The Mobile Web Handbook

Published 2014 by Smashing Magazine GmbH, Freiburg, Germany.
Printed in the EU. ISBN: 978-3-94454093-1.

Cover Design, Illustrations and Layout by Stephen Hay.
Copyeditor and Proofreader: Owen Gregory.
Editing and Quality Control: Vitaly Friedman.
eBook Production: Cosima Mielke.
Typesetting: Markus Seyfferth.

The Mobile Web Handbook was written by Peter-Paul Koch
and reviewed by Stephanie and Bryan Rieger and Vasilis van Gemert.

Links and updates of this book can be found at
http://quirksmode.org/mobilewebhandbook.

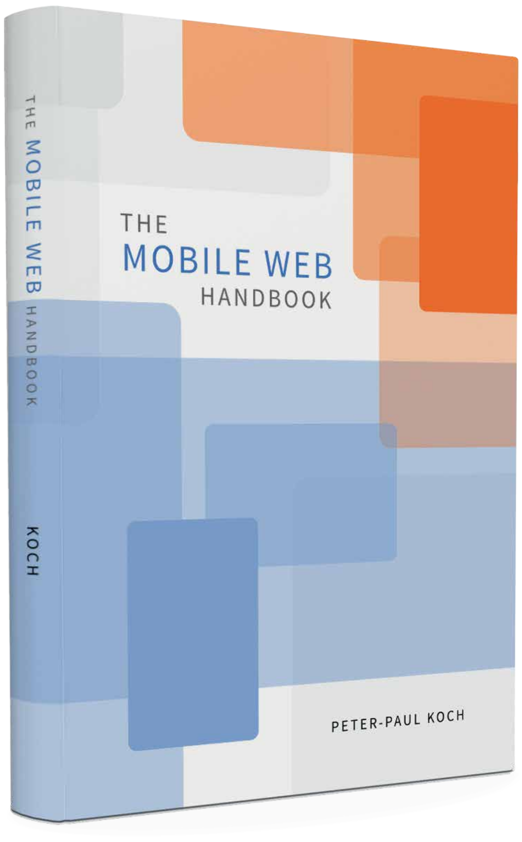

THE MOBILE WEB HANDBOOK
PETER-PAUL KOCH
THE MOBILE WEB HANDBOOK KOCH

Table of Contents

Introduction

Introduction

The Mobile Web Handbook explores the differences between mobile and desktop web development that we should be aware of when creating websites for both. It's not very technical — there are only a dozen or so simple code examples. It discusses no libraries or tools. It's about mobile web fundamentals.

There is no mobile web distinct from the desktop web. Developing websites for mobile is pretty much the same as developing for desktop, especially now that responsive design techniques allow us to adapt our CSS layouts to both huge desktop screens and tiny mobile ones.

Still, there's "The Mobile Web" in the title of this book, and that's not an oversight or marketing trick. It serves as a convenient shorthand for "touch-based small-screen web on more browsers than you've ever heard of." Mobile web development is not fundamentally different from desktop, but there are subtle distinctions that may cause you trouble if you're unaware of them.

It's best to see mobile web development as a layer that you apply on top of regular web development, and which contains a few new concepts and techniques that you must understand in order to create compelling mobile experiences. This book concentrates on that mobile layer, and highlights three topics:

1. On desktop we have only five browsers, but on mobile it's more like 20 or 30. These are not all separate browsers: many of them are subtly different versions of the same browser, especially of Android WebKit. Why is that? How do you handle it? Why is Android so complicated? How will the mobile browser market develop?

2. On desktop, there's only one single viewport: the browser window. On mobile, this viewport was split into two, and a third viewport was added. Why do we need three viewports? How do they work?

3. Desktop has its keyboard and mouse events, and touchscreen browsers need special JavaScript events to react to the user's touch actions. This may seem logical but Microsoft, of all companies, challenged that logic and raised interesting philosophical questions about the relationship between JavaScript events and interaction modes. On a practical level, the touch events have some special features that you need to know about.

