and cursor-based devices, use full-width links so that a link will activate if the user clicks on any pixel in the line containing it. Make sure there is only one link in each line.
- Use high-quality color images and fancier features for smartphones (we will discuss optimizing later).
- For cursor navigation, create medium-sized clickable zones for the cursor, moving by 5 or 10 pixels every time. Do not make the user travel a lot using the cursor; design all the clickable buttons near each other.
- Provide images that are sized on the server side for the current device's screen size and pixel density.
- Don't rely on fixed-position elements or on scrollable small areas (such as using `overflow: scroll`). Test them and provide alternatives if they are not compatible on some browsers.
- If you are providing a shortcut, a native app, or an offline version of your mobile website, create an alert at the top of the design alerting the user to download it. Don't show that alert after the first few views, or after the user has entered the download area. We will cover these techniques later.



Keep the text on your site to a minimum. Read every paragraph five times, and you will always find some word you can remove or a shorter way to say the same thing.

For low-end and mid-range devices, it is preferable to use a table design instead of floating `div`s, like in the first years of the desktop web. But keep in mind that using more than one item, link, or idea per line isn't a good practice on those devices.

I Didn't Want to Buy It!

Many years ago, I was browsing my operator's deck portal checking the games and content available for my device. I remember accessing the details of one game and seeing a screen that offered to let me buy it and have the charge added to my phone bill. I didn't want to buy the game.

The page asked me for permission to make the charge and offered two links, YES and NO. Both links were in the same line, one after the other. The focus was in the YES link (of course). What did I do? Instinctively, I pressed the right key to shift focus to the NO link.

What happened? The right key was a "go-to-link" action for my browser! I could only change the link focus using the up and down keys (like a tab), no matter where the links were located. So, because of a usability problem, I had to pay for content I didn't want.

That is why we need to test our websites on a wide range of real devices, and why I advise you to use only one link per line when targeting low-end and mid-range devices.

If you want to get deeper into mobile web design, you can read *Mobile Design and Development*, by Brian Fling (*http://www.flingmedia.com*) (O'Reilly). You can also find some useful native mobile user interface books on the market.

To get deeper information on usability research based on user testing, read *Mobile Usability*, by Raluca Budiu and Jakob Nielsen.

Mobile Web Best Practices (*http://mobilewebbestpractices.com*) is also a good source of links and information on mobile web design patterns.



Some low-end and mid-range devices have buggy CSS implementations, like the 100% width bug that generates a minimal (and annoying) horizontal scrolling action when this style is used on an element. You should test your design and change your strategy when problems like this crop up.

Touch Design Patterns

Touch devices have unique features in terms of design and usability. With the same amount of effort, the user can access every pixel on the screen; this encourages a different way of thinking about a design. Another difference is that the user will use her finger for touch selection (unless it is a stylus-based device). A finger is big compared to a mouse pointer, and the hit zone should reflect this.



The Touch Gesture Reference Guide is a great resource put together by Luke Wroblewski (*http://www.lukew.com*) that contains an overview of core gestures for most touch commands, tips on how to utilize gestures, visual representations of each one to use in documentation, and an outline of popular software platforms supporting them. You can download it from *www.mobilexweb.com/go/touchguide*.

Here are some useful design tips for touch devices:

- When the user touches the screen, parts of it will be obscured. Think about what will be hidden, and if it is important. Consider both right- and left-handed users.
- Provide a reasonable amount of space (20 pixels or more) between clickable elements.
- For frequently used buttons and links, provide a big clickable area (minimum 40 pixels width and height).
- For less frequently used buttons, you can use a smaller area (minimum 25 pixels).
- Provide quick feedback when a touch is accepted.
- Think about how scrolling will work.
- When using form input fields, try to insert the label above and hints below, not to the right or left of the input field. Generally, touch devices with virtual keyboards zoom in on the field when the user moves the focus to it, so the user will not see what is at the right or left of the input field while typing.
- Use finger gestures on compatible devices.
- Use infinite lists instead of pagination. An infinite list, like that shown in Figure 5-6, has a "More" final item that the user can click to dynamically add more elements to the list (via Ajax or other techniques). For performance purposes, you should reduce the number of pages shown at the same time. When adding new pages, the best way to approach this is to eliminate the first page from the DOM and create a "Previous" first item when the count reaches *n* pages (for example, 5). Doing this ensures that you will not have more than *n* pages shown at the same time.



Figure 5-6. An infinite list offers an option to load more items in the same list using Ajax.

- Use an auto-clear textbox feature for the most common text inputs. This is just an X icon at the right of the text input box that the user can click to automatically clear the input.
- Use the long-press design pattern (also known as "touch and hold") for contextual actions. This means that if the user presses a zone for two seconds or more, you can show a pop-up menu with contextual options.
- Prefer bottom-fixed to top-fixed tab navigation. The bottom of the screen (or the right, in landscape mode) is nearer the finger while browsing than the top zone.
- Analyze native touch applications for usability ideas.
- Hide the URL bar so you can have more space available for your app.
- Don't create touch gestures on the borders of your page (top, bottom, left, or right), as some mobile browsers will capture those gestures for their own actions.
- When using touch gestures, make them obvious (following operating system patterns), such as swipe right or left for picture gallery navigation; if they're not obvious, provide alternative buttons and/or an example tutorial at the first load.

Tablet Patterns

Designing tablet a website leads us to a big question: should we base it on the desktop version or on the mobile version? The answer may differ, depending on factors such as tablet size (7" or 10") and how your content is structured.



If you want to browse a large selection of mobile design patterns, you can check out Mobile Patterns (*http://www.mobile-patterns.com*) and Pttrns (*http://pttrns.com*).

If you are not providing a specific layout for tablets, starting from the desktop version is frequently the preferred way, as the screen size allows the display of more content. However, remember that tablets may be using cellular networks or not-so-fast WLAN networks, such as a public WiFi connection.

Providing smartphone-optimized experiences for landscape tablets doesn't seem that good an idea, as too much space is being wasted (as you can see in Figure 5-7).



Figure 5-7. When creating tablet experiences, starting from the smartphone version usually leads to waste of space.

If you are starting from a desktop website, two suggestions are important:

- Replace all the Flash content with alternative versions, as most tablets either don't support Flash or may remove its support in the near future.
- Provide more padding and bigger areas for clickable elements—Google's home page for desktops and tablets, shown in Figure 5-8, provides an example.

Official UI Guidelines

Official user interface guidelines from the manufacturers, links to which you can find at www.mobilexweb.com/go/uiguides, are another source of inspiration for mobile web design. Here, you will find guidelines, samples, tips, and descriptions of common mistakes. Many of the guidelines focus on native application development, but we can apply most parts of them to mobile web design, too.



If you apply the long-press (touch and hold) design pattern, you should be aware that browsers have their own long-press actions for clickable elements, like links, images, or text, for copying and pasting. In WebKit-based browsers, you can disable text selection on text items using the `webkit-user-select:none` style and then create your own menu.

The most important guides are:

- iOS Human Interface Guidelines
- UI Guidelines for BlackBerry Smartphones
- Android User Interface Guidelines
- Nokia Design & User Experience Guidelines
- Windows Touch UI Guidelines

What Not to Do

The website WTF Mobile Web (*http://wtfmobileweb.com*) collects dozens of examples of what not to do on mobile websites. There is a quote on this site that I would like to mention here:

> Respect your users' time and context. Don't penalize people for how they choose to access your sites, products and services. Work towards universal access to content.

While there's a long list of websites with big usability problems on this site, I've made a quick selection in Figure 5-9 whose problems you should be able to spot pretty easily.

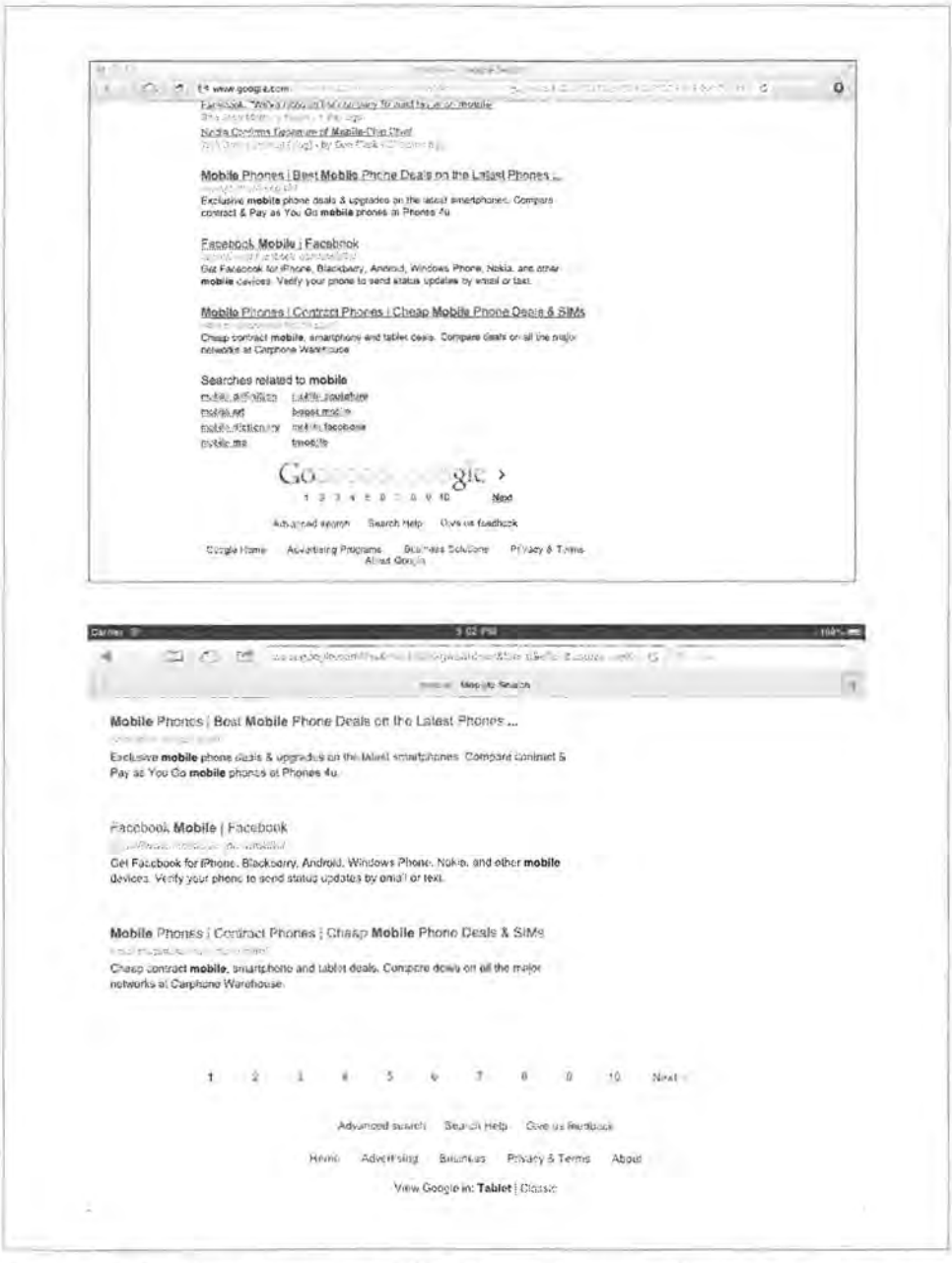


Figure 5-8. The Google home page offers the desktop version for tablets, but with some small enhancements, such as larger clickable areas.

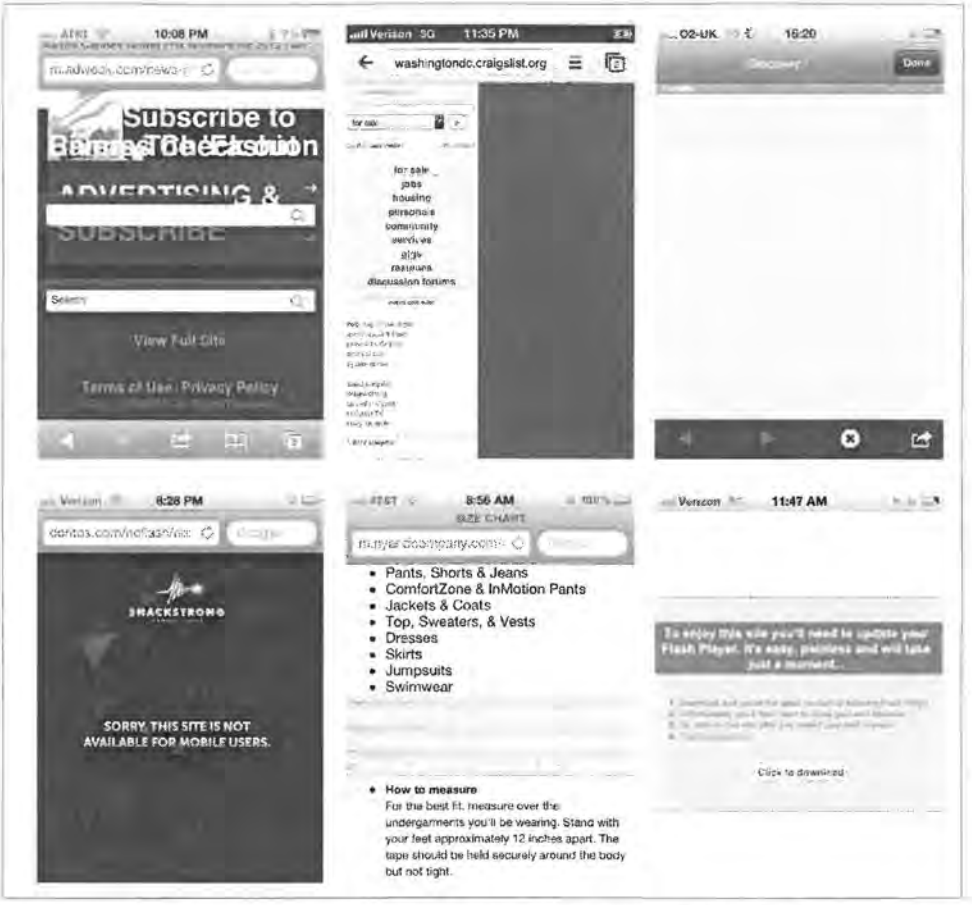


Figure 5-9. WTF Mobile Web offers a list of what you need to avoid when creating mobile websites.



If you don't know yet what WTF means, well…you can search for its definition on the Web.

The list of problems can be summarized as follows:

- Too-small text whose size cannot be increased.
- "Unsupported browser" messages (remember, it's the Web!). These are worst when they suggest that the user download IE or some other desktop browser.
- Big fixed elements that create unusable experiences.
- "Install Flash Player" messages.

- "Mobile version available soon" messages, without even a link or a way to open the desktop version.
- Full-Flash websites that show only an error icon on the screen.
- Too-narrow small links for a touch device.
- Very large paragraph elements that can't fit on the screen, so the user needs to scroll right and left to read them.
- Forcing the user to install a native app instead of providing a website (even if a website exists).
- Providing a 320px-wide fixed view on a large tablet.
- Providing full-screen installation invitation dialogs when the user is not in the browser (such as when the user is in a web view or inside a pseudo-browser).

CHAPTER 6

Markups and Standards

Finally, we have arrived at the best part: coding! If you are reading this chapter but have skipped the previous ones, I encourage you to read them. Mobile web development is not just about coding; it is important to understand the full ecosystem, including what types of devices and browsers are available, and to be aware of mobile design and usability issues. That said, let's take a look at the available markup languages and the relevant standards.

First, the Very Old Ones

Although you're unlikely to use them in mobile web projects today, some familiarity with at least the basic concepts of the old markup languages can be useful. One day you may need to migrate an old mobile website, or work with older devices, and I wouldn't be satisfied if I didn't talk a little about them.

One of the first mobile web markup languages to be developed was *HDML* (Handheld Device Markup Language). Similar to HTML, it was developed by a company called Unwired Planet (the company that became Openwave and was later taken over by the Myriad Group). This markup language was never released as a standard, but it helped in the creation of WML (Wireless Markup Language).

Why Not HTML from the Beginning?

The first specialized language for the mobile web, HDML, appeared in 1996. Why not use the well-known HTML from the beginning? There were a few main issues. For one thing, mobile devices were so limited in terms of network access and CPU and memory resources that it was necessary to create very small rendering solutions. A mobile browser couldn't process nonstrict markup and decide what to do if the developer forgot to close a tag, for example. The other issue was the need to create mobile-specific functionality in the markup, like keyboard shortcuts.

Over time, mobile devices evolved into what we know today; now some mobile devices are even more mature than desktop ones, and mobile devices began using HTML5 before desktops, well before the standard (still in draft format) was finished.

WML

WML was incorporated into the WAP 1.1 standard and was the first standard of the mobile web. It was standardized not by the World Wide Web Consortium (W3C), but rather by the WAP Forum (known today as the Open Mobile Alliance), an organization made up of many players from the mobile industry working on standards in this market.

We have already agreed that WML is absolutely deprecated today. Any nonsmartphone will still understand WML, but I want you to consider it a historic language, like Latin, instead of a current standard. Depending on your target, you may still want to create a basic WML version like that in Figure 6-1, but it is not the place to start.



Figure 6-1. A typical WML document used to contain just text, links, and maybe some small image; it was always focus-based and optionally could execute WMLScript code, but that was very rarely used.

In fact, some modern browsers (Safari on iOS, Android browser, IE) do not read this format anymore, as shown in Figure 6-2. It was the markup for WAP 1.1, and the first

(and almost the last) version was created in 1998! Just think about what mobile phones were like that year. Still, if you search for "filetype:wml" in Google, you'll find nearly a billion results using this format. And Google did not index the majority of the WAP 1.1 mobile web!



Figure 6-2. Both Safari on iOS and the Android browser show the WML source code instead of rendering it.

A couple of years ago, I asked some big mobile portals about their WML usage. The Weather Channel (*http://m.weather.com*) was the first to give me a good answer (from Cathy Rohrl, Product Manager – Mobile Web):

> To have a WML-compliant site is not that big of an issue. It was easy to build and it's just out there. But the importance of having WML today is supporting the concept of access to EVERYONE, everywhere. You start tempting people with older handsets, and they'll want more. Another year and we may completely mothball the site, but even then I don't think we'll take it down. It will just become a site that is not actively maintained.

Internal nonaudited private reports of US traffic on The Weather Channel's mobile site indicate that 5% of traffic in 2008 was WML-only, decreasing to 2% in 2009 and even less in early 2010.

A WML file is an XML file, normally using the *.wml* extension. It is similar to HTML in some ways and very different in others. Let's take a look at a typical WML file:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
   "http://www.wapforum.org/DTD/wml_1.1.xml" >
<wml>
   <card id="home" title="Welcome to Old Mobile">
      <p mode="wrap">This is a <b>typical</b> paragraph in WML</p>
      <p mode="wrap">It can include images,
         <a href="http://wap.yahoo.com">External Links</a> and
         <a href="#two">Internal Links</a>.
      </p>
   </card>
   <card id="two" title="Second screen">
      <p>This is like a second page in the same document</p>
   </card>
</wml>
```

We can recognize many tags found in HTML here, like `p`, `b`, and `a`, and they have the same functionality. Other tags the two standards have in common include `img`, `br`, and `input`.



Today it is common to use the self-closed tag `<br />` in XHTML files instead of the classic `<br>` without a closing tag. WML, as one of the first XML-based markup languages, was the pioneer in using the self-closed tag.

However, you'll also notice some differences. First, a WML file starts with a root `wml` tag after the DOCTYPE declaration. A WML document is also called a *deck*. Every deck can contain many cards. A *card*, identified by a tag with the same name, is one visible page in a browser; it is like the contents of a `body` tag in HTML. So yes, a WML file can contain many pages in the same document. This was a great feature for speeding up the performance of the mobile web in the early 2000s.



The Document Type Declaration (known as the DOCTYPE) is an instruction in an XML document (or an SGML document, such as HTML) that allows the browser to match that document with a Document Type Definition (DTD) so it can tell how to understand the document.

WML was conceived for mobile devices. Consequently, we will find tags and attributes supporting mobile device functionality (such as voice calls, keyboard support, adding contacts to the phone book, and accessing the SIM card) in the standard. The best part is that we can use the well-known anchor tag to create an absolute link, a link to a relative document, or a link to another card in the same document using the *#card_name* URL.



The multiple cards design pattern in WML is very useful. We will use it in our modern mobile websites using JavaScript, the Document Object Model (DOM), and even Ajax. You'll need to wait a few pages for that, though!

There is a lot to say about WML; in fact, I have a book on WML on my bookshelf that's more than 600 pages long. But to be honest, WAP 1.1 pages today are so simple that this quick introduction should be enough for you to understand WML.



WML is not compatible with CSS, and its minimal design support includes the use of the tags `big`, `small`, `b`, and `i` using a "best effort" mechanism. Many old WML browsers had only one font and no bold or italic support.

If you are still curious about WML, you can use Adobe Dreamweaver to create WML files with code hinting support. When you select File→New, you will find WML in the "Other" section. Of course, you can also use any text editor and a WML-compatible emulator. The best WML emulators today are the Nokia Series 40 emulators (only for Windows), as they show real rendering and work well on modern desktops running Windows.

WML Was Not Alone

WML does not generally support GIF, JPEG, or PNG images (although some browsers did accept GIF and JPEG images, starting with color screens). Images in WML files were typically in *WBMP* (Wireless Bitmap) format. WML also supports scripting using a language called *WMLScript*, loosely based on ECMAScript. They aren't worth discussing; just know that they existed and talk to your grandchildren about them. A WBMP file is just a 1-bit-per-pixel bitmap file, in black and white.

Other common scenarios involved compiled WML and WMLScript files. These files were compiled by the developer or by a proxy or WAP gateway between the user and the web server. A free tool for compiling WMLScript files is the old Nokia Mobile Internet Toolkit, still available for download.

At the same time that WML appeared on the market, in 1998, compact HTML (cHTML) also appeared, mainly in the Japanese market. cHTML is a subset of HTML with additions for mobile features, like support for access key shortcuts, pictorial characters (emoticons, or Emoji), and Japanese characters. It was submitted as a standard to the W3C but its adoption was mainly in Japan, with some implementations in the Netherlands, Italy, France, Australia, and the United States. Early versions of cHTML lacked support for JPEG, tables, backgrounds, frames, and stylesheets.

Current Standards

In terms of the mobile web today, our real work will be directly related to the following standards and pseudo-standards:

- HTML5 and other sub-standards
- XHTML Mobile Profile 1.0, 1.1, and 1.2
- XHTML Basic 1.0 and 1.1
- HTML 4.01
- De facto standard mobile HTML extensions
- WAP CSS
- CSS Mobile Profile (CSS MP)
- CSS 2.1
- CSS 3.0 and other sub-standards, such as CSS3 transitions and CSS3 columns

This may seem overwhelming, but don't panic: it isn't really that complicated. We can distinguish two main types of standards: HTML-based and CSS-based.

Politics of the Mobile Web

Why are there so many standards? The first answer is politics. Politics? Yes. Many actors are involved in the mobile web, and everyone wants to be part of the decision-making process. Were mobile web standards "mobile enough" to be managed by mobile standards organizations, like the Open Mobile Alliance (OMA)? Were they "web enough" to be managed by web standards organizations, like the W3C? Do the manufacturers have enough power to decide on their own markup?

Those kinds of questions are responsible for the nightmares that can occur with markup. Here's another look at the preceding list, grouped by owners:

W3C mobile web & web standards:

- XHTML Basic 1.0 and 1.1
- CSS Mobile Profile (MP)
- HTML5 and subsequent standards
- CSS3 and subsequent standards

Web Hypertext Application Technology Working Group (WHATWG) web standards:

- HTML Living Standard

OMA mobile web standards:

- XHTML Mobile Profile (MP) 1.0, 1.1, and 1.2
- Wireless CSS (WCSS) or WAP CSS

Manufacturers' extensions to the standards:

- De facto standard HTML extensions
- CSS custom extensions

Managing multiple standards

The first bit of good news is that, with the exception of some new features in HTML5 and CSS3, all the standards are similar and compatible with each other. The second piece of good news is that HTML-based browsers—including every mobile browser on the market today—have a "good effort" mechanism to manage nonrecognized tags and attributes. This is at the heart of HTML's evolution.



On today's mobile web, we will use HTML5 and CSS3 for smartphones and tablets and XHTML MP with WAP CSS for feature phones and some other devices.

I remember my first desktop HTML project in 1996, and the projects after that. The list of compatible tags was different for each browser on the market (at that time, Mosaic, Netscape Navigator, the AOL Browser, and, a bit later, Internet Explorer). However, it wasn't a big problem. If the browser did not understand a tag or an attribute, it just ignored it. The same is true of most mobile browsers. This will be very helpful in enabling us to manage all standards at the same time (with some exceptions: basically, older devices).

As the mobile device manufacturers were nearer to OMA than the W3C, they officially implemented the WAP 2.0 standard using XHTML MP and WAP CSS. However, almost all browsers also understand XHTML Basic and CSS MP, and most mid-range and high-end devices understand full desktop web standards (HTML and CSS).



Don't rely only on the standards. Even two devices supporting the same standard may render different results for many tags, attributes, and styles. We will analyze every usage, and I'll recommend the best solution.

Delivering Markup

Before we talk about the individual standards and the differences between them, we will analyze how to deliver each standard to a mobile device. First, as in the desktop web, all static document markups use the *.html* extension, and the style ones use the *.css* extension. Of course, we can deliver HTML5, XHTML MP, or XHTML Basic using a dynamic template, a *.php* or *.aspx* file, or servlet Java.

So, how does the device know which standard we coded a website in? By reading the MIME type and the DOCTYPE. The MIME type is a string sent by the server telling the browser the format of the document, and the DOCTYPE is the first line in the HTML file. If you omit the DOCTYPE the file should still work in many browsers, but don't do this! The other thing to notice is that in HTML 3.2, 4.0, and optionally HTML5 the opening tag should be:

```
<html>
```

while for all the other XHTML subtypes it should be:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

All CSS standards use the same MIME type as in the desktop web (`text/css`), and there is no format indicator inside the CSS. That is why for the style file, we will not need to define which standard we are using. The selectors and attributes used will determine compatibility. Figure 6-3 illustrates how MIME types and DOCTYPEs travel through the network, one in the header and the other inside the document.

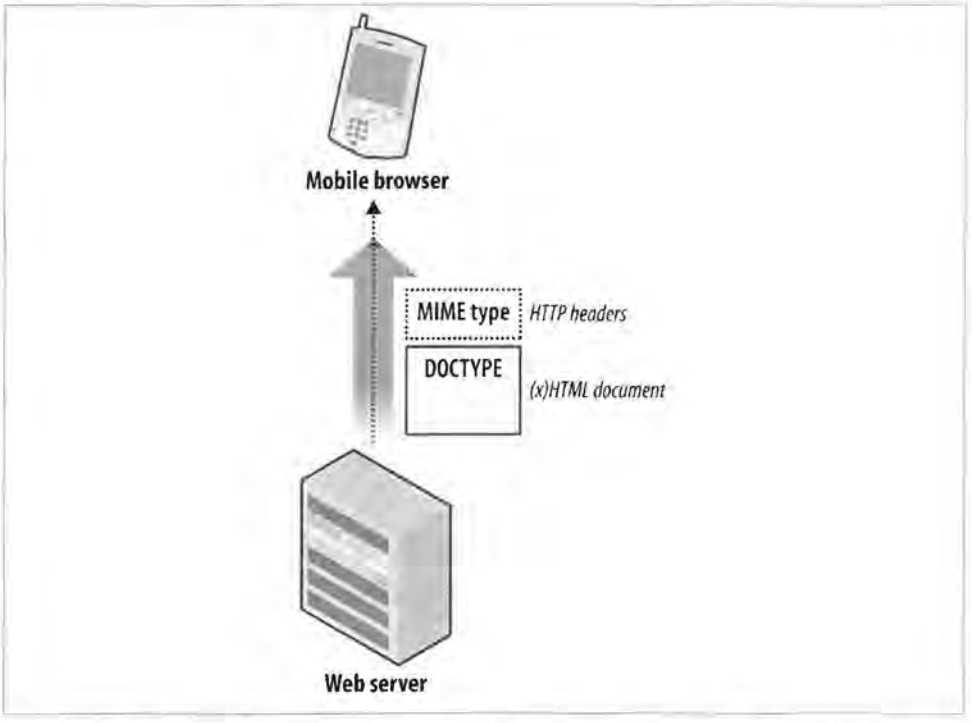


Figure 6-3. The MIME type travels with the server's response headers and the DOCTYPE is defined inside the HTML document.

The preferred MIME types and DOCTYPEs are listed in Table 6-1.

Table 6-1. MIME types and DOCTYPEs for today's standards

Standard	Preferred MIME type	DOCTYPE
XHTML MP 1.0 (first version)	`application/xhtml+xml`	`<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN" "http://www.wapforum.org/DTD/xhtml-mobile10.dtd">`
XHTML Basic 1.1	`application/xhtml+xml`	`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN" "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">`
XHTML MP 1.2 (last version)	`application/vnd.wap.xhtml+xml`	`<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.2//EN" "http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd">`

Standard	Preferred MIME type	DOCTYPE
XHTML 1.0	application/xhtml+xml	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
HTML 4.0	text/html	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
HTML5	text/html	<!DOCTYPE html>

Safari on iOS will render a file differently if the markup is using the HTML5 or the XHTML Mobile Profile DOCTYPE, as shown in Figure 6-4. The biggest difference will be the viewport used. This will be covered in Chapter 7, but for now, it's good to know that a file in XHTML MP markup will not start zoomed out, like a normal HTML file.



Figure 6-4. The same document, with the same MIME type, rendered in Safari on iOS: the version on the left is using the XHTML Mobile Profile DOCTYPE and the version on the right a nonmobile XHTML one.

If we are delivering XHTML (any version) we should also include the `meta` tag to tell the browser the content type of the file, using the right MIME type or `text/html` (even if we are using XHTML or XHTML MP, as the W3C recommends), and define the charset used (UTF-8 in almost all situations):

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

In HTML5, this `meta` tag becomes:

```
<meta charset="UTF-8">
```

This is optional, but it may be useful if we don't define the charset used in the HTTP header and we don't use the XML header. Using the header alternative is the preferred and most compatible way to do it.



If we use basic markup compatible with all standards, some old and basic devices will not understand the `text/html` MIME type, and some smartphones will not understand the XHTML MP MIME type. In a later chapter we will learn how to change this attribute dynamically, even when we deliver the same code.

XHTML MP can be delivered using the OMA MIME type (`application/vnd.wap.xhtml+xml`), the XHTML type (`application/xhtml+xml`), or even the HTML type (`text/html`), with the same result. The OMA recommends using the first one, but using the XHTML type will work well in almost all situations. Some older and low-end devices render the page differently depending on the MIME type used.

This Book's Testing Suite

All the testing documents used in this book are available for free at www.mobilexweb.com/tests; you can test every feature tested here yourself with any mobile browser. For less typing, you can use the Mobile Tiny URL, typing *t.ad.ag* in your browser's address bar (you will need to type *8123124* on almost every numeric keypad). This is a valid URL on the Internet, and it uses only the first characters associated with each numeric key to reduce keypresses. You can create your own URLs for easy mobile typing by accessing www.mobiletinyurl.com from your desktop browser.

Every suite was tested on the latest versions of some of the platforms available at the time of this writing, on all the major versions available on the market, and on older versions of the same platforms. As mobile browsers are evolving quickly, new versions could have different results. You can follow my blog (*http://www.mobilexweb.com*) or my Twitter account (*http://www.twitter.com/mobilexweb*) for updates.

Charset encoding

For the best compatibility for Latin languages, we should deliver any XHTML with `UTF-8` defined in the XML header or in the `Content-Type` HTTP header. If we are delivering just HTML or content in other languages, we can use other encodings.

Conclusion about MIME types and DOCTYPEs

What is the advantage of using the mobile headers? On some devices (those using Safari on iOS, for example) the behavior may differ if you use or don't use the mobile headers, and for some devices using them is the semantically correct solution. This will be our flag saying, "Hey, this is a mobile website, and it is not intended to be used from a desktop." This metadata is very helpful for search engine robots to determine which pages are mobile-ready.

In the mobile HTML5 era, on smartphones and tablets, the way to determine if a page is mobile-ready is through other meta tags that we will see later in this book, such as the viewport meta tag.

XHTML Mobile Profile and Basic

XHTML MP is based on the W3C's XHTML Basic, and they are almost the same. The W3C has an online mobile validator (*http://validator.w3.org/mobile*); it accepts XHTML Basic and MP as valid markup.

XHTML Mobile Profile is a subset of XHTML. It is XML-based, so we need to follow the strict rules. If you have never worked with XHTML 1.0 or 1.1, let's analyze the differences compared with working with HTML5:

- The file must have a root element (`html` tag).
- Every tag name and tag attribute must be in lowercase.
- Every attribute must have a value, and it must be enclosed in quotes. For example, `<option selected>` is invalid; you must use `<option selected="selected">`.
- Every tag must be closed. This may seem obvious, but it is not; tags like `<img>`, `<input>`, and `<br>` don't need to be closed in HTML, but they do need to be closed in XHTML. The general rule is to use self-closed tags, like `<br />`.
- The tags need to be closed in reverse order. If you open a paragraph and then a link, you must close the link before closing the paragraph.
- XHTML entities must be well formed. A mandatory space should be ` ` and an ampersand character should be `&`.
- The DOCTYPE declaration is mandatory, and the XML opening tag is optional. In fact, for mobile browsers we should not insert the XML opening tag.



This is not a book about XHTML, HTML5, CSS, or even JavaScript. I assume you have some basic experience with these markup and programming languages; if not, you will find a lot of resources on the Web and excellent books from O'Reilly Media to help you get started.

Available Tags

We have finally arrived at the level of code. XHTML MP, as a subset of XHTML derived from HTML, will look familiar to most web developers.

The Space Before the Final Closing Slash

You may be familiar with the use of `<br />` in recent years, whether in XHTML or in HTML. Do you know why the space is included before the closing slash? In an XML file, we can use `<br/>` without a space, and it's valid. The space is for backward compatibility with non-XHTML browsers that don't expect a final slash in the tag. Using the space ensures that most older browsers will understand the tag as a line break.

The tags available in both XHTML Mobile Profile 1.2 and XHTML Basic 1.1 (the two standards are almost at the same level) are listed in Table 6-2. Some features, like scripting support, were added in XHTML MP 1.1 and others, like object support, in the last standard (1.2).

Table 6-2. HTML tags available in XHTML MP 1.2 and Basic 1.1

Tag types	Tags available
Structure	`body`, `head`, `html`, `title`
Text	`abbr`, `acronym`, `address`, `blockquote`, `br`, `cite`, `code`, `dfn`, `div`, `em`, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`, `kbd`, `p`, `pre`, `q`, `samp`, `span`, `strong`, `var`
Links	`a`
Presentation	`b`, `big`, `hr`, `i`, `small`
Stylesheet	`style`
Lists	`dl`, `dt`, `dd`, `ol`, `ul`, `li`
Forms	`form`, `input`, `label`, `select`, `option`, `textarea`, `fieldset`, `optgroup`
Basic tables	`caption`, `table`, `td`, `th`, `tr`
Other	`img`, `object`, `param`, `meta`, `link`, `base`, `script`, `noscript`

If we compare previous versions of XHTML MP and Basic, the differences are bigger. The last XHTML Basic standard (1.1) added almost every addition in XHTML MP 1.2, and now the two are almost equivalent.



XHTML Mobile Profile 1.2 is the last standard from the OMA. The first draft was presented in 2004 and the approved version was released in 2008. Remember that it takes some time for browser developers to comply with new standards, and more time for manufacturers to get devices using the new standards to the market.

We can still use a tag that is not supported in our declared DOCTYPE. It will not validate against the DTD, but most mobile browsers that don't support it will simply ignore the tag without any error visible to the user.

Official Noncompatible Features

Every mobile browser on the market today should understand and render the tags listed in Table 6-2. However, in XHTML MP (and Basic), there are also several tags, techniques, and technologies that are officially not supported. We will still test them in every browser, though, because as we've seen there are many full HTML browsers on the market, and others that will understand some noncompatible features. All of the following are officially unsupported:

- Nested tables (tables inside other tables)
- Full table tags: `thead`, `tbody`, `rowspan`, and `colspan` attributes
- Full form tags: `<input type="image">`, `<input type="file">`
- Editing tags: `ins`, `del`
- Image maps
- Frames
- Iframes
- Deprecated formatting tags: `font`, `dir`, `menu`, `strike`, `u`, and `center`

We will check all browsers for compatibility with those features, as well as the following:

- Adobe Flash
- The `XMLHttpRequest` object (Ajax)
- SVG
- The `canvas` HTML5 tags
- Other embedded objects: Windows Media, QuickTime, Java applets
- Multimedia HTML5 tags: `audio` and `video`
- Opening links in new tabs or windows

We will also verify which URL schemas are available for each browser.

Creating Our First Compatible Template

Let's create a very simple markup template that will be compatible with all devices. I really recommend that you use the source code view if you are using a visual web tool, like Adobe Dreamweaver or Microsoft Expression Web. You should feel comfortable with nonintrusive, semantic HTML code for mobile web development.

Our template will look like this:

```
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.2//EN"
    "http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>First Template</title>
</head>