Browsers, viewports, and touch events are the main themes of this book. There are also a few smaller items: the rise and fall of browsers and operating systems; what proxy browsers are; why a few CSS declarations such as `position: fixed` are more difficult to get right on mobile than on desktop; and becoming an accomplished mobile web developer by setting up a device lab and reconsidering outdated development practices. As a bonus, you will learn *why* responsive design works. (Not *how*. You already know how. But do you know *why*?)

So here we go. It's going to be quite a journey.

What This Book Doesn't Cover

In order to manage your expectations, here are a few topics that are not treated in this book. This is about the mobile web, so there is no information on native apps. You can use this book for creating hybrid apps (that is, apps written in HTML, CSS, and JavaScript but wrapped in native code), but only for the web component, not for the native one.

I'm not a designer, so I don't say anything about design except for some very vague general tips. No design patterns, either.

The mobile market is very volatile, and browsers and devices that are a hit now could be a memory in a year's time. That's why I try to steer clear of inspecting individual devices and browsers, though sometimes I make an exception for Safari on iOS because it's so very influential on web development thinking.

Finally, the most complicated caveat: this book only investigates fundamental differences between desktop and mobile, and generally ignores topics such as AppCache, which, though more important on mobile than on desktop, are not unique to mobile. This is sometimes a subtle distinction, but it helped me a lot in keeping the scope of this book, and of my research, to manageable levels.

Companion Site

Writing a book about the mobile web is challenging because it's one of the fastest-changing environments ever — faster by far than the traditional desktop web. I write this in summer 2014, and by the time you read it things will have changed. That's why I try to concentrate on fundamental issues and problems, and don't pay too much attention to quick-shifting details such as browser bugs.

Still, you need to know about the bugs as well. That's why I created a companion site at http://quirksmode.org/mobilewebhandbook that contains links to my browser research to back up what's in this book — or, as time progresses, to show which mobile browsers have mended the errors of their ways, or changed, or done something else of note.

In this book I occasionally give browser compatibility notes, but more often I'm rather vague; for instance, saying that "many browsers" support this or that. The companion site always gives a breakdown of those browsers, and includes notes on bugs.

Tablets

The Mobile Web Handbook focuses on mobile devices; that is, small devices that fit in the palm of your hand and can establish a connection over a mobile network. It does not really cover tablets or other types of devices.

Still, a lot that's in the Handbook also applies to tablets. Tablets, too, have touch-based browsers, and although they have larger screens than mobile phones, they're still smaller than most desktop screens and have three viewports instead of one.

Besides, what exactly is a tablet? Samsung, in particular, tends to bring out more and more very large phones, which you can easily see as small tablets instead. The Microsoft Surface is a tablet with an attachable keyboard, which converts it more or less into a laptop computer.

Is the Samsung Galaxy Note 8.0, released in Q2 2014, a huge phone or a mini tablet? Or is the distinction meaningless?

Right now we can't tell if tablets are going to remain a separate device category, or if they'll quietly fold into the phone and laptop categories. From a technical perspective it doesn't really matter, though. Tablet browsers are mobile browsers in all respects, and obey the same rules and restrictions. Although this book will hardly mention tablets again, you can safely assume that anything you build for mobile will work on a tablet as well, with the obvious caveat that a tablet screen is bigger than a phone screen and your responsive design should accommodate that.

Thank Yous

This book didn't spring from my forehead fully formed. Plenty of people were involved, and I'd like to thank all of them. Vitaly Friedman saw the potential of this book, signed me up, and was the general editor for all chapters. Markus Seyfferth arranged all practical matters such as contracts and printing. Stephanie Rieger was good enough to be the technical editor for all chapters. Stephen Hay signed on for the cover, illustrations, and overall book design. Patrick Lauke edited the Touch and Pointer Events chapter, a topic he knows more about than most other web developers I know combined. Max Firtman went over the Browsers and Android chapters and provided valuable feedback.

Then a compelling presentation by Jason Grigsby and a discussion with the MSIE team caused me to overhaul the Touch and Pointer Events chapter once more. Finally, Vasilis van Gemert read through the entire second draft from the perspective of a teacher, while Owen Gregory signed up for those last finicky copy edits that make a good book a great one. Thank you all, ladies and gentlemen. The book wouldn't have been as good as it is now without your timely help. All remaining errors are, unfortunately, my own.

***

Now let's get started with a general overview of the mobile world. You'll find that it differs a lot from the desktop world we're used to.

Chapter 2

Browsers

Chapter 2

Browsers

If you're used to the simple five-browser ecosystem that exists on the desktop, you're in for a surprise in the mobile market. So far, I have identified about 30 mobile browsers, ranging from lousy to great. Not all of these browsers are equally important: in fact, about 20 of them are somewhat marginal. And just like on desktop, there may be differences between two versions of the same browser.

The Google browsers, Android WebKit and Chrome, come in several flavors, and each flavor may have several versions. In fact, the Android browser situation is so complicated that I'm going to give it an entire chapter of its own. The current chapter mostly ignores Android and instead talks about the other platforms, in particular iOS, as well as some general principles.

You will find no compatibility information here: by the time the book is printed, it would be outdated. You should turn to the companion site at http://quirksmode.org/mobilewebhandbook for details on the differences between the browsers.

Browser Types

There are four browser types on mobile: default browsers, downloadable browsers, proxy browsers, and WebViews. These categories overlap in places: a browser does not necessarily belong to just one category. For instance, the proxy browser Opera Mini is downloaded by many users, but is the default browser on some feature phones.

Default Browsers

Every phone has a default browser; that is, a browser that's part of the firmware, usually developed by the OS vendor. Thus Safari, developed by Apple, is the default browser for iOS; and IE, developed by Microsoft, is the default browser for Windows Phone. The table below summarizes the default browsers of the platforms.

Platform	Default browsers	Remarks
iOS	Safari	
Android	Android WebKit or Chrome	Several flavors of both (see next chapter)
BlackBerry	BlackBerry WebKit	
Windows Phone	IE	
Symbian	Symbian WebKit	
Firefox OS	Firefox	
Sailfish	no name yet	Gecko-based
S40	S40 WebKit on older ones; Xpress on Asha.	Xpress is a Gecko-based proxy browser
Other feature phones	Varies: Opera Mini, NetFront, UC Mini, or others	Opera Mini and UC Mini are proxy browsers.

Sales market shares of mobile OSs in 2012 and 2013

Most default browsers are tightly integrated with the underlying OS, to the point where it is not possible to update the browser separately. Thus, in order to get a new Safari version you have to update iOS; the same goes for IE and Windows Phone. This causes default browsers to develop more slowly than other types of browsers, which could mean in future we have to go through another period where one old, bad default browser holds back the entire mobile web, just as IE6 held back the desktop web. Fingers crossed.

Incidentally, device vendors frequently refuse to give their default browsers names. That's why I use the unimaginative but fairly clear "[Platform] WebKit" when necessary, and my compatibility tables are riddled with Android WebKit, BlackBerry WebKit, Symbian WebKit, and more.

Downloadable Browsers

There are a lot of browsers users can download and install for themselves. Opera, Firefox, Chrome, and UC are a few important ones. In practice, this is only possible on Android, since installing other rendering engines is not allowed on iOS, and no vendor has yet produced a downloadable browser for the small platforms.

One advantage downloadable browsers have over default browsers is that it's possible to update them whenever a new version is available. The latest and greatest features usually land in downloadable browsers first, which is one of the reason web developers tend to like Chrome, Opera, and Firefox. We web developers are not like regular consumers in that respect, though.

It appears that there is a difference between the Western developed nations and the developed nations of east Asia. In the West, few consumers bother to install a different browser — or even know it's possible. In Asia, consumers do download alternative browsers, such as UC or QQ in China, and Puffin in Korea. A common reason is that these browsers offer better integration with local social networks. Asian browser statistics often show downloadable browsers that rarely occur in the West.

What's the point of creating a downloadable browser? The answer is a combination of becoming or staying relevant on mobile, and making money. These two goals are connected: the more relevant you are, the more money you make. Browsers want more market share, and the best way of getting that is to be included as a default browser on some device or another. Before it comes to that, though, these browsers have to show their worth by making a free version available for anyone to try. We'll get back to making money with browsers later in this chapter.