<body>
    <h1>First Template</h1>
    <h2>Programming the Mobile Web</h2>
    <p>Welcome to the first template of this book</p>
    <p>It <strong>should work</strong> in every mobile browser
       on the market</p>
    <ol>
        <li><a href="http://m.yahoo.com" accesskey="1">Yahoo!</a></li>
        <li><a href="http://m.google.com" accesskey="2">Google</a></li>
        <li><a href="http://m.bing.com" accesskey="3">Bing</a></li>
    </ol>
    <p><img src="images/copyright.gif" width="150" height="50"
          alt="(C) mobilexweb.com" /></p>
</body>
</html>
```

Here are some comments on this code; the results are shown in Figure 6-5:

- We are using the XHTML MP DOCTYPE.
- We are using standard header tags for titles; that is, `h1` ... `h6`, not `p` or `div` tags.
- We are using the paragraph tag (`p`) to enclose text.
- We are using an ordered list to show a link menu. The option numbers match the `accesskey` attributes of the anchor (`a`) tags.
- We provide a `width`, `height`, and alternate text for all images.

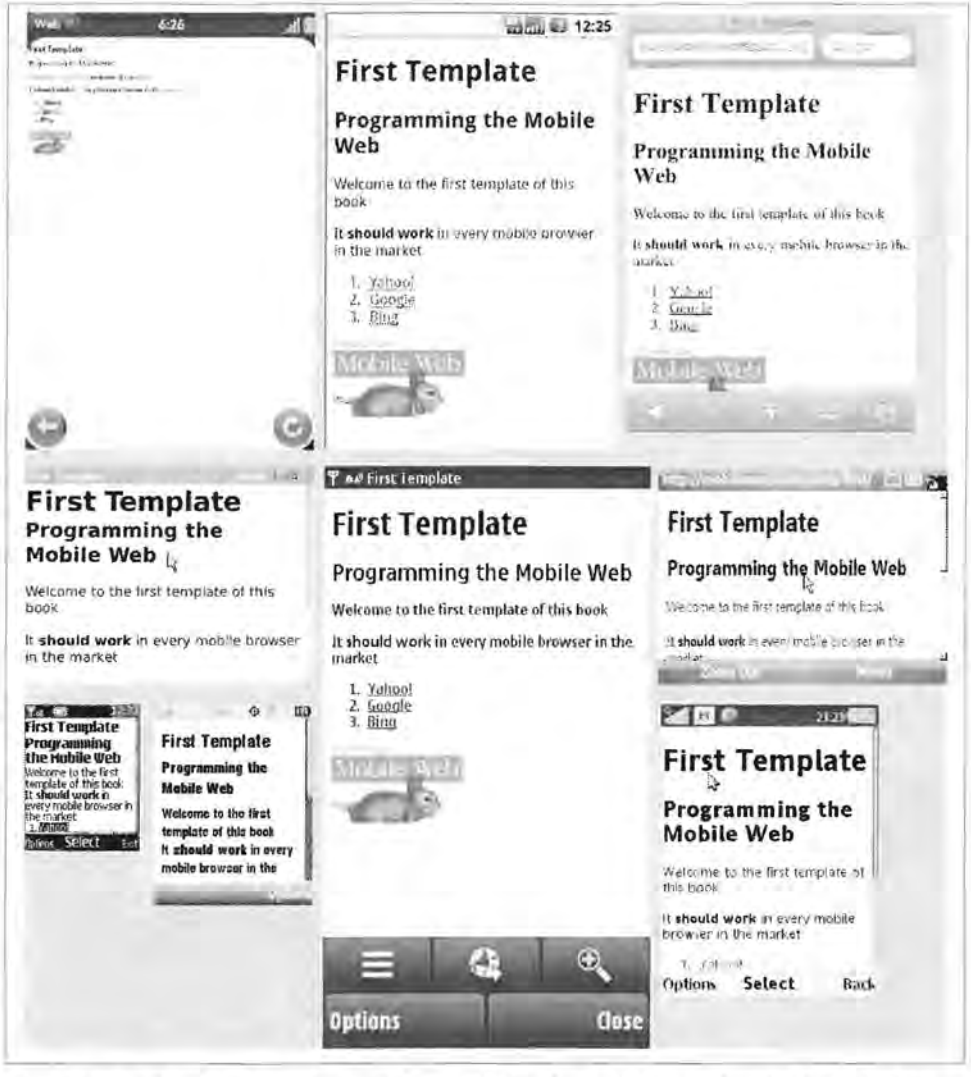


Figure 6-5. The same template without CSS in the webOS, Android, iOS, BlackBerry 5.x, Symbian, Windows Mobile, Nokia S40 (low-end device), Motorola, and NetFront browsers.

As these images prove, this code works on every platform. I know what you're thinking: "Hey, this is an awful experience for the iPhone." If you're tempted to throw away this book right now, wait! Give me a chance. Using this exact code, we will create a great iPhone (or other smartphone) experience in the following pages. Well, OK, there may be some little changes to the code, but not so many. Our goal will be to keep our document template as simple as this one, even for very complex HTML5 web apps.

Markup Additions

WML, as a mobile-specific language, has many mobile-prepared tags and attributes. This is not true of XHTML MP or Basic, and that's why many developers were against WAP 2.0 in the early 2000s. Some vendors, such as Nokia, even tried to create their own markup supersets over XHTML with mobile-specific features.

The only mobile-specific addition is present in XHTML Basic 1.1, but even with the XHTML MP DOCTYPE it will work if the browser understands that markup. The addition is the attribute `inputmode`, available for the `input` and `textarea` tags; it allows us to specify input mode tokens indicating the expected type of the input characters (`latin`, `thai`, `arabic`) and modifier tokens for the text input (such as `predictOff` to deactivate predictive input). XHTML MP adds similar support with WAP CSS, which we will cover later.



WML had a `format` attribute for the `input` tag that was similar to `inputmode`. Some browsers, such as Openwave, still understand this attribute in an XHTML file.

Mobile HTML5

Let's be clear: there is nothing officially called "Mobile HTML5" in terms of standards. We are talking about the same HTML5 as we can use on other platforms. However, there are some specific behaviors, hacks, and even tags that we will want to use when creating HTML5 documents for the mobile web.



While many developers still do not have a clear idea of exactly what HTML5 is, the W3C has created its own description, and it has reached a point where it will not add any more features to it; the W3C has also begun working on the next version, called HTML 5.1 (*http://www.w3.org/TR/html51*), which is in a very early draft state at the time of this writing. While versioning HTML5 is still a subject of much discussion in the community, the truth is that today we don't need to define a version number; we can just use features when they are compatible with the current browser.

The history of HTML5 deserves more space than I can dedicate here. To keep it brief, originally the W3C was against it and was promoting XHTML 2.0. Some W3C participants disagreed, and they split off and created a new organization called WHATWG (*http://whatwg.org*), whose purpose was discussing and creating the next generation of markup languages for the Web.

Originally called Web Applications 1.0 before HTML5, the new standard began as a couple of tags and JavaScript APIs that modern browsers should implement. Apple was one of the first companies to apply these ideas (in Safari for iPhone), trying to fill the gap created by the lack of Flash support. At that time, the term HTML5 was not yet in use.



The W3C created the HTML5 logo to promote all technologies around the Web. You can find out more about the HTML5 logo from the W3C (*http://w3.org/html/logo*).

After a couple of years, the W3C finally gave up on XHTML 2.0 and created a new project called HTML5 based on the WHATWG discussion. It then started the normal discussion—and, let me say, necessary large steps—to arrive at a standard language. The standard is still in draft and in discussion today, but HTML5 has become a buzzword with a broad meaning. That is why WHATWG decided to change its own name for the new standard to "HTML Living Standard," indicating something that is evolving every day and doesn't have snapshots or versions. That living standard idea is more representative of what HTML5 really is today, from a community perspective: a standard in constant evolution. From the W3C's perspective, however, HTML5 is a spec that will have a limit (to be finished in 2014), and then we will be talking about other versions, such as HTML 5.1, for adding more features.

Creating Our First HTML5 Template

HTML5 does not follow XML rules, so it's less strict in terms of syntax. The minimum HTML5 template looks like this:

```
<!DOCTYPE html>
<title>My title</title>
```

If you are wondering where the `html`, `body`, and `head` are, they are all optional tags in HTML5. Even if you open them, you don't need to close them, so the following code is also valid HTML5:

```
<!DOCTYPE html>
<title>My title</title>
<html>
<head>
<body>
```

Usually, a mobile HTML5 template also includes some important meta tags; we'll cover these in the next chapter.



Many of the new features in HTML5 have a fallback mechanism, meaning that, with the same code, we can provide alternative solutions with noncompatible browsers. In some cases, we can use JavaScript libraries that will also provide the same solution, known as *polyfills*. However, in the mobile web space we need to be careful about adding large JavaScript frameworks.

Syntax Rules

Being a non-XML language, HTML5 has its own rules. The most important rules for understanding the rest of the code in this book are:

- The `html`, `body`, and `head` tags are optional.
- There are elements that must always be closed, such as `ul`, `script`, `select`, or `div`.
- There are elements that you can close, self-close, or not close at all, such as `p`, `li`, `option`, `input`, `head`, `body`, `img`, `br`, or `link`. For example, `<br></br>`, `<br />`, and `<br>` are equivalent.
- Element and attribute names are not case-sensitive.
- Boolean attributes exist. In those cases, you don't need to provide a value. The mere presence of the attribute (as in `<input required>`) indicates a true value. If you really need to define a value, then the same name as the attribute means true while anything else means false; for example, `<input required="required">` means true.
- Attribute values require quotation marks (double or single) only when the value includes spaces or other special characters. Therefore, in most situations you don't need quotation marks, such as `<input type=text>`.



Even though HTML5 doesn't follow XML rules, you can still stick to those rules, and it will be fine. In multiplatform mobile web development where you can support older browsers, you should stay with XML syntax so you can then reuse templates with HTML code with XHTML MP documents.

You can find a lot of resources to learn HTML5, including excellent books from O'Reilly Media and other publishers, so I won't take too much time explaining all the differences here; we'll just look at what's necessary to create our multiplatform mobile web solutions.

New Elements

We are going to talk about the usage of new elements (elements not present in XHTML 1.0) in the following chapters. In the meantime, here's a quick list of new elements that we have in HTML5:

- Semantic block elements: `header`, `footer`, `article`, `section`, `nav`, `aside`
- Semantic inline elements: `time`, `output`, `progress`, `mark`
- Text elements: `wbr`, `ruby`
- Multimedia elements: `video`, `audio`, `canvas`



As we'll see in Chapter 10, we can inline SVG static or animated content using an `svg` element, and it will work on mobile smartphones.

CSS for Mobile

Web (and mobile) browsers have a great feature that makes our lives much easier in the CSS world. If we use any selector or attribute that the browser doesn't understand, the browser will just ignore it. This will be very helpful in the following pages. Usage of CSS 2.1, CSS 3.0, CSS Mobile Profile, and WAP CSS is the same; we specify CSS selectors and attributes for those selectors. The standards only tell us which selectors and attributes are supported, and we will find browsers that do not properly render standard ones and do properly render noncompatible tags.

If you are interested in having W3C-valid markup, remember that XHTML Basic 1.0 doesn't support CSS, and 1.1 added support, but only for a `style` or `link` tag with external styles. The W3C standard doesn't support the inner styles defined in the `style` attribute.

WCSS, or WAP CSS (the OMA standard that comes with XHTML MP), is a CSS 2.0 subset, like CSS MP (the W3C standard that comes with XHTML Basic). That's why we will focus here on CSS 2.0 features (and beyond). We'll begin by talking generally about "mobile CSS," and later we will see how the different mobile browsers handle each of those features.

WCSS Extensions

The Open Mobile Alliance standard added to CSS2 some new attributes that we can use in mobile browsers. As this is how CSS defines extensions, every new attribute has a dash (-) as a prefix. We will see later that some mobile browsers also understand some nonstandard extensions, like WebKit-based browsers. But again, don't worry; we will

look at compatibility attribute by attribute so you understand how to manage incompatibilities the best you can.

Access keys

The first attribute is `-wap-accesskey`; it is the counterpart of the XHTML `accesskey` attribute. It can be used with any interactive element (generally, the `a`, `textarea`, `label`, `input`, and `button` tags). The possible values are the digits 0 through 9 and the special values * and #. For some browsers on devices with numerical keypads, this attribute can be used to create shortcuts to access those elements. Some browsers do nothing with it, some browsers set the focus on that element when the user presses the key, and other browsers directly fire the action associated with it (go to a link, submit a form).



We should use only the standard keys 0–9, *, and # as access keys. We cannot assign functions to soft keys or any other special keys in HTML. WML allows us to assign links to soft keys, but this functionality has no effect in modern browsers.

We can assign the same key to only one element in the same page. That's why the `-wap-accesskey` attribute is useful only with ID selectors or with inline styles. You shouldn't use this attribute with element or class selectors.

The next three samples all have the same result:

```
<a href="http://mobilexweb.com" accesskey="0">Our website</a>
<input type="submit" value="Send" accesskey="9" />

<a href="http://mobilexweb.com" style="-wap-accesskey: 0">Our website
   </a>
<input type="submit"" value="Send" style="-wap-accesskey: 9" />

<style type="text/css">
#linkWeb {
   -wap-accesskey: 0;
}
#btnSubmit {
   -wap-accesskey: 9;
}
</style>

<a href="http://mobilexweb.com" id="linkWeb">Our website</a>
<input type="submit"" value="Send" id="btnSubmit" />
```



Access keys work with only a few browsers today. This is because many of them use the keypad for accelerators (shortcuts for browser functions like scrolling, going back, or reloading). That is why we can use them only if they are not the preferred or only way to access functionality on the website.

Marquees

If you've been doing web development for a long time, like me, you probably hate the nonstandard `marquee` element that many people used to insert in web pages. WAP CSS revived this technique to create small animations without images that do not require Flash. A marquee is generally a text that scrolls from one side of the screen to the other, wrapping around continuously. In some mobile browsers it can contain any HTML code, including images and even tables. However, don't scroll too much heavy markup, for the sake of your visitors and the performance of your website.

To create floating, scrolling text, use any paragraph element, like `p` or `div`, define the `display` attribute as `-wap-marquee`, and assign values to some of the CSS attributes listed in Table 6-3.

Table 6-3. Marquee WAP CSS attributes

Attribute	Possible values	Description
`-wap-marquee-dir`	`ltr` or `rtl`	Direction of the scrolling. Can be left to right (`ltr`) or right to left (`rtl`).
`-wap-marquee-loop`	Any number or infinite	Animation count. The `infinite` value creates a never-ending animation.
`-wap-marquee-speed`	`slow`, `normal`, or `fast`	Speed of the animation, without fine control.
`-wap-marquee-style`	`scroll`, `slide`, or `alternate`	Possible styles for the animation.

The following sample shows how to use a marquee to present an offer to the user:

```
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN"
    "http://www.wapforum.org/DTD/xhtml-mobile10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Mobile Web Test</title>
<style type="text/css">
.offer {
    display: -wap-marquee;
    -wap-marquee-dir: rtl;
    -wap-marquee-speed: medium;
    -wap-marquee-loop: infinite;
    -wap-marquee-style: scroll;
}
```

```
.offer strong {
    color: red;
}
</style>
</head>

<body>
<div class="offer"><strong>Fly to the Moon</strong> Special offers this
month starting at US$145.00. Apply now and see us from the sky.</div>
<h1>TravelWithUs.com</h1>
(...)
</body>
</html>
```

The result is shown in Figure 6-6.



Figure 6-6. A Nokia Asha showing a marquee animation in progress using standard WAP CSS.

Now let's take a look at how browsers react to this tag. We may be tempted to use this display type to show a large amount of text in a small space, but if the mobile browser doesn't understand the marquee display, all that text will appear on the page, pushing down the important content! That is why we should consider alternative solutions for noncompatible browsers, like hiding the content. For example:

```
.offer {
    display: none;
    display: -wap-marquee;
    -wap-marquee-dir: rtl;
    -wap-marquee-speed: medium;
    -wap-marquee-loop: infinite;
    -wap-marquee-style: scroll;
}
```

In the preceding code, we first assign `display: none` to remove the text from the display. Then we assign `display: -wap-marquee`. If the browser understands the WAP CSS marquee styles, it will replace the none value. If not, it will just ignore the second setting

and the none will win. We can also apply this style to a `marquee` HTML element, so it can work in all possible marquee-compatible browsers. The problem is that the code will not validate against standards (if we are interested in that).

Try to avoid using marquees for important information. You may want to use them to reduce the space taken up by information that is not directly relevant, or to have some kind of animation free of plug-ins. Avoid the usage of links, images, or any other nontext markup inside a marquee, and create an alternative CSS stylesheet for noncompatible devices.

CSS form extensions

Another great enhancement in WAP CSS is the ability to define useful information for form input. We will talk about this in depth later, but for now let's see what extensions are included in the standard. They are listed in Table 6-4.

Table 6-4. WAP CSS form extension attributes

Attribute	Possible values	Description
`-wap-input-format`	Complex pattern (see Chapter 11)	Defines the pattern of the text. Can be applied to text fields, password fields, and textareas.
`-wap-input-required`	`true` or `false`	If `true`, requires the user to provide some content before exiting the field. Can be applied to text fields, password fields, and textareas. This attribute has precedence over the `format` attribute in the `input` tag, if both are defined.

CSS3

CSS version 3.0 is now split into different subgroups of standards. Therefore, we have basic CSS3 support and a group of other sub-standards in the W3C, such as:

- CSS Background and Borders
- CSS 2D & 3D Transforms
- CSS Transitions
- CSS Animations
- CSS Columns
- CSS Flexible Box Layout
- CSS Fonts
- CSS Device Adaptation
- CSS Regions

HTML5 Compatibility Levels

HTML5 in mobile devices is usually thought of as a mix of W3C standards, drafts, and de facto practices, including HTML, CSS, and JavaScript APIs. The W3C maintains the *Mobile Web Initiative* (*http://w3.org/mobile*), where you can find some useful links, discussions, and best practices.



In the following chapters, we will analyze all the HTML5 JavaScript APIs that we can use on mobile web platforms, including resources to understand current compatibility.

W3C discussion groups have reached some conclusions, including the following resources:

- Mobile Web Best Practices 1.0 (*http://www.w3.org/TR/mobile-bp/*), from July 2008
- Mobile Web Applications Best Practices 1.0 (*http://www.w3.org/TR/mwabp/*), from December 2010
- Standards for Web Applications on Mobile (*http://www.w3.org/Mobile/mobile-web-app-state/*), current state and roadmap, updated frequently

Testing Your Browser

One of the biggest problems with HTML5 is that every browser has different compatibility in terms of the standards supported. While there is no standard to define how "HTML5 compatible" a browser is, there are some initiatives in the community to solve this problem.

HTML5 Test

The most-used resource today to measure HTML5 support is *html5test.com*. This website will execute a test suite on your browser and give you a score based on the APIs/features supported, plus a bonus score for any optional features supported (such as video codecs). In my opinion, this website is primarily useful for comparing different versions of the same browser, as in Figure 6-7; if you compare two different browsers the temptation is to interpret a higher score as indicating better compatibility, but the problem is that this algorithm does not consider which APIs are most useful to you—maybe Browser A has a better score than Browser B, but only Browser B supports the API you are looking for.



Figure 6-7. Different Android browsers on the same device will have different scores on the HTML5 test.



HTML5 Test does not test all the mobile-specific APIs. Therefore, the score on your mobile phone will not take into consideration some of these standards.

Ringmark

Ringmark (*http://rng.io*) is an open source mobile browser testing suite that will give you not an exact score, but a compatibility level (a ring) regarding your current support. The test has three different rings covering several standards and features that a browser may support. If one feature on one ring fails, the suite will not execute the next ring's tests.



Later in this book, we will analyze all the APIs and features that Ringmark tests.

Ringmark will identify a browser as being Ring 0, Ring 1, or Ring 2 compatible. Therefore, if you are creating an app or a game, you can say that it is compatible only with Ring 1 browsers, for example, so you will know which APIs and features will work properly.



If you want to test your own mobile browser, just point it to *rng.io*.

At the time of this writing, no mainstream mobile browser has reached the Ring 1 mark. The first Ring 1 compatible browser was Dolphin for Android, released in August 2012, as seen in Figure 6-8. The prize for the first Ring 1 compatible default installed browser was claimed by BlackBerry with the new BlackBerry 10 platform at the beginning of 2013.

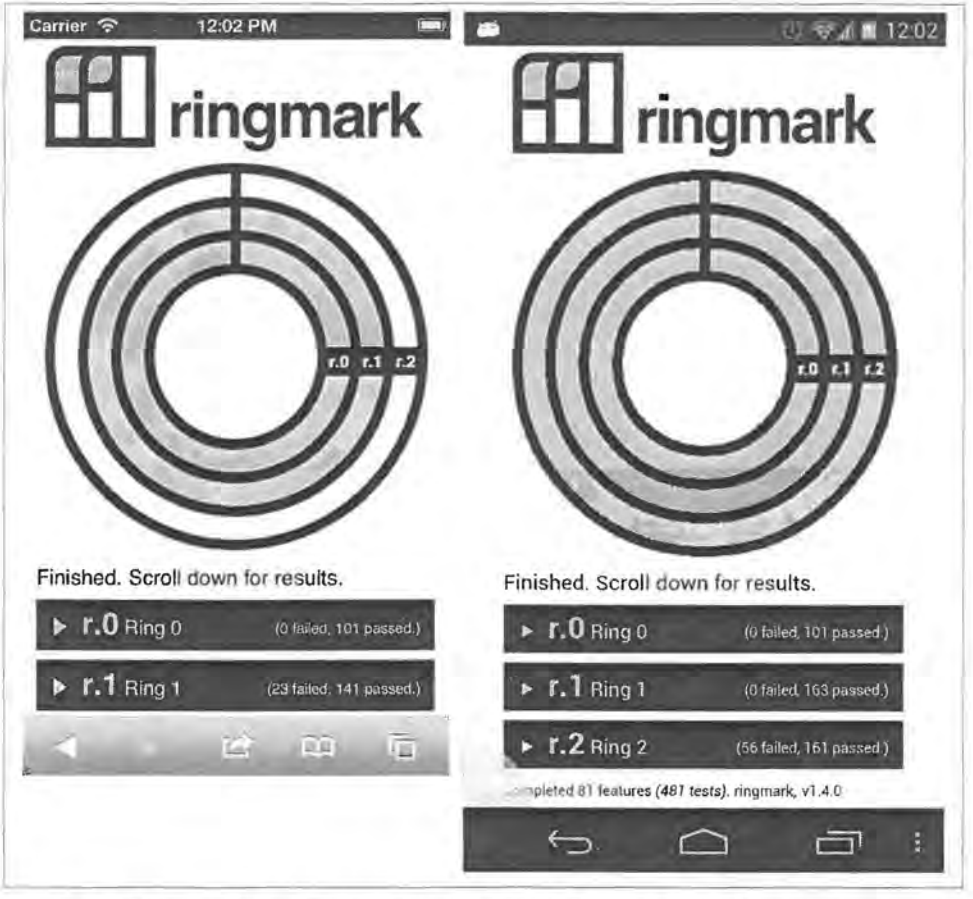


Figure 6-8. Ringmark is a mobile browser HTML5 testing suite; here we can see that Safari on iOS 6 reaches Ring 0, and Dolphin for Android reaches Ring 1.



It's OK to be a little confused after reading about all the incompatibilities. This is just an introduction to general and mobile-specific standards—just theoretical information. Actual concrete practice will make it much clearer!

CHAPTER 7

Basics of Mobile HTML5

The standards are sometimes utopias, while the real world is something different. Many devices officially support standards, but in practice some feature is missing; many other devices add support for more technologies besides what is covered by the standards.

They are decent resources, though, and they contain much good advice for multiplatform mobile web development. We will take that a bit further here, testing every feature in the standards (and some nonstandard ones) to draw real conclusions about their usage.

We will go through a typical document, from the heading to the body structure, looking at the most common design patterns for document structure, including forms, frames, tables, links, and images. We will test every possible solution for each topic on every mobile platform so we can get some useful information about what we can and cannot use.

The Document Head

The head part of a mobile web document will be very similar to that in a desktop web document, with the addition of some new meta tags useful only in mobile browsers.

Title

First we'll define a `title`, as for any other web page. The space available for the title in a mobile browser is small compared with a desktop browser (Table 7-1 gives the average lengths of the titles displayed on the different phone-factor platforms). The page title is used as the heading at the top of the screen on some devices; other devices also use the title as the default text for bookmarks and the history list.

Table 7-1. Average characters used in titles on smartphone platforms

Browser/platform	Average number of chars used in titles
Safari on iOS	40 chars in portrait and 60 in landscape (75 on 4.5" devices). Hidden after the user scrolls the page. Hidden in full-screen web apps.
Android browser	After 2.0, titles are not displayed on the screen.
Chrome for Android	Titles are not displayed on the screen. You can see the first 15 chars on the window list, or tabs on tablets.
Internet Explorer	No usage.
Symbian/S60	35 chars in portrait, 20 in landscape.
Nokia Series 40	20 chars in third edition. No usage in fifth and sixth editions.
Firefox	15 chars in portrait and 50 in landscape, replacing the URL when the page loads.
BlackBerry	15–30 chars, depending on screen width.
webOS	No usage up to webOS 1.3. In webOS 1.4, the title appears only if the user scrolls down from the top.
NetFront	No usage.
UC browser	10 chars in portrait and 30 in landscape, replacing the URL when the page loads.
Opera Mobile	Depends on the screen, between 20 and 60 chars.
Opera Mini	Depends on the screen, between 20 and 60 chars.

Every mobile title needs to be:

Meaningful
: Avoid duplicate titles for every page of your mobile site using only your company name. However, on your home or entry page, don't use "Home Page," use your company or product name and keep it very short. This may be the most bookmarked page.

Short
: Keep the title between four and eight words long. If mentioning the name of your company, do that last (for example, "Big Mac - Meals in McDonald's"). Use small words first; some old devices truncate the title after 10 or 12 characters.

Concise
: Don't waste words. For example, avoid using "Mobile" in the title; the user knows that she is using a mobile device.



When creating full-screen experiences or native web apps, usually the title seems useless. However, there are two reasons why you should always add a `title` element: 1) some APIs, such as dialogs and permission-based APIs, will use it as a dialog window title; and 2) the `title` element is mandatory if we want to validate against HTML5.

Website Icons

In the early 2000s, everyone rushed to insert *favicon.ico* files in their websites' root files to see how the icons would be added to Internet Explorer's address bar. Today, in the desktop web those icons are more useful for tab iconography. But what about in mobile browsers?

In HTML, the standard way to add an icon file is to use the following `link` tag:

```
<link rel="icon" href="favicon.png">
```

Originally, the icons were in Windows ICO format (similar to BMP), but these files are difficult to export from well-known graphic editors and are not optimized in size. Today, you can use PNG for mobile compatibility. Originally the icon size had to be 16×16 pixels, but now they can be any square size and the browser will resize them. Usually, the preferred size for this icon is 32×32 pixels.



X-Icon Editor (*http://www.xiconeditor.com*) is an online HTML5-based icon creator tool specially created for the Internet Explorer ICO format. When you download the editor, you can upload your PNG icon and it will be exported to ICO format. It is recommended that you start with a 64×64-pixel icon.

Internet Explorer 9 for Windows Phone 7.5 ignores the icon, but IE10 uses it for tablets and other devices running Windows 8. It ignores the `rel="icon"` definition and accepts `rel="shortcut icon"` instead; also, the icon must be in ICO format. Therefore, we should also include the definition:

```
<link rel="shortcut icon" href="favicon.ico">
```



If you are creating XHTML documents—for example, using XHTML Mobile Profile—remember that you need to close every element (as in `<link ... />`). In HTML5 self-closing is not necessary.

Browser support for icons varies. As you can see in Figure 7-1, some tablet browsers, such as Safari on the iPad, do not use the tab icon, while other browsers, such as Google Chrome IE10 for Windows 8, do.



Figure 7-1. Some tablet browsers use the icon for the tab, while others—such as Safari on the iPad—ignore it completely.

SVG is an excellent icon format because it is resolution-independent. However, while some desktop browsers support this format, it is not yet supported on mobile devices. PNG is the recommended solution for most browsers, other than Internet Explorer (IE supports the ICO format only, and it may include different-resolution icons in the same file).

In Figure 7-2 we can see different mobile browsers using the website icon.

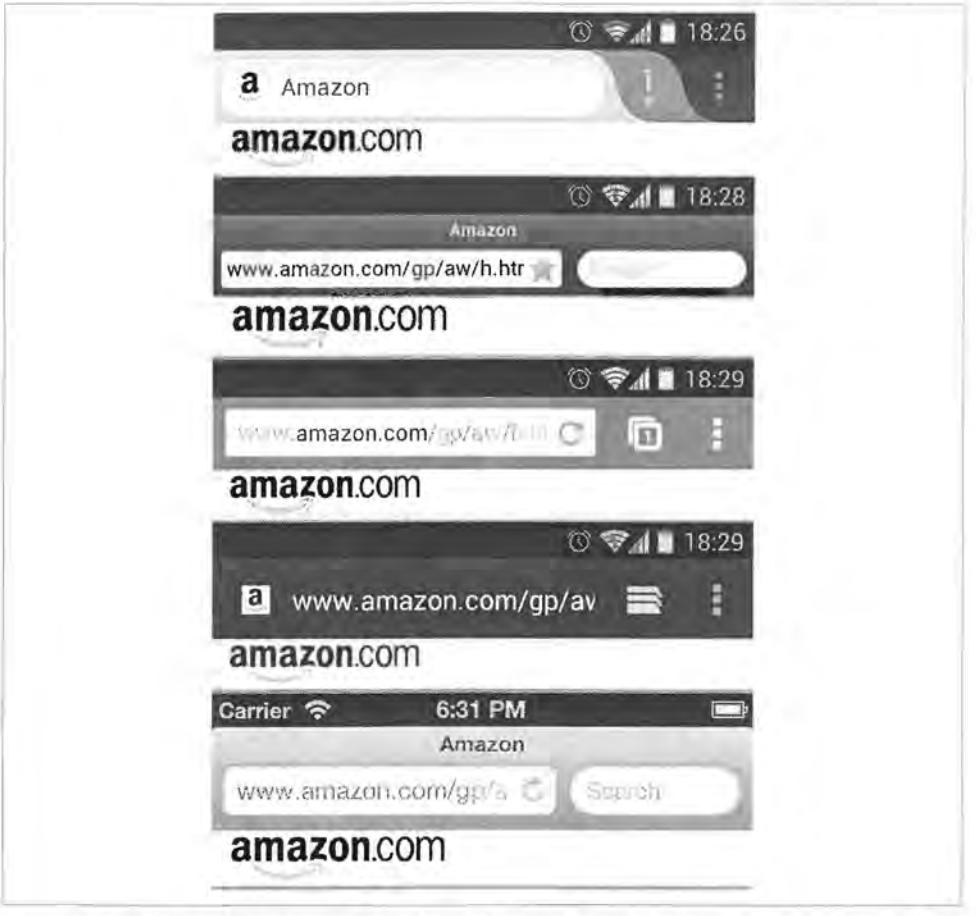


Figure 7-2. The website icon is rendered on some mobile browsers and ignored on others.

Some browsers support bookmarking, and others also support adding icons to the Home screen. Let's talk first about bookmarking. Some platforms don't associate any image with the bookmark, some platforms take a screenshot of your website (as seen in Figure 7-3) and use that as the icon, and some others use the website icon.



Figure 7-3. Some browsers, such as the Android browser on Android 4.1, will take a screenshot of the current page and add that to bookmarks instead of using the website icon.

In Table 7-2 we can see current browser compatibility with website icons.

Table 7-2. Icon display compatibility table

Browser/platform	Usage on toolbar	Usage on tab/ window list	Preferred format and size (px)
Safari on iOS	No	No	
Android browser	In the URL bar	No	PNG or ICO 32×32
Chrome for Android	No	Yes	PNG or ICO 32×32
Nokia Browser for Symbian	No	No	
Nokia Browser for S40	No	No	
BlackBerry < 7.1	In the title area (some devices)	No	PNG or ICO 16×16
BlackBerry PlayBook and BB10	Yes	Yes	PNG or ICO 16×16
NetFront	No	No	
Internet Explorer	No in IE9, yes in IE10 for Windows 8	No	ICO multiresolution 64×64 and 32×32
Opera Mobile	Some versions	No	PNG or ICO 32×32
Opera Mini	Some versions	No	PNG or ICO 32×32
Firefox for Android	Yes	Yes	PNG or ICO 32×32
webOS browser	No	No	

Home Screen Icons

Some platforms support a way to create shortcut icons for the device's Home screen or applications menu. For this purpose, they usually ignore the website icon because of its low resolution.

Apple Touch Icons

Safari on iOS was the first platform to support Home screen icons, and for that purpose Apple created another type of icon, known as a *Web Clip*. Safari on iOS requires a square PNG file (with no transparency preferred) using a `link` element with a `rel` value of `apple-touch-icon` or `apple-touch-icon-precomposed`, so it can coexist with the previous icon declarations:

```
<link rel="apple-touch-icon" href="iphone_icon.png">
```

If we use the `apple-touch-icon` version the icon will automatically be given rounded borders, a shadow, and a reflective shine, like the one shown in the middle of Figure 7-4. If you don't want your icon to have the reflective shine effect, instead use the following meta tag alternative:

```
<link rel="apple-touch-icon-precomposed" href="iphone_icon.png">
```



Figure 7-4. The original iOS icon file, the final appearance once the website has been added to the Home screen using the nonprecomposed version, and the icon using the precomposed link version.

Up to iOS 3.1 (known as iPhone OS at that time), 57×57 pixels was the only available size for the Web Clip. Starting with iOS 3.2, 72×72 appeared as the icon size for the original iPad. Before iOS 4, server-side techniques were necessary to deliver iPhone or iPad icons using the same `link` element.



The `rel` HTML attribute accepts multiple value declarations using spaces. However, Safari on iOS will ignore the whole `link` element if we try to provide more than one `rel` value at the same time, such as `rel="icon apple-touch-icon"`.

For iOS 4.0 and newer, Safari supports a `sizes` attribute that we can apply to different link definitions. The value defines the possible icon sizes that we can use on different iOS devices with different pixel densities. For example, devices with higher DPIs (such as the iPhone 5) need an icon size of 114×114 pixels, while high-resolution iPad devices (such as a third-generation iPad) require a size of 144×144 pixels.

If we don't provide different sizes, the device will resize the icon with quality loss.



You should always provide an icon with 90-degree corners for iOS. Safari will add rounded corners to your icon by default. If you provide an icon with an alpha transparent background, Safari will use a black background by default.

If we want to provide Home screen icons for every iOS version, the code for `apple-touch-icon` will look like this:

```
<-- iPad icons -->
<link rel="apple-touch-icon" href="icons/72.png" sizes="72x72">
<link rel="apple-touch-icon" href="icons/144.png" sizes="144x144">

<!-- iPhone and iPod touch icons -->
<link rel="apple-touch-icon" href="icons/57.png" sizes="57x57">
<link rel="apple-touch-icon" href="icons/114.png" sizes="114x114">
```



While you can provide only one big icon without a `sizes` definition and Safari will resize it, remember you are delivering a big file for older devices, which may cause performance issues. Also, the icon might not look good on lower-resolution devices.

If more than one icon appears to be valid for the current context, Safari will take the last one. Therefore, up to iOS 3.2 (which does not support the `sizes` attribute), the last declaration will be taken.

If we don't provide one version of the iOS sizes, Safari will use the next-lower version. For example, if we provide only 57-, 72-, and 114-pixel versions, the third-generation iPad (expecting 144 pixels) will use the 114-pixel version.

When Is the Web Clip Icon Retrieved?

The appropriate icon for the current device is not retrieved when the page loads. Instead, on iOS 6 and beyond, the icon is retrieved when the user presses the Share icon before selecting "Add to Home Screen," as seen in Figure 7-5. Prior to iOS 6, the icon is retrieved from the server when the user selects "Add to Home Screen." The Share icon, which looks like a box with an arrow jumping out of it, is located in the bottom bar on iPhone and iPod devices and the top bar on the iPad (see Figure 7-6).

With that in mind, there is a possibility that the page will have loaded entirely, but when the user tries to add it to the Home screen, the network will not be available anymore. In that case, the icon will be replaced by a screenshot. If you want to force the icon, you can pre-cache it using typical image caching techniques or using a data URI.

In Chapter 10, we will get further into the data URI protocol; for now you just need to know that it means that the icon will be embedded inside the HTML page and will not be retrieved by a different network request. The advantage is that the user will have the icon immediately and will not get the strange effect of seeing a screenshot of the website being replaced by the icon after a while. The main disadvantage is that the icon is always inside the HTML and there is no cache opportunity.



Figure 7-5. This is the process on iOS 6 for adding a page to the Home screen—once installed, it looks like any other native app on the operating system.



Figure 7-6. On the iPad the add to Home screen process is similar to the iPhone, but the Share icon inside Safari is in the top bar as opposed to the bottom bar on the iPhone and iPod touch.

If you don't define the `apple-touch-icon` or `apple-touch-icon-precomposed` link element, Safari will look for the existence of the file anyway on the server when the user tries to add the page to the Home screen. From iOS 4.0, the search algorithm will look for the files in the following order:

- */apple-touch-icon-<size>x<size>-precomposed.png*
- */apple-touch-icon-<size>x<size>.png*
- */apple-touch-icon-precomposed.png*
- */apple-touch-icon.png*

Therefore, if an iPhone 5 is accessing your website and you don't provide an Apple-specific Home screen icon, the browser will try */apple-touch-icon-114x114-precomposed.png* first, and then continue with the other three options. If there is no file available on the server, Safari will take a screenshot of your website and use that as the icon.



If you don't provide a Home screen icon though a `link` element, Safari will make four requests to your server anyway that will generate 404 error codes.

Icon titles. The title that appears below the icon, as seen in the last sequence in Figure 7-5, is automatically taken from the `title` element of your HTML document. The text will be cropped if it has more than 13 characters; therefore, if you have a long title, you need to be careful and make sure the first 13 characters are the title you want on the Home screen. If necessary, you can pad it with spaces to control the display.

In iOS 6 and beyond, a new meta tag appears to solve this problem, allowing you to provide an alternative title for the Home screen. We will cover iOS web app meta tags later in this book, but for now, here's how to provide alternative text for the Home screen icon using the `apple-mobile-web-app-title` meta tag:

```
<meta name="apple-mobile-web-app-title" content="My App Name">
```

Other platforms using Apple's link

Some versions of the Android, BlackBerry, and Nokia browsers use the `apple-touch-icon` link element (yes, even using the Apple name).

Nokia Browser for Symbian. Since version 8.3.1 of the Symbian browser, Nokia has supported the `apple-touch-icon` link element and also a `nokia-touch-icon` link element with the same syntax. Using the `nokia`-prefixed one, we can differentiate Symbian icons from other platform icons. Symbian icons should be 54×54 pixels.

We can add support for Symbian icons as follows:

```
<!-- Nokia Symbian -->
<link rel="nokia-touch-icon" href="icons/54.png">
```

Nokia Browser for MeeGo. The browser available on the Nokia N9 also supports the `apple-touch-icon` link element and expects an icon size of 80×80 pixels, reading the `sizes` attribute when the user adds the page to the Home screen (as seen in Figure 7-7). There is no support for this icon in the Symbian or S40 edition of the browser.

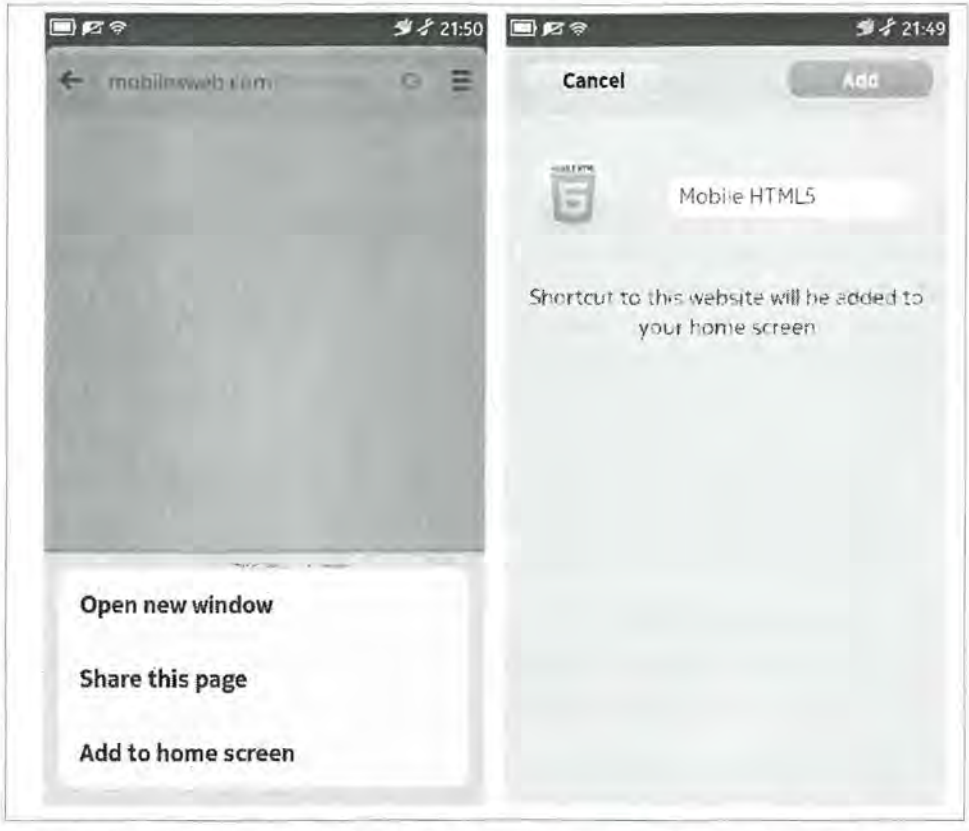


Figure 7-7. The Nokia Browser for MeeGo (available on the Nokia N9) supports 80×80-pixel icons using the same apple-touch-icon declaration, but be aware that your icon will be inside some Nokia-specific rounded corners.

We can add support for icons in the Nokia Browser for Meego using:

```
<!-- Nokia MeeGo -->
<link rel="apple-touch-icon" href="icons/80.png" sizes="80x80">
```

BlackBerry browser. The BlackBerry browser for devices running OS versions 7.0 and 7.1 also supports the `apple-touch-icon` link element when adding a website to the Home screen, but it does not support the `sizes` attribute. The BlackBerry browser supports two different icon sizes, depending on the current device's resolution: 68×68 and 92×92 pixels. However, because it ignores the `sizes` attribute, in the end it will just take the last definition of a series and resize it if necessary, as seen in Figure 7-8.

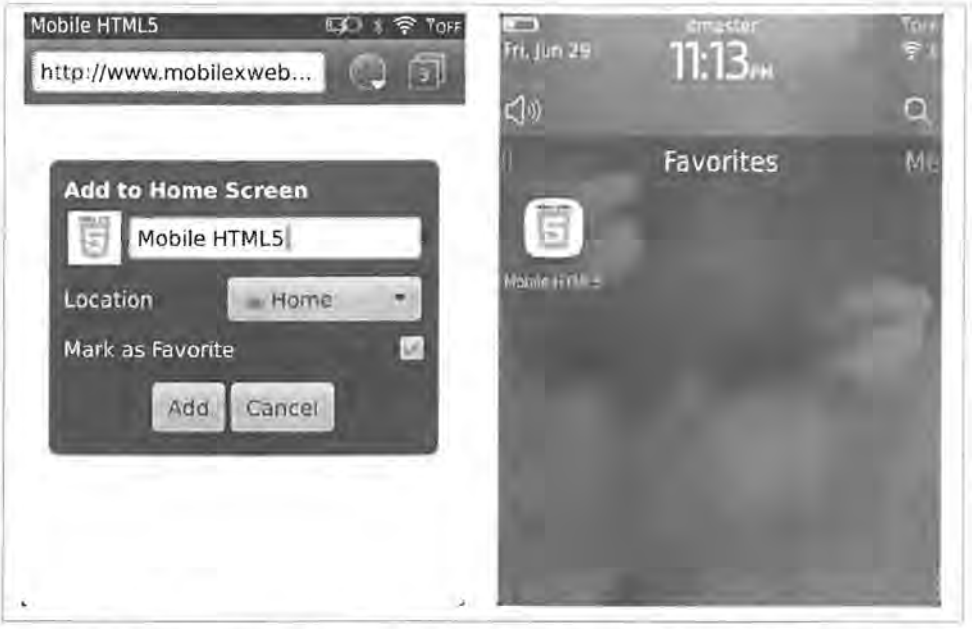


Figure 7-8. The BlackBerry browser on OS 7.x supports the apple-touch-icon link element but ignores the sizes attribute; it will take the last declaration and resize the icon to 68×68 or 92×92 pixels.

The browser on BlackBerry 10 devices also allows us to define the icon for the Home Screen menu using the Apple's syntax. The recommended icon size is 150×150 pixels.

Here's how to define the icon for BlackBerry devices:

```
<!-- iOS and Nokia icons -->
...
<!-- BlackBerry icon, last declaration -->
<link rel="apple-touch-icon" href="icons/bb.png" sizes="others">
```

Android browser. From version 1.5 to version 2.1,the Android browser supports the `icon` link and the `apple-touch-icon-precomposed` link; from version 2.2 it also supports the `apple-touch-icon` link. Android ignores the `sizes` attribute, like the BlackBerry browser. Therefore, if we want to provide an icon for iOS devices and a different icon for Android in the same HTML, we can use the following trick:

```
<!-- Android icon precomposed so it takes precedence -->
<link rel="apple-touch-icon-precomposed" href="icons/android.png"
      sizes="1x1">
```

Android devices ignore the `sizes` attribute, but if the browser finds different icon declarations, it will take the last one. iOS devices after 4.0 will ignore the `sizes="1x1"` link because it has an invalid value.

When you have different declarations, the precomposed declaration has priority. If you don't have any precomposed declarations, Android will use the first link available and not the last, unlike the BlackBerry browser.

Providing only a website icon with `rel="icon"` or providing a small Home screen icon makes the Android browser use a small icon inside a standard bookmark icon, as seen in Figure 7-9.

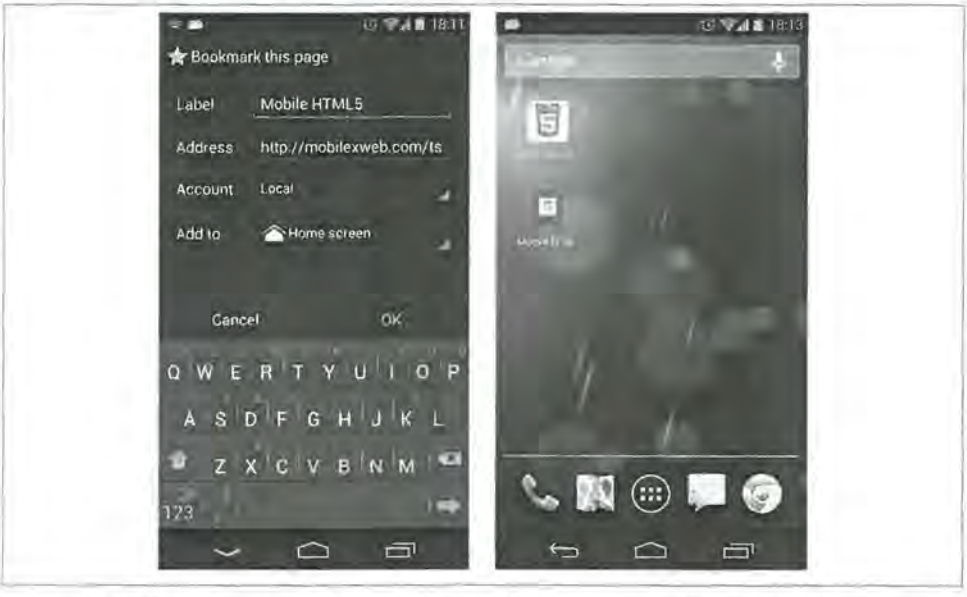


Figure 7-9. The process to add an icon to the Home screen on the Android browser may vary per Android version and manufacturer.

The Android icon sizes for the Home screen (Launcher) are:

- Low-density screens: 36×36 pixels
- Medium-density screens: 48×48 pixels
- High-density screens: 72×72 pixels
- Ultra-high-density screens: 96×96 pixels

There is no way to define four different icons in HTML right now. Therefore, the best solution is usually to deliver a 96×96 icon, which will be automatically resized as necessary; we can also use a server-side technique to provide the right one, but this probably involves more work than it's worth. From version 3.0, the Android browser adds rounded corners to the icon.



Google Chrome for Android doesn't have a way to add icons to the Home screen at the time of this writing. Firefox for Android supports adding a bookmark to the Home screen, but it has no support for `apple-touch-icon`. Instead, it uses the `rel="icon"` link for the Home screen icon, if it has enough resolution. Opera for Android always uses a default icon when adding a website to the Home screen.

In Android the applications menu has icon support, as well as the home screen. There is no way to add an icon in the applications menu using this technique, though. You need to create a native application to insert an icon into the applications menu.

Android 2.3 and later do not have an option for "Add to Home screen." However, on 2.3, if the user adds the page to his Bookmarks, there is an "Add shortcut to Home" contextual option. On 3.x and 4.x the user can add a Bookmark and select the Home screen as the destination.



As device manufacturers may customize the Android browser, there are some versions on some devices that will never take your icon and will always insert a default bookmark icon on the Home screen.

Windows Start tiles

The Metro user interface available in Windows Phone and Windows 8 includes a way for Internet Explorer to "pin" a website to the Start screen—a sort of Home screen. The icons are called *tiles*, and the way to define what is shown there differs according to the Windows and IE versions.



Opera for desktops has a way to define an icon through the `link` element—even with `apple-touch-icon`—for the Speed dial access (a sort of Opera Home screen). With meta tags and/or HTTP headers, we can even customize how the icon and mini view render, and we can update it frequently. Unfortunately, this feature is not enabled at the time of this writing for Opera Mobile.

In Windows Phone 7.x, there is no official way to define the tile. When the user pins a website, the browser will automatically take a screenshot of a portion of the rendered HTML. The trick to define your own icon is to create a separate page that includes the icon you want. Google, for example, will suggest that the user add it to the Home screen, then take the user to a second page containing the desired icon, as seen in Figure 7-10; IE then takes the screenshot and creates the final icon.



Figure 7-10. The Google home page creates a hack to provide its customized tile for the Start screen on Windows Phone 7.x.

In Internet Explorer 10, from Windows 8, we can define our own tile icon and the background image to be used in the Start screen when the user pins the site, as we can see in Figure 7-11. To do so we use the `msapplication-TileImage` and `msapplication-TileColor` meta tags:

```
<meta name="msapplication-TileImage" content="tile.png">
<meta name="msapplication-TileColor" content="#ef0303">
```



Figure 7-11. Internet Explorer 10 allows us to "pin" websites to the Start screen and define our own tile icon and preferences.

The tile image should be a 144×144-pixel PNG file, preferably with a transparent background; the tile color accepts as a value a CSS color that should be the main color of your logo or visual presence. We can use hexadecimal RGB values (such as `#333333`), named values (such as `red`), or CSS functions, such as `rgb`. If you don't provide a color, IE will use the most prominent color of your website icon or your tile image.



To fit in with Metro styles, you can just use a small white or black icon over a plain color instead of a big colorful image.

The image and color you specify are also used in history searches and other places inside Internet Explorer.



The IE10 tile size of 144×144 is the same as the high-resolution iPad's icon size. However, be careful using the same icon for both platforms, as the idea behind how they are rendered on the screen is really different.

Internet Explorer 10 will use the title of the current page for the tile's title. If we want to define a different one, we can use the `title` meta tag.

Live badge notifications. Windows 8 includes the ability to update the badge number for a tile in the Start screen, so it becomes a *live tile*. Internet Explorer will poll for updates for pinned sites if we provide the right details.

The first action is to create a badge XML file, using a predefined template. This is a really simple XML file that defines the current badge value (number) to show inside the tile. Of course, we can make a dynamic XML file.

The XML looks like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<badge value="5" />
```

Then we define the live badge notification settings, using an `msapplication-badge` meta tag. We provide the polling frequency and the badge XML URL:

```
<meta name="msapplication-badge"
      content="frequency=1440;polling-uri=http://mysite.com/badge.xml">
```

The possible values for `frequency` are:

- `30` for 30 minute polling
- `60` for 1 hour polling
- `360` for 6 hours polling
- `720` for 12 hours polling
- `1440` for 1 day polling (default value)

Summary of Home screen icon compatibility

Table 7-3 explores browser compatibility for Home screen icons in HTML documents, including sizes and implementation method. If you want to support all of these browsers in the same HTML, you can use the following snippet, in this specific order:

```
<!-- iPad high-resolution -->
<link rel="apple-touch-icon" href="icon_144.png" sizes="144x144">
<!-- iPhone/iPod high-resolution -->
<link rel="apple-touch-icon" href="icon_114.png" sizes="114x114">
<!-- Nokia N9 -->
<link rel="apple-touch-icon" href="icon_80.png" sizes="80x80">
<!-- iPad low-resolution -->
<link rel="apple-touch-icon" href="icon_72.png" sizes="72x72">
<!-- iPhone/iPod low-resolution -->
<link rel="apple-touch-icon" href="icon_57.png" sizes="57x57">
<!-- Nokia Symbian -->
<link rel="nokia-touch-icon" href="icons_54.png">
```

```
<!-- BlackBerry 7.x -->
<link rel="apple-touch-icon" href="icon_92.png" sizes="1x1">
<!-- Android Browser (and also iOS 3.2 and older) -->
<link rel="apple-touch-icon-precomposed" href="icon_96.png"
      sizes="1x1">

<!-- Internet Explorer 10 icon -->
<meta name="msapplication-TileImage" content="tile_144.png">
<meta name="msapplication-TileColor" content="#ef0303">
```

Table 7-3. Home screen icon display compatibility table

Browser/platform	Supported link rel types	Supports sizes attribute	Icon sizes to provide
Safari on iOS	`apple-touch-icon` and `apple-touch-icon-precomposed`	Yes	57×57 for iPhone and iPod touch low resolution, 72×72 for iPad low resolution, 114×114 for iPhone and iPod touch high resolution, 144×144 for iPad high resolution
Android browser 1.5 to 2.1	`apple-touch-icon-precomposed`	No	72×72 (will be resized on some devices)
Android browser 2.1, 3.x, and 4.x	`apple-touch-icon` and `apple-touch-icon-precomposed`	No	96×96 (will be resized on some devices)
Nokia Browser for Symbian	`apple-touch-icon`, `nokia-touch-icon`, and `touch-icon`	No	54×54
Nokia Browser for MeeGo	`apple-touch-icon` and `apple-touch-icon-precomposed`	Yes	80×80
BlackBerry 7.x and 10	`apple-touch-icon` and `apple-touch-icon-precomposed`	No	92×92 (will be resized on some devices) 150×150 for BlackBerry 10
Firefox for Android	`icon`	No	96×96 (will be resized on some devices)
Internet Explorer 10	No link, uses meta tag	No	144×144 (background color can be defined)



The HTML5 specification supports the `sizes` attribute for the `<link rel="icon">` element with a different syntax from the Apple solution, but at the time of this writing no mobile web browser is using this feature.

Native and installed web app icons

In Chapter 16, we will see how to deal with icons for native and installed web apps through package metadata.

The Viewport

We talked about the MIME type and DOCTYPE in the last chapter. As you saw, these are very helpful in telling browsers that documents are prepared for mobile delivery. However, this is not enough for mobile browsers that can read any desktop website. Those browsers treat the pages differently if they are for desktop users or are optimized for mobile devices. The differences are in the initial zoom scale and some possible changes in the layout.

If you are creating a mobile-optimized version of your website, you need to tell the browsers to be aware of this. The most common approach to say to the browser, "Hey, I'm a mobile website" is to use a `viewport` declaration.

The problem

The `viewport` is the area in which the page fits. The W3C defines it as "a window or other viewing area on the screen through which users consult a document." In classic web development, the viewport was usually exactly the current browser window.



Some mobile browsers, such as Safari on iPhone and iPod, define a default value of 980 pixels for the viewport width. You should redefine this if you want to take control of how many pixels you are going to use on the screen.

The W3C defines the `canvas` as "the space where the formatting structure is rendered." The canvas is conceptually infinite, so we can define a width of 10,000 pixels, and it usually works. Different algorithms can define the canvas's limits, depending on the browser, how we are defining an element's dimensions, and its content (such as an image greater than the current viewport's size).

If the canvas is larger than the viewport, browsers usually add scroll bars and implement panning behaviors on mobile devices.

In classic browsers, if you shrink the window, the viewport shrinks. In mobile browsers, starting with Safari for iPhone in 2007, that's not what happens. In mobile browsers, there is no "window" concept. Therefore, as the iPhone 5 has a screen width of 640 pixels and the iPhone 3GS has a screen width of 320 pixels, it seems logical to assume that the viewport—the area available for your content—is 640 pixels wide on the iPhone 5 and 320 pixels wide on the 3GS. That's not true.

In Figure 7-12, the Apple website is using the default viewport. Therefore, on the iPhone 3GS and iPhone 5 the website is being rendered as if it were a 980 pixel-wide window on a desktop, shrunk to fit the smaller screen sizes.



Figure 7-12. Apple.com has no mobile website or viewport declaration, so Safari defines a desktop-like viewport, using a width of 980 pixels, even though the iPhone 3GS (left) has a screen width of 320 pixels and the iPhone 5 (right) has a width of 640 pixels.

The viewport's default size may vary per browser, but it has no direct relationship with the screen's width, as on desktops.