WebViews

A WebView is an OS's browsing interface for native apps. For instance, a Twitter client may call on the platform's WebView to show a webpage when the user clicks on a link in their feed. A game's help pages may be webpages, in which case the game app uses the platform's WebView to display them.

Apple doesn't allow the installation of other rendering engines on iOS devices. Therefore, other browsers wanting to move to iOS are forced to use Apple's WebView. This goes for Chrome on iOS, and also for Opera Coast.

In general, WebViews are separate programs that use many low-level components (such as rendering engines) of the default browser, but may differ in other respects. Testing on WebViews may therefore be a good idea if you expect your pages to run in them.

Proxy Browsers

Then there are the proxy browsers. Their rendering engines, responsible for parsing and executing HTML, CSS, and JavaScript, are found not on the device but on a remote server. They do this to save their users money.

The opposite of a proxy browser is a full browser, and it works as we expect a browser to. When the user requests a page, the browser fetches the HTML, CSS, JavaScript and other assets via HTTP, and once it has everything, it renders and shows the page. All of these steps take place on the client, and take up memory, processor time, and battery life.

Proxy browsers are different:

1. The user requests a page. They send not a normal HTTP request, but a special request to a special proxy server over an encrypted connection.

2. This proxy server makes the normal HTTP request to the web server the user wants to access. It requests the HTML as well as all assets, such as CSS, JavaScript, images and so on.

3. The proxy server contains a rendering engine, which renders the page as usual.

4. The proxy server then compresses the rendered page into a kind of image of the website: think of it as a PDF or an image map. It has hotspots for links, and the user can also select text and zoom a bit.

5. The proxy server sends this file to the client, again over an encrypted connection.

6. The client shows the file to the user. If the user taps on links or does something that requires code execution, the process is repeated.

Opera took the lead in the proxy browsing world primarily because it was the first to see the opportunities and enter the market. Nowadays, though, serious competition is available. There are three important proxy browsers:

1. Opera Mini: used throughout the world, especially in developing countries on low-end devices. Based on Presto at the time of writing, although Opera will eventually switch to Blink.

2. UC Mini: used mainly in China but branching out powerfully across the world. This browser will become more important as time goes by. Based on Gecko.

3. Nokia Xpress: the default browser for Nokia's Asha (S40) low-end phones, and also available for Windows Phone. Based on Gecko. Now that the Asha line is discontinued by Microsoft, it will gradually lose its market share.

Advantage: Cheap

Proxy browsers primarily serve to save the user money. Because all the proxy client has to do is show static files, allow for clicks or taps on links, and generate a simple UI, it's fairly light and able to run even on low-spec phones. Users do not have to buy an expensive smartphone in order to access the web.

Besides, all the client receives is a highly compressed file, which is much lighter than raw HTML, CSS, JavaScript and image files, and it uses only a single request and response. This saves a lot of mobile data traffic — Opera claims up to 90%. Also, this will work even on older networks, which is important to developing-world operators that don't want to spend money on upgrading their entire network.

Thus, proxy browsers serve to make the web accessible even to low-income users who can't afford a desktop computer or a smartphone. Unsurprisingly, they're especially popular in the developing world, while being marginal in developed countries. Still, even affluent smart-phone users on excellent connections will notice a distinct increase in speed when they switch to a proxy browser.

Disadvantage: No Client-Side Interactivity

There's a disadvantage to proxy browsing, too: no client-side interactivity. Proxy browsers support JavaScript, but every time the user causes a JavaScript event (by clicking on an Ajax link or something similar), the client sends a request back to the server for instructions. The server executes the script, fetches new assets if necessary and sends back the updated page, which, as far as the client is concerned, is a completely new page.

It's important to realize that this lack of client-side interactivity is a feature, and not a bug. By giving up client-side interactivity, users save themselves a lot of money. Executing JavaScript costs users money, and some prefer not to pay the price.

Working with Proxy Browsers

You must learn to work with proxy browsers. Download Opera Mini to your iOS or Android device now and start testing in it. A proxy browser doesn't quite work like the browsers you're accustomed to, and many users will get their first taste of the web via a proxy browser. Having at least some experience with them is mandatory.

The problem is not in the HTML or CSS — they work pretty much as you'd expect. It's in the JavaScript that you'll encounter the most serious problems. Any time a proxy browser encounters anything dynamic, it has to go back to the server and ask for new instructions. Thus, there's always a lag of a second or more between activation and execution.

Although proxy browsers support JavaScript, most of them disallow certain events. For instance, if you have an `onscroll` event handler, it should fire whenever the user scrolls. But in a proxy browser, that would mean making a server request with every few pixels of scrolling, which would make the page completely unusable. Therefore, proxy browsers disable the `scroll` event. The same goes for the mouse and touch events.

As a rule of thumb, assume that only events that clearly show the user's intent to load new data will work in proxy browsers. In addition, `mouseover` is widely supported because so many websites depend on it, and load and unload because they will be processed on the server anyway. You can expect `click`, `change`, `focus`, `submit` and the like to work, but `mouseout`, the touch events, the key events, `resize` and `scroll` will not work.

I advise you to keep it simple and concentrate on the `click` event, which always works everywhere. Add `submit` if you're working with forms. That's it, though — do not expect other events to work on proxy browsers.

Hybrid Browsers

Since saving bandwidth is such an obviously excellent idea on mobile, the true proxy browsers have been joined by hybrid browsers: browsers that can function either as full or as proxy browsers. In most of them you can switch bandwidth saving on and off. They include Amazon Silk, Puffin, Opera Mobile, and Chrome. Unfortunately the details of their hybrid behavior vary a lot, and it's hard to give general rules.

Exactly how hybrid proxy browsers divide up the work between client and server depends on the browser and the settings. See the Silk description at http://smashed.by/silk; the Chrome data compression proxy description at http://smashed.by/data-compression; and for more information on Opera Turbo http://smashed.by/turbo. I have not been able to locate similar instructions for Puffin.

The iOS Browser Situation

Now that we know the various browser types, we can understand the iOS browser situation. Remember the crucial fact: Apple does not allow the installation of another rendering engine.

1. The iOS default browser is Safari. Duh.

2. In addition, iOS has a WebView for native apps that need it. Up to and including iOS7 it was slightly different from Safari, but at the time of writing the promise is that these differences will disappear in iOS8.

3. Chrome on iOS may not install its Blink rendering engine, and is therefore forced to use the Apple WebView. The same goes for Opera Coast.

4. Opera Mini, however, neatly evades Apple's restrictions because its rendering engine resides on a server. Installing the Opera Mini client is allowed, and therefore this browser is available on iOS.

In other words, the only non-Safari iOS browsers that it makes sense to test in are the proxy browsers. At the time of writing there's no other proxy browser for iOS but Opera Mini, but that might change.

In particular, Chrome on iOS tests are relatively useless. Although the Chrome app offers you integration with your Google account, when it comes to actually rendering webpages it must use Apple's WebView. Thus, although you can test on Chrome for iOS if you feel like it, this does not tell you anything about the real Chrome on Android, which is a completely different browser.

The Browser Situation On Other Platforms

The other platforms are even simpler to understand than iOS. They have their own default browsers, and usually Opera Mini is also available. Although in general the installation of other rendering engines is allowed, no vendor has yet decided to build a new browser for BlackBerry, Windows Phone, or any of the others.

Rendering Engines

Every browser has a rendering engine that is responsible for the interpretation of HTML, CSS, and the DOM parts of JavaScript. Just like on desktop, there are four important rendering engines on mobile: Gecko, Trident, WebKit, and Blink. In addition, Opera's old Presto engine lives on in Opera Mini for now.

Until about 2010 BlackBerry, NetFront, UC, and a few other browsers had their own proprietary rendering engines, but with the advent of mobile browsing as core platform functionality it became clear that these engines were inferior to the desktop ones, especially in JavaScript and performance. Therefore all proprietary mobile rendering engines were replaced by desktop ones.