Safari on iOS (for all devices, including the iPad) has a default value of 980 pixels wide for the viewport. That means that if we render an image that is 980 pixels wide, it will be scaled down to fit the available width on every iPhone and iPod touch on the market. It also means that if we define a body's child element as 100% width it will be rendered as 980 pixels wide in the canvas, if there are no margins and no padding.

You may be wondering why a default value of 980 is used, when the real widths are 320 or 640 pixels. The answer is that Safari on the iPhone/iPod believes that your website is optimized for desktops by default, and usually all desktop websites work properly in a 980-pixel-wide browser window. Of course, there are not 980 real pixels on the iPhone in portrait mode; therefore, Safari scales the page to fit the current real width.

In mobile browsers, the viewport concept has been divided in two:

- The *initial viewport* or *device viewport* is the "real" viewport that the current screen has, without any rules applied. This is like the classic concept of the viewport available on most desktop browsers (for example, 320 pixels wide on the iPhone 3GS).
- The *actual viewport* or *layout viewport* is the current viewport, imaginary at some point, that was applied by the browser because of its default behavior or the developer's definition—for example, 980 pixels wide on the iPhone and iPod, if we don't override the defaults. This is the important viewport from a CSS and HTML perspective.

From a designer and developer's perspective, it's like having a 980 pixel-wide browser window that someone has applied the *Honey, I Shrunk the Kids* machine to. Therefore, to get a horizontal scroll bar we need to define an element with a width greater than 980 pixels (in this particular case), ensuring that the canvas will be larger than the actual viewport.



Before Safari on iOS, all mobile browsers defined the actual viewport as the initial viewport. Therefore, as the screens on mobile devices are usually smaller than desktop windows, websites that were not mobile-optimized had horizontal scrolling (the canvas was larger than the viewport).

When the actual viewport is a different size than the initial viewport, a formula is applied to the dimensions to determine how the canvas is rendered. First, the viewport relationship needs to be calculated—that is, the initial viewport size divided by the current viewport size.

For example, if the width of the initial viewport is 320 and the width of the current viewport is 980, the calculation for the initial zoom factor equals 0.3265. That is the ratio of the zoom that is being applied to the canvas when it is rendered on the first load.



To simplify the problem, some mobile browsers change the current viewport size if the canvas doesn't fit inside, overriding defaults or developer declarations. Therefore, if you change the canvas dimensions—for example, increasing the size of an image—the current viewport size will be increased and the zoom factor will be recalculated, resulting in an automatic zoom-out behavior. Usually, the default limit for a zoom-out operation is the bigger of the current canvas size and the current viewport size.

If you're confused, just give me a couple more minutes. When the user zooms in or out using the pinch gesture or any other zoom gesture (such as double tap), the zoom ratio is changed from the original calculation, and then scroll bars may appear. On mobile browsers scroll bars are applied not in the actual viewport, but to the initial viewport. To understand how this works, Figure 7-13 shows a diagram of what is happening in the case of a canvas larger than the actual viewport that is being applied on a different initial value. An example might be a 1,500-pixel image being drawn on a 980-pixel actual viewport that is shown on a 320-pixel initial viewport.



Initial viewport and *current viewport* are names defined by the W3C CSS Device Adaptation group, while *device viewport* and *layout viewport* are names defined by the community on the Web for the same concepts.

Viewport declaration

While some browsers have a default current viewport size and others calculate it from the current canvas size, we have a method to define our own specific current viewport size though a meta tag. This method was invented by Apple on the original iPhone, and after that it was cloned by almost every modern smartphone and tablet browser out there.

For the near future, the W3C is trying to standardize it through the CSS Device Adaptation standard (*http://dev.w3.org/csswg/css-device-adapt*). The W3C is trying to move the declaration to CSS instead of using a meta tag, as we will see later.

The meta declaration follows the following syntax:

```
<meta name="viewport" content="{declaration}">
```

The `{declaration}` should be replaced with one or more `attribute: value` declarations, comma-separated.



You can think about the viewport width definition as defining a logical window size that will be used on all compatible devices, regardless of real screen size: it is a way to normalize your design to one "virtual screen."

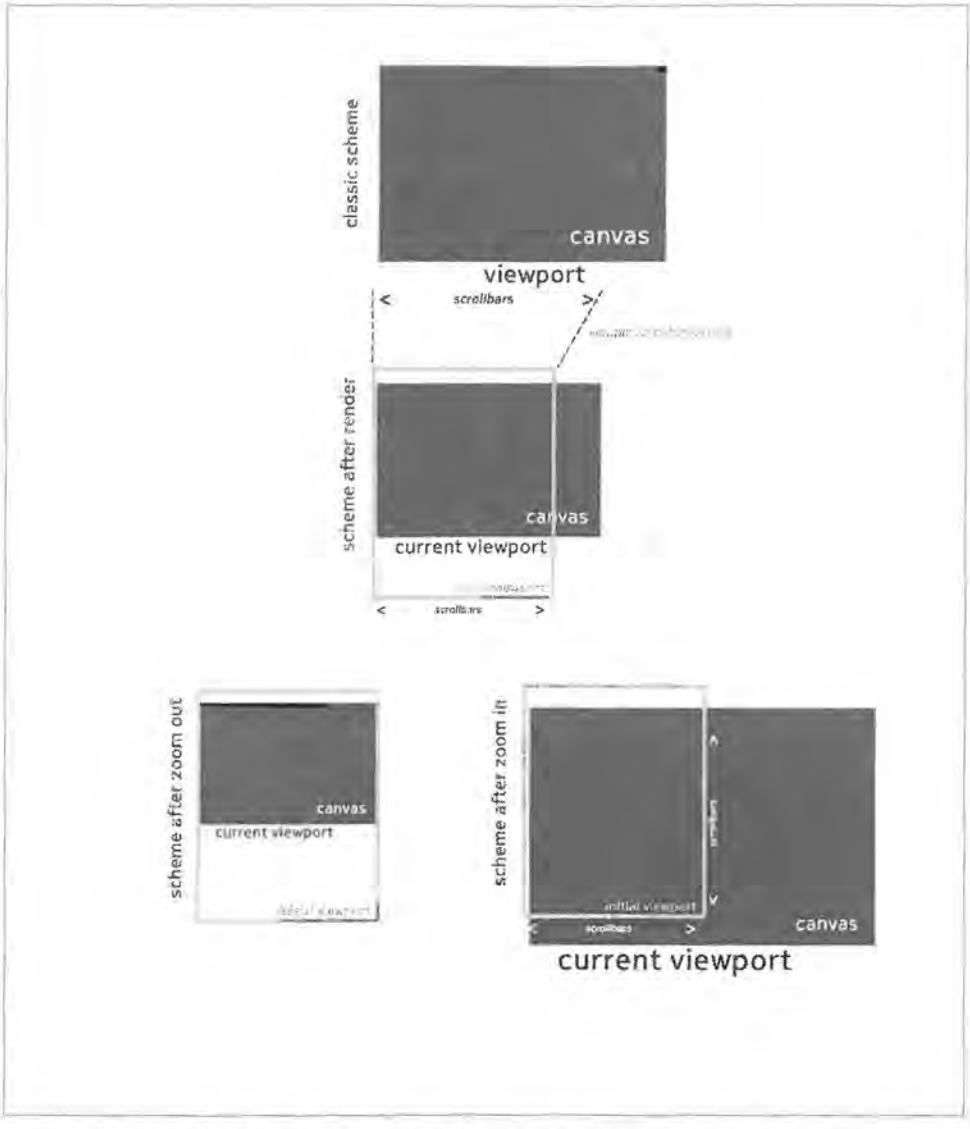


Figure 7-13. Some browsers will increase the current viewport size if the canvas is larger than the defined value, to simplify the operation.

You can specify the current viewport's width or height, and it can be larger or smaller than the total visible area of the screen (the initial viewport). This is where the initial scale and zoom features of the mobile browser come into play. If you are creating a mobile-friendly website, it shouldn't need to be zoomed in or out, so you can say to the browser that you want to start with a scale of 1:1 (current viewport = initial viewport = visible area).

The content of the tag can be a comma-separated list of one or more of the attributes listed in Table 7-4.

Table 7-4. Viewport metadata attributes

Attribute	Possible values	Description
`width`	Integer value (in CSS pixels) or the constant `device-width`	Defines the viewport width
`height`	Integer alue (in CSS pixels) or the constant `device-height`	Defines the viewport height
`initial-scale`	Floating value (0.1 to *n*); 1.0 is no zoom	Defines the initial zoom scale of the viewport
`user-scalable`	`no` or `yes`	Defines whether we will allow the user to zoom in and out in the viewport
`minimum-scale`	Floating value (0.1 to *n*); 1.0 is no scale	Defines the minimum zoom scale of the viewport
`maximum-scale`	Floating value (0.1 to *n*); 1.0 is no scale	Defines the maximum zoom scale of the viewport
`target-densitydpi`	Integer value (70 to 400) in DPI, or one of the constants `device-dpi`, `high-dpi`, `medium-dpi`, or `low-dpi`	Defines how to manage different resolutions regarding the viewport's width

You can define a viewport with a fixed size (in case you are showing a desktop-friendly website), or with a size relative to the visible area. The most common approach for our mobile template is to define the `width` as `device-width`:

```
<meta name="viewport" content="width=device-width">
```

If you don't want the user to be able to zoom in and out, you can define both the `maximum-scale` and the `minimum-scale` as `1.0`, or just set `user-scalable=no`:

```
<meta name="viewport"
      content="width=device-width,minimum-scale=1.0, maximum-scale=1.0">
<meta name="viewport" content="width=device-width,user-scalable=no">
```

Even if you believe that your design is good enough and perfectly readable for a mobile screen, some users will still want to zoom in to get a closer look at your content, so disabling zooming may make some users angry. Also be aware that some mobile browsers will always provide the zoom ability, even if you force `user-scalable=no`.



Safari on iOS will use a default viewport of 320px on iPhone/iPod and between 480px and 640px on iPad when your website has an XHTML MP DOCTYPE, or if it's hosted on a *.mobi* domain, instead of the default 980px.

Some browsers allow us to change the viewport dynamically using JavaScript, or even start without one and inject the meta tag after the page has loaded. Other browsers will

just ignore any updates to the viewport meta tag after the page has loaded. Figure 7-14 shows the same image viewed in viewports of different sizes and at different `device-width` scales.



Figure 7-14. The same 300×300-pixel image viewed in the default Safari viewport, in a 1,500-pixel viewport, at a device-width scale of 1.0, and at a device-width scale of 2.0.

What is device width?

Just when you think you're getting to grips with all of this, I have a bad news: `device-width` is not exactly the device's screen width. I've told you that usually providing a viewport with `width=device-width` for a mobile version leads to the best results. However, consider the following device examples:

- Apple iPhone 3GS, Safari on iOS, screen width = 320 pixels
- Nokia N8, Nokia Browser for Symbian, screen width = 360 pixels
- Nokia Lumia 900, Internet Explorer 9, screen width = 480 pixels
- Samsung Galaxy SII, Android browser, screen width = 480 pixels
- Apple iPhone 5, Safari on iOS, screen width = 640 pixels
- Samsung Galaxy SIII, Android browser, screen width = 720 pixels

Our first thought may be that `<meta name="viewport" content="width=device-width">` will give us a current viewport equal to the initial viewport or the screen width. Well, here comes the other tricky part of the viewport—the good news is that it exists to simplify our lives.

Most of the devices listed here report `width=320` when we define `width=device-width`, although some Android devices with wider screens (such as the Galaxy SIII) report `width=360` so you can fit more content. That means we can design to one and only one virtual device with a width of 320 pixels, and every browser will adapt the design to the initial viewport. Basically, we don't need to worry about the different screen sizes.

As many websites are optimized for a viewport width of 320 resolution-independent pixels, the Android browser on 4.x devices with wider screens (such as 4.5–5" screens) overrides the `320` value with `360` so you have more space on the screen to fit more content, instead of just resizing it. Google Chrome on the same devices uses `320` if you specify the exact value and `360` if you set `width=device-width`.



Viewporter (*https://github.com/zynga/viewporter*) is a JavaScript mini-framework created by the game developer company Zynga to simplify setting up the viewport and help immersive apps get the most out of it.

To summarize, the most-implemented width values (in resolution-independent pixels) when using `width=device-width` at the time of this writing are:

`320`
: iPhone/iPod, Android browser/Google Chrome (medium-sized screen), Windows Phone (all screen sizes), BlackBerry (all screen sizes), Firefox/Opera for Android (all screen sizes), and other phone platforms

`360`
: Android browser and Google Chrome 4.x or greater on a wider screen (4.5–5"), Symbian devices

`400`
: Android browser on large-screen devices (more than 5", also known as *phablets*, such as the Samsung Galaxy Note)

`600`
: 7" tablets, such as the Kindle Fire and BlackBerry PlayBook

`768`
: iPad, iPad mini, and other tablets

`600-1024`
: Browsers on Android-based tablets (based on device's width in resolution independent pixels)



Usually on mobile phones `device-width` is `320`, as this is the average screen width on devices with a medium-density screen. Scales are measured on medium-resolution devices; a scale of 1:1 on a high-resolution device such as the iPhone 5 is defined in CSS pixels, and it will give you 320 pixels as the current viewport width instead of the real 640 device pixels.

Give me back the pixels

In some specific situations—such as for immersive apps or games—you may want to have a current viewport equal to the initial viewport, so that you have available every pixel that is there on the screen. That is, on the Galaxy SII, your viewport will have 480 pixels, and on the Galaxy SIII 720 pixels, instead of both being resized to 320. Of course, in this situation it's up to you to manage differences, and you need to be careful about minimum font sizes and other visible stuff.

The Android browser added an extension to the viewport meta tag to provide this feature: the `target-densitydpi` attribute. At the time of this writing, the Android browser, BlackBerry browser 7.0+, Chrome for Android, Opera for Android, and Nokia Browser for Symbian support this extension.

It basically defines the DPI values that we want to use for the width and height viewport definitions. We can use a specific DPI value (such as `target-densitydpi=320`), or some Android constants, such as `low-dpi`, `medium-dpi` (the default), or `high-dpi`. If we want every pixel for us, we need to use `width=device-width` and `target-densitydpi=device-dpi`:

```
<meta name="viewport"
      content="width=device-width,target-densitydpi=device-dpi">
```



Safari on iOS will just ignore the `target-densitydpi` declaration. If you want to have every iOS pixel for you, you need to manually define double the width for the viewport for high-resolution devices, such as `width=640`. However, this isn't a good idea in most situations.

Landscape behavior

What happens when the user switches to the landscape orientation? The initial viewport gets wider and shorter, but your current viewport usually maintains the default width value, or the value you defined in the viewport meta tag. Therefore, if you provide a specific value, such as `width=320`, when the user switches to landscape the result will be the viewport stretching your content, as you can see in Figure 7-15.

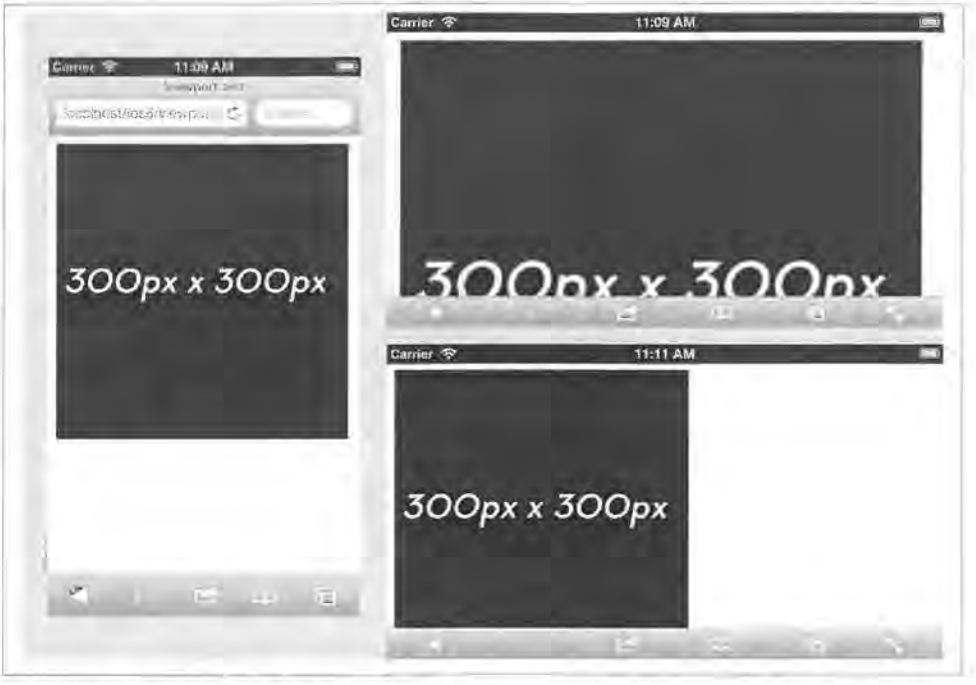


Figure 7-15. By default, Safari on iOS will maintain your current viewport when going to landscape (top right), but with some hacks you can force it to give you a new current viewport with all the available space (bottom right).

The problem with the lack of standards comes when you define `width=device-width`. There are two approaches:

- Take the portrait's `device-width` as the landscape's width value.
- Take the portrait's `height` value as the landscape's width value.

Safari on iOS and the Nokia Browser for MeeGo always use the `device-width` in portrait mode as the width value, so basically when you go landscape you are not getting more width for your content; your canvas is just resized. Other browsers understand that now the `device-width` should be the previous available height in portrait mode, which means you will have a wider current viewport by default.



Be careful when trying to get all the possible width in landscape mode, because some devices on the market are wider than others in this orientation. Just take the example of the iPhone 4 and iPhone 5: you will get 480 CSS pixels (960 device pixels) and 568 CSS pixels (1,136 device pixels), respectively.

While Apple appears to see this as the proper result—your website will render exactly as in portrait mode, but bigger—some developers refer to this as the *iPhone Safari Viewport Scaling Bug*. One solution is to use `maximum-scale=1.0` on the viewport. That forces Safari on iOS not to use a scale greater than 1, so you will always get 1:1 scale. The problem is that you are also limiting the user's rights to scale the content.

The solution for Safari involves some JavaScript that changes the viewport meta tag dynamically based on the context.



Some versions of Safari also have a bug on the orientation change event where the viewport is being scaled or moved from the original position. The JavaScript hacks, such as the solution presented by Scott Jehl on his GitHub (*https://github.com/scottjehl/iOS-Orientationchange-Fix*), usually fix this strange behavior.

For example, Shi Chaun devised a solution, available on his blog BigHub (*http://www.blog.highub.com*), and later improved on at *www.gist.github.com/901295*—that involves these steps:

- Change the viewport's width when the screen orientation changes.
- Define `maximum-scale=1.0`, capture the pinch gesture (when the user zooms), and dynamically reset the viewport's `minimum-scale` and `maximum-scale` to the defaults when that occurs.