Most browser vendors decided to use WebKit. Trident and Presto, back when it existed, were proprietary, and so not an option. As for Gecko, its use beyond Firefox is restricted to UC Mini and several Nokia-descended browsers. The lack of adoption is probably caused by the fact that back in 2009, when most vendors took these decisions, Gecko was still far too heavy for mobile processors and memory constraints.

Meanwhile Mozilla has streamlined its engine in order to create Firefox Mobile, but that change came too late to profit from the initial wave of rendering engine replacements.

Google forked Blink from WebKit in 2013, when the wave of replacements was over. Nowadays it's becoming an option for Android vendors. We'll go into that in the next chapter.

There Is No WebKit on Mobile

So many mobile browsers use WebKit as their rendering engine that it's more efficient to list the ones that do not:

- IE Mobile uses Trident.
- Opera Mini uses Presto, but will eventually replace it with Blink.
- The Chrome browsers use Blink. We'll get back to them in the next chapter.
- Firefox Mobile and Firefox OS use Gecko.
- UC Mini, Nokia Xpress, and the default browser on the Sailfish OS by Jolla also use Gecko.

Any browser not mentioned above uses WebKit. At first sight, the fact that so many browsers use WebKit seems like a powerful aid to web developers. Unfortunately, if a browser uses WebKit it does not mean it's the same as any other WebKit-based browser. In fact, there are considerable differences between them.

WebKit is a rendering engine, not a browser. If you hand it HTML, CSS, JavaScript, and images, it will deliver a rendered page. However, it does not contain the modules necessary to request the assets or to actually show the rendered page on the phone's screen. It depends on the OS for interfacing with the keyboard, mouse, and touchscreen. Platform owners have to provide all these functionalities.

WebKit provides support for hardware-accelerated animations but does not contain the modules that communicate with the GPU and that make sure that hardware animations actually show up on the screen. If you want modern form fields such as `<input type="date">`, you must write the date interface yourself. WebKit includes Apple's JavaScriptCore as the default JavaScript engine, but you may decide to switch to another, such as Google's V8. Finally, you may use a different WebKit version than the other guy, but even if you don't, two browsers that both use WebKit 537 may be quite different.

So, there is no WebKit on mobile. A lot of browsers use more or less the same rendering engine but differ a lot in their details. Testing your website in all individual WebKit-based browsers is best. If it works in Safari for iOS, it will not necessarily work in BlackBerry WebKit, or Android WebKit, or Obigo, or Symbian WebKit, or Dolphin for Android, or… well, you get the point.

Making Money From A Browser

Why do people make browsers? There are two fundamental reasons: providing your platform with one, and making money. Any smartphone needs a browser. Therefore Apple, Google, Microsoft, Samsung, BlackBerry and others must provide one. Simple.

However, other vendors want to make money with their browsers — even if only enough to pay their engineers. There are three business models for making money from browsers:

1. Selling your company (and browser).
2. Selling licenses for your browser.
3. Search engine deals.

When I started on mobile back in 2009, I tested all the downloadable browsers I could find. Most of them were pretty crappy, but there was one notable exception: the Iris browser for Windows Mobile created by a small Canadian company called Torch Mobile. Several months later BlackBerry acquired the company to build a new WebKit-based browser for its platform, netting the founders and engineers a nice bit of money.

This doesn't happen very often. I had the feeling that back in 2012 the small Californian Dolphin browser groomed itself for acquisition by Facebook, but nothing came of it. Not all that many companies are interested in buying a browser, it seems, and the ones that are have already done so.

Selling licences is a more forward-thinking business model. Opera, especially, makes money from Mini licenses sold to operators, mostly for use on feature phones without a good default browser. The operator gets a customized Opera Mini build for their devices without the Opera logo. This is good for the operators, since browsing users spend more money, but the operators don't have to spend money on creating their own browsers. I assume UC has similar deals in place.

Finally, all browsers have deals with all search engines in which they get a small fee every time a browser user uses the search engine. These deals are shrouded in mystery. The fact that they exist is well known, but the details, especially the financial ones, are secret and will likely remain so. All browsers do it: it's the easiest way for a browser vendor to make money. The search engine deals are not restricted to downloadable browsers — default browsers do the same, both on desktop and on mobile. The deals are more vital for downloadable browsers, though, which usually don't have other sources of income or the backing of a wealthy corporation.