This solution involves defining the viewport as follows:

```
<meta name="viewport" content="width=device-width,initial-scale=1.0">
```

And using the following JavaScript code:

```
(function(doc) {

    var addEvent = 'addEventListener',
        type = 'gesturestart',
        qsa = 'querySelectorAll',
        scales = [1, 1],
        meta = qsa in doc ? doc[qsa]('meta[name=viewport]') : [];

    function fix() {
        meta.content = 'width=device-width,minimum-scale=' +
                       scales[0] + ',maximum-scale=' + scales[1];
        doc.removeEventListener(type, fix, true);
    }

    if ((meta = meta[meta.length - 1]) && addEvent in doc) {
        fix();
        scales = [.25, 1.6];
        doc[addEvent](type, fix, true);
```

```
    }
}(document));
```



Be careful when testing viewport changes on simulators or real devices because some of them cache current viewport settings in memory and if you just refresh, you will not see the changes reflected. You need to close the window and start again, or type the URL again.

Viewports in CSS

While Apple went for the meta tag and every mobile browser today supports it, the W3C, based on an Opera proposal, is going to a CSS-based solution for the future viewport definition. At the time of this writing, only Opera Mobile and Internet Explorer 10 support the CSS viewport standard, defined as a draft at www.dev.w3.org/csswg/css-device-adapt.

The definition includes the `@viewport` declaration that today may be prefixed by vendors, as in `@-ms-viewport` on IE10 or `@-o-viewport` for Opera browsers. For example:

```
@viewport {
    width: 320px;
}
@-ms-viewport {
    width: 320px;
}
@-o-viewport {
    width: 320px;
}
```

The attributes compatible with the viewport CSS declaration are listed in Table 7-5.

Table 7-5. Viewport CSS declaration attributes

Attribute	Possible values	Related meta attribute	Description
`max-width`, `min-width`	Any CSS measure, `auto`, or the `device-width` constant. Usually a constant with px unit.	None	Defines the viewport maximum or minimum width
`width`	Any CSS measure, `auto`, or the `device-width` constant. Usually a constant with px unit.	`width`	A shortcut for both `max-width` and `min-width` at the same time
`max-height`, `min-height`	Any CSS measure, `auto`, or the `device-height` constant. Usually a constant with px unit.	None	Defines the viewport maximum or minimum height
`height`	Any CSS measure, `auto`, or the `device-height` constant. Usually a constant with px unit.	`height`	A shortcut for both `max-height` and `min-height` at the same time

Attribute	Possible values	Related meta attribute	Description
`zoom`	Floating value (0.1 to *n*) or percentage; 1.0 is no zoom.	`initial-scale`	Defines the initial zoom scale of the viewport
`user-zoom`	`fixed` or `zoom`.	`user-scalable`	Defines whether we will allow the user to zoom in and out in the viewport (`zoom`) or not (`fixed`)
`min-zoom`	Floating value (0.1 to *n*) or percentage; 1.0 is no scale.	`minimum-scale`	Defines the minimum zoom scale of the viewport
`max-zoom`	Floating value (0.1 to *n*) or percentage; 1.0 is no scale.	`maximum-scale`	Defines the maximum zoom scale of the viewport
`orientation`	One of the constants `auto`, `portrait`, or `landscape`.	None	Forces browser orientation lock to `portrait` or `landscape`; `auto` means accepting default browser values

One of the biggest advantages of defining the viewport inside CSS is that it allows us to use CSS media queries, which we will cover in Chapter 11. That means we can say things such as, "If the current device's width is less than 480 pixels, then define a current viewport of 320 pixels; if not, fix it at 800 pixels." In other words, it's easy to implement threshold values and make fixed viewport declarations. The same idea can be applied to other conditionals, such as the current orientation.

For example, Internet Explorer 10 for Windows 8 supports viewport definitions for different orientations with the following syntax:

```
@media screen and (orientation: landscape) {
  @-ms-viewport {
    width: 1024px;
    height: 768px;
  }
  /* CSS for landscape layout goes here */
}

@media screen and (orientation: portrait) {
  @-ms-viewport {
    width: 768px;
    height: 1024px;
  }
  /* CSS for portrait layout goes here */
}
```



Internet Explorer in Windows Phone 8 has some unexpected behavior when working with viewports. If the `meta` tag is used with `width=device-width`, the viewport is always defined as 320 pixels. On the other hand if the CSS viewport declaration is used with `width: device-width`, the current real device width is used, such as 480 on medium-resolution devices and 768 on high-resolution devices. Using the CSS version will create different size experiences on different devices.

Viewport compatibility. Table 7-6 shows what happens if you try the viewport meta tag in different browsers, to see which ones detect it and do something with it. Remember that we can still add it in noncompatible browsers because the meta tag accepts any content.

Table 7-6. Viewport usage compatibility table

Browser/platform	Default width	Meta version	CSS version	Can disable zoom	target-density-dpi support	device-width on landscape
Safari on iOS	980px	Yes	No	Yes	No	PW
Android browser 2.x	800px or CW	Yes	No	Yes	Yes, 2.2+	PH
Android browser 3.x and 4.x	980px	Yes	No	Yes	Yes	PH
Chrome for Android	980px	Yes	No	No	Yes	PH
Nokia Browser for Symbian	IV or CW	Yes, 5th+	No	Yes	Yes, ^3+	PH
Nokia Browser for MeeGo	980px	Yes	No	Yes	Yes	PW
Nokia Browser for S40	IV	Yes, 2011+	No	No	No	N/A
BlackBerry browser for smartphones	IV or CW	Yes, 4.6+	No	Yes	Yes, 7.0+	PH
BlackBerry browser for PlayBook and BB10	IV or CW	Yes	No	Yes	Yes	PH
Internet Explorer	1024px	Yes, 7+	Yes, 10+ (`-ms-` prefix)	Yes	No	PH
Opera Mobile (Presto)	980px or CW	Yes	Yes, 11+ (`-o-` prefix)	Yes	Yes	PH
Opera Mini	980px or CW	Yes from 6	Yes from 6	Yes	Yes	PW
Firefox	980px	Yes	No	Yes	No	PH
webOS browser	980px or CW	Yes	No	No	No	PW

[a] IV: initial viewport's width; CW: canvas width if wider than the default value.

[b] PH: portrait's height; PW: portrait's width.



While on smartphones usually `device-width` is `320` or `360`, on tablets the `device-width` value may change per device. The iPad (even third generation) will give you `768`, Chrome for Android and the Android browser use the initial viewport's width (changes per device), and the BlackBerry browser for PlayBook will give you `600`.

Viewports for older devices. Devices shipped before 2009 didn't work with the viewport meta tag. However, some platforms offered an alternative meta tag that defines a similar concept: "Don't try to zoom out or reflow my content because it's already mobile-optimized."

BlackBerry and some other platforms use this meta tag for defining mobile-friendly documents:

```
<meta name="HandheldFriendly" content="True" />
```

This meta tag is still detected on modern browsers because of backward compatibility, but should use the viewport for today's devices. If we define both, the viewport will take precedence.

Internet Explorer Mobile (formerly Pocket IE) introduced this meta tag in Windows Mobile 5—like the `HandheldFriendly` tag, it still works today, but the viewport meta tag takes precedence if both are defined:

```
<meta name="MobileOptimized" content="width" >
```



Mobile Internet Explorer allows us to activate the ClearType technology to smooth fonts for easier reading using the tag `<meta http-equiv="cleartype" content="on">`.

A not-so-standard variation is to use the `alternate` link meta tag. This is intended to be used in desktop documents, defining an alternative URL for the same content intended for viewing on different media (`handheld`, in this case):

```
<link rel="alternate" media="handheld" href="http://m.mysite.com" >
```

Some mobile sites (like the Google mobile home page) also include the same `link` tag in the mobile page with an empty `href` attribute, like a flag saying that this is alternative content for mobile devices and should not be considered as duplicated content:

```
<link rel="alternate" media="handheld" href="" >
```

I don't have real evidence yet that this works in any mobile browsers or for search engine optimization (SEO) purposes, but it won't do any harm.



Defining mobile meta tags can be useful for transcoders, helping them determine whether to show the mobile version as we've created it rather than transcoding the content as a full website. We will talk about transcoders in Chapter 9.

Changing the Navigation Method

The Symbian browser (on third edition and later devices) has two possible methods of browsing on devices with keyboards: the standard/normal way (cursor-based) and a hidden focus-based mechanism. A meta tag available for these devices allows us to change the default navigation method to a simple focus mechanism:

```
<meta name="navigation" content="tabbed">
```

This should be used only if you have a vertical tabular design (for example, a list of links using the whole width of the page). Using this tag will disallow the mouse events and hover effects over the page. If your design supports only vertical navigation, focus-based navigation will be faster for the user than the standard cursor navigation.

BlackBerry browser for smartphones (version 7.0+) and all versions of the PlayBook and BB10 platforms support two meta tags: `cursor-event-mode` and `touch-event-mode`. The first one defines how to handle trackpad events. The default value is `processed` and means the browser will take care of some UI behaviors, such as click to zoom or click and hold to display a context menu. If we want to override that behavior and have total control of the trackpad events from JavaScript, we should use the `native` value, as follows:

```
<meta name="cursor-event-mode" content="native">
```

We can create a similar effect on touch devices for the default touch gestures using the `touch-event-mode` meta tag. By default, the value `processed` is applied, meaning that the browser handles some touch gestures (such as pinch-to-zoom, conversion from touch events to mouse events). We can override all the defaults using the `native` value:

```
<meta name="touch-event-mode" content="native">
```

or use the value `pure-with-mouse-conversion`, allowing us to detect some touch events while others are passed directly to the browser. We'll talk about touch events later, in Chapter 15.



Later in this book we will cover other meta tags that we can use for specific purposes on some platforms, but that are not useful for all kinds of websites.

Removing Automatic Links

Safari on iOS and the BlackBerry browser include an automatic behavior that can lead to confusion and problems: every number in the body content that looks like a phone number (the same pattern) will be converted to a call action link. We'll cover how to create this kind of link in a later chapter, but for now you just need to be aware that this automatic link behavior can lead to some weird false positives.

To remove the automatic behavior, Safari on iOS looks for the following meta tag:

```
<meta name="format-detection" content="telephone=no">
```

The BlackBerry browser uses a different meta tag to disable the automatic link function:

```
<meta name="x-rim-auto-match" content="none" forua="true">
```

If we don't provide this meta tag, the BlackBerry browser will also autodetect email addresses and convert them to links.

Metadata for Sharing

Sharing on social networks is a popular feature with users. Modern mobile browsers, such as Safari on iOS 6 (see Figure 7-16), the BB10 browser, and Internet Explorer 10, include a simple way to share the currently open website on Twitter or Facebook. However, to enable our mobile websites to be shared properly, we need to include some metadata.

Sharing a mobile web page raises the question, "What happens when a follower or friend tries to access this URL on a desktop browser?" We have different options for handling this, depending on the architecture of the mobile solution.

If you are providing a different URL for mobile devices, you can:

- Deliver the same experience to desktop users (the default, not recommended).
- Redirect the user to a desktop version of the same URL. For this, you need to have a mapping mechanism from mobile to desktop web and vice versa.

If you are using the architecture of one URL for all devices, either because you are using Responsive Web Design or a server-side detection and delivery mechanism, you don't need to do anything about the URL.

Some browsers leave the hard work of reading our websites and getting useful content (such as the title and a description) up to the social networks, while others do the dirty work themselves. In both cases, the information that we should provide is the same, as (fortunately) most elements are part of the Open Graph standard (*http://ogp.me*).



Figure 7-16. Some mobile browsers allow the user to share the current page easily on different social networks—we should prepare our content so it's optimized for that operation.

The most useful meta elements you can define on your website for sharing are:

`title`
: The page title. While the `title` element is usually fine, sometimes we want to change the title when a page is being shared. For this purpose, some platforms use a `title` meta tag, if it's defined. IE10 uses this tag to prepare the preview content.

`description`
: A short description of the current document. IE10 uses this tag to prepare the preview content.

`og:title`
: Same as `title`, but from the Open Graph standard. This is used when the social network is gathering information from our document.

`og:description`
: Same as `description`, but from the Open Graph standard. This is used when the social network is gathering information from our document.

`og:image`
: The URL of the image that will be shown with the title and description on the social network

You can find more information on the Open Graph protocol website (*http://ogp.me*).

The following is an example of the sharing metadata we can use:

```
<meta name="title" content="Programming the Mobile Web, 2nd edition">
<meta name="description" content="A book with all the information you
  can find on mobile web development using HTML5">
<meta name="og:title" content="Programming the Mobile Web, 2nd
  edition">
<meta name="og:description" content="A book with all the information
  you can find on mobile web development using HTML5">
<meta name="og:image" content="http://mobilexweb.com/images/cover.png">
```



The classic desktop web meta options, like `refresh` and `cache-control`, work well on mobile browsers. Usage of the `refresh` meta tag for autoupdating documents is not good practice for mobile devices, though: it is difficult to scroll on some mobile browsers, and an unsolicited page refresh can be unpleasant for the user. You can do an Ajax autoupdate if it is really necessary to keep the document updated.

Almost every mobile browser supports caching, either in meta tags or using HTTP headers. For example:

```
<meta http-equiv="expires"
      content="Mon, 5 Mar 2015 01:01:01 GMT">
```

Hiding the URL Address Bar

Some browsers hide the URL bar when the document is loaded, so there is more space for content. Some other browsers do this when the user starts to scroll the page; the user can get the URL back again by scrolling to the top or opening the contextual menu. And there are yet other browsers, such as Chrome for Android and almost every tablet browser, that do not support any option to hide the URL bar.

To hide the URL bar in browsers without any automatic URL-bar-hiding behavior, such as Safari on iOS or the Android browser, we just need to scroll the window to 0 pixels (iOS) or 1 pixel (Android) on the *y*-axis, as in the following example:

```
If (window.addEventListener) {  // Make sure addEventListener exists
    window.addEventListener("load", function() {
        setTimeout(function() { window.scrollTo(0, 1)}, 0);
```

```
    });
}
```

The main disadvantage of the previous code is that if the user is trying to load the page with a hash (an internal page anchor), we will still be sending her to the (0,0) coordinate of the window.

Scott Jehl (*http://scottjehl.com*) has released a full snippet, open source, that works properly in both the Android browser and Safari on iOS (and other browsers as well) that we can use to hide the address bar without encountering this problem. The fix is available at *gist.github.com/1183357*, and the latest version at the time of this writing looks like the following:

```
/*
  * Normalized hide address bar for iOS & Android
  * (c) Scott Jehl, scottjehl.com
  * MIT License
*/
(function( win ){
   var doc = win.document;

   // If there's a hash, or addEventListener is undefined, stop here
   if( !location.hash && win.addEventListener ){

      //scroll to 1
      window.scrollTo( 0, 1 );
      var scrollTop = 1,
         getScrollTop = function(){
            return win.pageYOffset || doc.compatMode === "CSS1Compat"
            && doc.documentElement.scrollTop || doc.body.scrollTop || 0;
         },

         //reset to 0 on bodyready, if needed
         bodycheck = setInterval(function(){
            if( doc.body ){
               clearInterval( bodycheck );
               scrollTop = getScrollTop();
               win.scrollTo( 0, scrollTop === 1 ? 0 : 1 );
            }
         }, 15 );

      win.addEventListener( "load", function(){
         setTimeout(function(){
            //at load, if user hasn't scrolled more than 20 or so...
            if( getScrollTop() < 20 ){
               //reset to hide addr bar at onload
               win.scrollTo( 0, scrollTop === 1 ? 0 : 1 );
            }
         }, 0);
      } );
   }
})( this );
```

Native App Integration

Some platforms allow us to integrate a website with a related native application. Therefore, if a user arrives at our website from a search engine or other source and we have a native application that can offer more services to the user, the browser will invite the user to install the app (if it's not already installed) or to open the app directly from the browser.

At the time of this writing, only iOS since 6.0 and Windows 8 allow this kind of native and web app integration, but more platforms will be getting on board soon. For updates on this feature, see www.mobilehtml5.org.

iOS Smart App Banners

Since iOS version 6.0, Apple allows us to link a website with a native app through a service called *Smart App Banners*. When the user accesses a website with a Smart App Banner definition, Safari will check if the user has the associated app installed or not. If so, it will invite the user to open the app from the website using an OPEN button. If it's not installed, it will check the current device and App Store country to see whether the user can install the app. If so, the user will be invited to install the app directly from the website using a VIEW button.

The banner shown in Figure 7-17 takes up 156 pixels (312 on high-resolution devices) at the top of the screen until the user clicks on the screen below the banner or on the close button, at which point your website gets the full height. It acts like a DOM object at the top of your HTML, but it's not really on the DOM. On the iPad, especially in the landscape orientation, it seems a little space-wasting.

If the app is not installed, the banner will show a "loading" animation for a few seconds while the system verifies whether it will work on the user's current device and whether it's available via the App Store in the user's country. If the app is not available in the user's country or will not work on the current device, the banner hides automatically (for example, if it's an iPad-only app and the user is browsing with an iPhone, or if the app is available only on the German App Store and the user's account is in the US).



Figure 7-17. The Smart App Banner appears when you don't have the app installed.

To define a Smart App Banner in an HTML document, we use a meta tag called `apple-itunes-app` with a few declarations, including:

`app-id` *(mandatory)*
: The ID of the native app.

`app-argument` *(optional)*
: Arguments that the native app will receive from the website; should be formatted as a URL.

`affiliate-data` *(optional)*
: If you participate in the iTunes Affiliate Program to earn a commission on sales, you can add the relevant information here. The format is `siteId=9999&partnerId=ID`.

To make it work, the app you want to link to needs to be already approved on the Apple App Store. Then, you can go the iTunes Link Maker (*http://itunes.apple.com/linkmaker*) to retrieve the application ID you need to use in the meta tag. The ID is a nine-digit numeric value; to find it, search for your app by name on the Link Maker and extract the number from the URL, as seen in Figure 7-18.

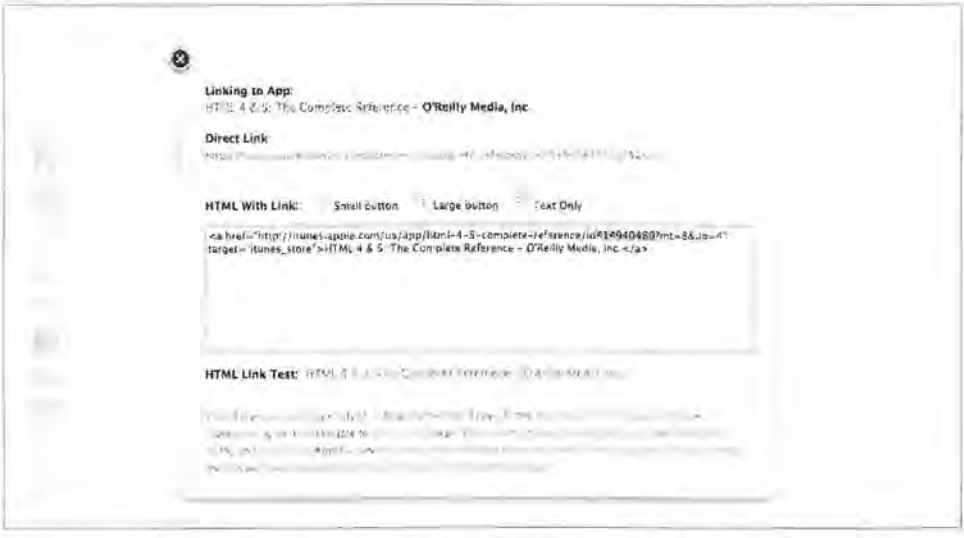


Figure 7-18. On the iTunes Link Maker website we can find the application ID to be used in the Smart App Banner meta tag.



Your native app in the App Store can also be created using HTML5, such as an Apache Cordova or Adobe PhoneGap application.

For example, to create a Smart App Banner for the app available for the iOS version of this book, we would use the following code:

```
<meta name="apple-itunes-app" content="app-id=393555188">
```

If you want to send arguments to the app, you can do this using a URL specified in the `app-argument` declaration. The URL can be the URL that the user came to the app from, where you can extract useful information, or any data in URL format that you can parse from the native side. The idea is that the user can continue working in the native app or seeing the same information that he was seeing on the website, without having to start from scratch. The value of this argument can be generated dynamically on the server or client side.



From Objective-C, you can implement the method `application:openURL:sourceApplication:annotation:`. The second parameter of that method will be the URL passed from the website.

Windows Store app connections

If you have a Windows Store app (on Windows 8), you can connect your website to it with some meta tags. Internet Explorer will show a menu like the one in Figure 7-19, allowing the user to quickly get the app from the store if it's not already installed. If the app is installed, the menu option changes to "Switch to *<name>* app."

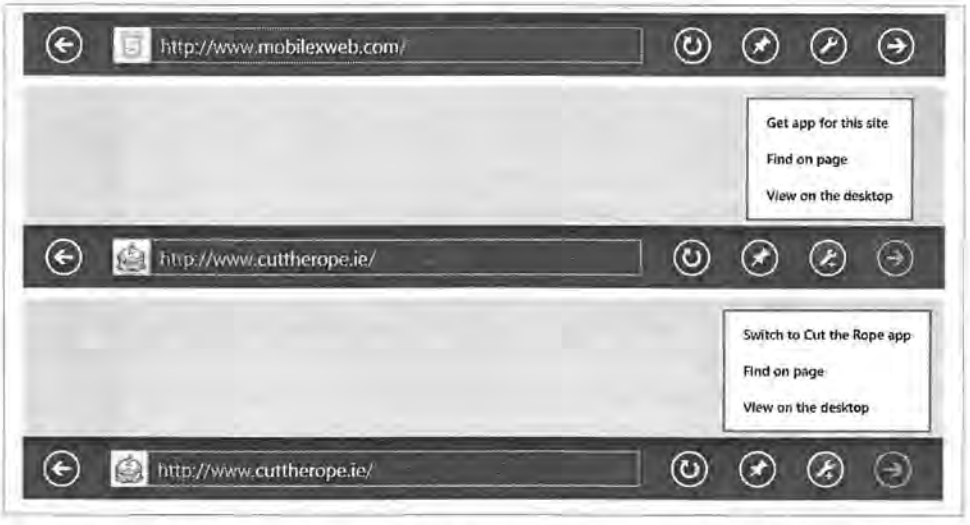


Figure 7-19. When a Windows Store app connection is defined, the tools icon changes to a + symbol and it offers an invitation to install or switch to the app.

There are five meta tags that we can define in our website for this connection:

`msApplication-ID` *(mandatory)*
: The identifier defined in the application manifest.

`msApplication-PackageFamilyName` *(mandatory)*
: The package family created by Microsoft Visual Studio to identify the app.



Instead of using Visual Studio to get the package family name, you can browse the web version of the Windows Store, locate your desired app, and look at the source code. You will find the information in a `packageFamilyName` JavaScript variable.

`msApplication-Arguments` *(optional)*
: A string that will be passed to your app. If we don't provide this meta information, Internet Explorer will pass the current URL automatically.

`msApplication-MinVersion` *(optional)*
: Enforces a minimum version of your app. Therefore, if the user has an older version of the app, when she tries to switch to it she will first be sent to the store to update it.

`msApplication-OptOut` *(optional)*
: Allows your HTML document to not use the connection in some situations. Possible values are `install` to prevent the installation invitation if the app is not installed, `switch` to prevent the switch invitation if the app is installed, and `both`.



A Windows Store app can be created using HTML5, C++, C#, or Visual Basic, and it must be approved by Microsoft. To submit an app to the Windows Store, you must have a developer account.

Cut the Rope (*http://cuttherope.ie*) is an addictive game available on many mobile platforms, such as iOS and Android; an HTML5 version, sponsored by Internet Explorer 10, is available online. The game is also available as a Windows Store app. To offer the functionality seen in Figure 7-19, the following meta tags are used:

```
<meta name="msApplication-ID" content="App" />
<meta name="msApplication-PackageFamilyName"
      content="ZeptoLabUKLimited.CutTheRope_sq9zxnwrk84pj" />
```

The Document Body

The body is the most important section of the document, as it defines the content that the user will see.

Key best practices include:

- Avoid formatting tags such as `big`, `format`, and `center`.
- Use semantically correct, clean XHTML or HTML5; we will define styles later with CSS.
- Don't create a large document. Larger documents cause problems on old browsers (and caching problems even on some modern ones).
- If you have a lot of text to show, for feature phones, separate the content into many pages.
- Don't use tables for layout.

Main Structure

A typical mobile document will be divided into four main sections:

1. Header
2. Main navigation
3. Content
4. Footer

The header should be as simple as possible, using an `h1` title or a logo or company banner. The main navigation should consist of no more than five main links, ordered by likelihood of use in a mobile context (most to least probable). The content is obvious; the footer should include very brief copyright information, a home link, a back link, and optionally other related links (such as "go to top").

This is a simplification, I know, but most mobile pages should fit this structure. If your structure is more complex, give some careful thought to whether that complexity is necessary.

The Hell of Transcoders

Some carriers have decided to install and execute in their networks a *transcoder* that proxies every mobile web request—even those made with nonproxied browsers—to create a "better experience" for the user. This is a very bad practice from a developer's perspective, for the following reasons:

- It compresses the content, the document, the CSS, the JavaScript, and the images without our consent.
- It changes our layout and design.
- It can even change our markup language.
- It removes all the original HTTP headers from the browser, blinding us from knowing which devices are accessing our websites.

Luca Passani, a well-known mobile web developer, created a manifesto in 2008 addressing content reformatting problems, available at *wurfl.sourceforge.net/manifesto*. The W3C is also taking part in this issue; it has issued a document on the Content Transformation Landscape (*http://www.w3.org/TR/ct-landscape*).

The basic document structure should look like this. Separating every section with a `div` tag is not necessary for the document definition, but it is useful later for CSS styling. The main navigation can be an unordered list (`ul`) instead of a `div`:

```
<body>
   <header>
      <h1>Mobile Web</h1>
   </header>
   <nav>
      <ul>
         <li><a href="Tests">Tests</a></li>
         <li><a href="Blog">Blog</a></li>
         <li><a href="Contact">Contact</a></li>
      </ul>
   <nav>
   <div id="content">
   </div>
   <footer>
   </footer>
</body>
```

The main content `div` should have as children only the tags `h2`–`h6`, `p`, `ul`, and, if necessary, other semantic elements such as `section` or `article`. I know, this doesn't seem so exciting. However, using CSS and maybe JavaScript libraries prepared for smartphones, we can take this simple markup and create great experiences for high-end devices. Using a simple document structure will be one of our best practices in the mobile world, to avoid duplication.

Navigation Link Menus

XHTML MP 1.2 recommends the usage of linked resources for navigation purposes. A navigation link menu is a series of `link` tags, generally defined in the `head` element, that refers to the main index file (the home page) and optionally the next and previous pages in a series of related documents. These links can be useful for indexing and search engine optimization purposes. While almost no browser today actually uses this information, it's useful for SEO purposes (IE10 has a similar behavior called jump lists). Here is an example of a navigation link menu for a photo gallery showing photo #2:

```
<link rel="start" href="index.html" />
<link rel="next" href="photo3.html" />
<link rel="prev" href="photo1.html" />
```

Go to top

Some mobile browsers, like Safari on iOS, allow the user to tap with a finger in the top section of the screen to scroll the page to the top. Other browsers have keyboard shortcuts for that. And many others don't have any such mechanism, or if they do, it's so obscure that most users probably don't know what it is. So, it is a good mobile web practice to insert an anchor at the top of the page (in the header) and a link to that anchor at the bottom:

```
<body>
   <header>
      <a name="top"></a>
      <h1>Mobile Web</h1>
   </header>
      ...
   <footer>
      <ul>
          <li><a href="#top">Go to Top</a></li>
          <li><a href="/">Go Home</a></li>
      </ul>
   </footer>
</body>
```

HTML5 Mobile Boilerplate

Wikipedia (accessed September 2012) defines boilerplate as follows:

> [B]oilerplate is the term used to describe sections of code that have to be included in many places with little or no alteration. It is more often used when referring to languages which are considered verbose, i.e. the programmer must write a lot of code for minimal functionality. The need for boilerplate can be reduced through high-level mechanisms.

When creating a new website, it is helpful to have a template (boilerplate) with all the well-known hacks, solutions, and best practices already implemented to use as a starting point, rather than having to start from scratch. HTML5 boilerplate (*http://html5boilerplate.com/mobile*) is starting to emerge: a mobile-specific version is maintained by the community as an open source project.

The latest version at the time of this writing includes many of the techniques that we have covered in this chapter, and you can of course add your own code to it. The template includes:

- Meta tags, such as viewport and icon references
- *normalize.css*, an alternative to CSS reset (a way to normalize defaults on different browsers)
- Some JavaScript libraries, such as Zepto.js and Modernizr (two libraries we will cover later in this book)
- A JavaScript helper class useful for some JavaScript-based hacks, such as the iOS viewport bug fix
- High-performance and mobile-specific configuration files for Apache

The Content

If we focus our content semantically, we will not have big problems adapting our design to different mobile scenarios. By "semantically," I mean using the well-known paragraph, structure, and list HTML5 elements. Be sure to understand and make proper use of every HTML element, such as `header`, `footer`, `article`, `abbr`, and `address`.

Block Elements

If we define a viewport allowing user scaling, most browsers support a way to zoom into a block element (such as a paragraph), using a double-tap gesture, for example. To make this work properly we should not use paragraphs that are forced to be wider than the screen (because we have defined the `width` and the `font-size` as fixed); this will force the browser to zoom out and the paragraph will be unreadable.

Lists

Using standard lists will help us a lot in defining our designs later with CSS and for semantic search engine optimization. For the mobile web, we should use the following list types:

Ordered lists (`ol` tag)
: For navigation link menus

Unordered lists (`ul` tag)
: To present lists of similar objects

Definition lists (`dl` tag)
: To show key/value details

The last one is perhaps the lesser-known list tag in web development. For example, if we are showing a product detail page, in many browsers it's better to use a definition list rather than a table for attributes:

```
<h2>iPhone 5</h2>
<dl>
   <dt>Price</dt>
   <dd>300 EUR</dd>
   <dt>Memory</dt>
   <dd>32Gb</dd>
   <dt>Network</dt>
   <dd>LTE, 3G, WiFi, Bluetooth</dd>
</dl>
```

The `dt` tag is used for the key (definition term) and the `dd` tag for the value (definition description). This is very useful, semantically correct, and clearer than using a table. Later, with CSS, we can rearrange the elements.

Tables

Repeat after me: "I will not use tables for document layout." Write it with a red marker on your bedroom ceiling, if it will help you remember. Using tables for document layout is bad in desktop web development. It is hell for the mobile web.

Mobile browsing is more a one-column experience, unless we are working with tablets. If you do want or need to use a table, you should limit it to at most five columns of tabular data (preferably with short column headings and data values).

We can define the table title (`caption`), the header (`thead`), the body (`tbody`), the footer (`tfoot`), and the columns (`colgroup`, `col`), and finally the rows (`tr`), the header cells (`th`), and the data cells (`td`). Cells can be merged using the `rowspan` and `colspan` attributes, and the design should be defined in CSS.



On some older devices and basic browsers for feature phones, sometimes the only way to add different elements in a row is to design with tables, as floating elements are not available.

Frames

Frames are one of the "better if you avoid it" features in the mobile world. I remember, back in 1997 (the Microsoft FrontPage era), being happy with the frames technique, creating fixed menu bars and dealing with links between frames. It was a happy time, until search crawlers came into action and frames became the worst thing you could ever do in a website. OK, background music can be even worse, but it is true that today the usage of frames is suitable only for intranet sites and noncrawled applications. Similar functionality can now be provided with the much more versatile Ajax.

The HTML frames mechanism allows the developer to split a document into *n* subdocuments, vertically and/or horizontally. Every frame is a different document (that's the problem for search engine spiders), and every frame manages its own scrolling (this is the problem for the mobile world).

We already know that in the mobile world, even though the viewport can be large, the screen is small. Splitting this small screen into smaller windows as frames can be difficult. The biggest problem is scrolling. As mobile browsers don't show visible scroll bars, the user has no clear indication where a frame or iframe exists to scroll inside. Some mobile browsers will not even allow the user to scroll inside a frame or iframe.

So, my final advice is: don't use frames when developing for the mobile web.



The inline frame (or *iframe*) is a modern way to do frames (although it was introduced by Internet Explorer in 1997). The `iframe` tag produces better results than frames. Today, iframes are often used by ad servers to serve advertisements from a third-party server. If you can, it's still best to avoid them; if you can't, offer content that doesn't need to be scrolled inside the iframe.

i-mode XHTML

Japanese devices from NTT DoCoMo used to have their own version of XHTML with extensions for XHTML and CSS, based on the old cHTML. Serving i-mode XHTML files requires using a new DOCTYPE and defining the charset UTF-8 or Shift-JIS for Japanese characters. For example, for the latest version, the first two lines of the file might look like this:

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE html PUBLIC "-//i-mode group (ja)//DTD XHTML i-XHTML(Locale/
Ver.=ja/2.3) 1.0//EN" "i-xhtml_4ja_10.dtd">
```

The latest version of i-mode XHTML at the time of this writing is 2.3; there is also a non-XML version, called i-mode HTML, which is currently at version 7.2. i-mode Browser 2.0 is the browser delivered by devices manufactured after May 2009; it includes the latest versions of both i-mode HTML and i-mode XHTML.

There is an excellent portal of information (*http://bit.ly/ZpVJUJ*) in English for versions 1.0 and 2.0 of the i-mode Browser.

The good news is that the latest versions of i-mode HTML and i-mode XHTML support almost every tag used in XHTML MP. The list of supported attributes differs, though, and a lot of new attributes and values are available as i-mode extensions. For example, an `li` element can have a `type` attribute with `circle` as the value, and a numeric text input can be defined with `type="text"` and `istyle="3"`.

Links

Hyperlinks are the heart of the Web, and this holds for the mobile web, too. You might think there isn't much to say about links, but that's not the case.

Every link in a mobile website should have the well-known `href` attribute, set to the URL of the desired resource, and the most important links on the page can have an `accesskey` attribute assigned for easy access via keyboard shortcuts, on devices that support access keys. The `target` attribute should be avoided for feature or social phones because those browsers don't have tab or multipage support.



Some devices support the usage of `tabindex` for focusable elements (links, form controls) to change the element order for browsers with focus-based navigation. However, changing the natural order of tabbing is discouraged, unless you have a difficult design and you want to improve the user experience.

If you are making a link to the desktop version of your website (a must-have, as discussed earlier), use `rel="alternate"` to specify that the link is to the same page in an alternative format.



In devices that support focus-based navigation (most low-end devices, and even some touch devices with a touchpad or scroll wheel, like the Nexus One and some BlackBerrys), it is important to define whole clickable zones. For example, if you want to make a title and description both clickable, use one link tag for both elements instead of two separate links to the same page. A single focus border will appear around the whole area you want to be clickable.

New windows

Some browsers accept the `target="_blank"` attribute, but depending on the browser the behavior is different. Some browsers simply open the URL in the same window, others create a new tab or window (allowing the user to browse between them), and still others open the new URL as a modal pop-up, in which case the user cannot go back to the first page until he closes the new one.

On tablet-based browsers, usually we have a tab navigation user interface, as we are used to in desktop browsers.



In Chapter 15, we are going to analyze how we can integrate with device services through different URIs using links.

Navigation lists

A navigation list is any list of links that are related in some manner and listed one after the other. The recommended way to create such a list is with `ol` or `ul` tags. With the ordered list, the number of the key to press to access each option is printed for us for free, but we still need to add the `accesskey` attributes to the `a` tags by hand:

```
<ol>
   <li><a href="option1.html" accesskey="1"></li>
   <li><a href="option2.html" accesskey="2"></li>
</ol>
```

Remember to use the nav container that is available in HTML5 to clearly mark navigational areas on the screen.

Accessibility

The accessibility of a mobile website should be as important as the graphic design, the performance, or the user experience. Some mobile operating systems, such as iOS, support accessibility features such as screen voice readers, gesture helpers, or contrast color utilities.

If your mobile website or app is *accessible* it means that anyone can use it, with any device. Here's some good advice for mobile web solutions:

- Leave zooming capabilities in place at all times.
- Provide your content using progressive enhancement, so it will work properly without any CSS enhancement or JavaScript layer.
- Verify that the website can be browsed with focus-based navigation.
- Use semantic HTML, not just `div`s and `span`s.
- Use HTML5 WAI-ARIA attributes.
- Provide alternate content when necessary, such as `alt` attributes for images or alternate information for `video` or `canvas` elements.
- Browse your website with accessibility tools enabled.



If you want to know more about mobile web accessibility, you can check out the W3C Mobile Accessibility website (*http://www.w3.org/WAI/mobile/*).

WAI-ARIA

Accessible Rich Internet Applications (*http://www.w3.org/TR/wai-aria*) (WAI-ARIA) is a W3C standard for accessibility that is compatible with some HTML5 mobile browsers. The main goal is to make accessible, rich pages that are not just plain documents but services and applications.

A compatible browser will understand a `role` attribute that can be applied to any HTML element, such as a `div` or `section`. The role definition allows the browser to understand your rich content, indicating whether the element is a navigation area, a slider, or some other rich element. The available roles are listed at www.w3.org/TR/wai-aria/roles.

For example, we can explicitly define that `aul` is a toolbar with `role="toolbar"`:

```
<ul role="toolbar">
</ul>
```

A screen reader, for example, will then understand that our `ul` is a toolbar and will react appropriately, such as by inviting the user to select one of the `li`s. WAI-ARIA also adds new `aria-*` attributes to define more specific metadata information, such as `aria-pressed`, which indicates if a push button is pressed or not.



Screen readers provide spoken feedback on what is touched and selected for blind and low-vision users. Semantic content and WAI-ARIA will help the reader to understand what the content really is. You can try it on your own smartphones and tablets on some operating systems.

If you want to know more about accessibility, check out the *Accessibility HandBook* by Katie Cunningham (O'Reilly).

Accessibility testing

If you want to test how accessible your mobile website is, you can:

- Browse your website from a focus-only device, such as a feature phone or a non-touch Kindle ereader.
- Enable VoiceOver on iOS, a screen reader available from Settings→General→Accessibility→VoiceOver.
- Enable TalkBack on Android, a screen reader available from Settings→System→Accessibility→Services in conjunction with the "Enhance web accessibility" option in 4.1.
- Search the application stores for accessible browsers and use them to view your own content.