The search engine deals become more valuable as more people use your browser. It's in the interest of downloadable browser vendors to encourage as many people as possible to use them. Whether they will succeed is an open question.

Statistics

It's time to take a look at statistics again. The best browser market share stats are the ones that come from your client's log files. Study them to find out what kinds of phones are used to visit their website.

Be aware that users of some browsers might not be able to use the website and so might be underrepresented. I usually look at statistics for the homepage or another important landing page and compare them with a few other pages. If a certain mobile browser is visiting the homepage in decent numbers but is nowhere to be seen elsewhere on the website, users of that browser are likely encountering a problem that you must solve.

Finding and using general worldwide mobile browser market shares is fairly hard. What we need are the mobile browser statistics of a first-rank website such as Google or Yahoo. Unfortunately, these companies keep their statistics a secret. As we saw, search engine vendors pay browser vendors a small commission for every query they send, and they want to hide these vital statistics from their competitors (and from browser vendors). That's why they do not share the browser make-up of their homepage hits.

So, we're reduced to using analytics services that gather these statistics from their clients and share them freely. Unfortunately, these services have a self-selecting bias because site owners (or web designers) have to sign up for them and install a counter script. Thus, even though these services present global data, it comes from a specific subset of websites. I encourage you to sign up the sites you make to one of these services and make the data a little more representative.

The choice is yours, then: either use the statistics, knowing they're incomplete and biased; or use none at all. To me, any data is better than no data, but your mileage may vary. At the time of writing, I know of three such services, and I encourage you to compare them.

- StatCounter (http://smashed.by/statcount)
- NetMarketShare (http://smashed.by/netms)
- Akamai (http://smashed.by/akamai)

Personally, I prefer StatCounter because NetMarketShare puts tablets and mobile devices in one category, and at the time of writing Akamai's "New Features", which comprise most of the mobile data, have serious and persistent interface problems. (You should try it, though. Maybe the problems have been solved by the time you read this.) So I used StatCounter for the data below.

Don't stare yourself blind on tiny differences that are statistically meaningless. What you're after with global stats is the broad picture of who wins and who loses. Chrome is a clear winner (but see the next chapter), while BlackBerry, Nokia, and Opera lose.

Browser	Q2 2014	Q2 2013	Q2 2012
Android WebKit	25%	30%	22%
Safari	23%	26%	24%
Chrome	18%	3%	-
Opera	12%	16%	22%
UC	10%	9%	8%
Nokia	4%	7%	11%
BlackBerry	2%	3%	5%
NetFront	2%	2%	4%
IE	2%	1%	1%
Other	2%	3%	3%

StatCounter: Global mobile browser stats, Q2 of three years

One note: what is "Opera"? Opera Mini, or the full Opera Mobile browser? Unfortunately, StatCounter does not give this information. I assume that 99% consists of Opera Mini, because that would align well with the fact that Opera is mostly present in developing countries, but that's a guess on my part and I may be wrong.

Still, all this information doesn't tell you which browsers will visit your client's site. If you don't have specific stats available, take a look at the stats for your country. They can be very, very different from the global stats. See the next table, for instance.

Browser	US	UK	India	Brazil
Android	21%	17%	12%	31%
Safari	50%	46%	1%	14%
Chrome	21%	19%	4%	37%
Opera	1%	4%	25%	6%
UC	2%	1%	34%	1%
Nokia	-	-	10%	3%
BlackBerry	1%	8%	-	-
NetFront	-	-	7%	-
IE	2%	3%	1%	4%
Firefox	-	-	-	1%
Other	2%	2%	6%	3%

StatCounter: Mobile browser stats of four countries, Q2 2014

Can you spot the differences? Safari rules in the developed West, but not elsewhere. BlackBerry is wiped out, except in the UK. UC is the largest browser in India, while NetFront also retains part of the market. IE and Chrome are more successful in Brazil than in other countries.

As you can see, there is no global mobile browser market — just a collection of local ones.

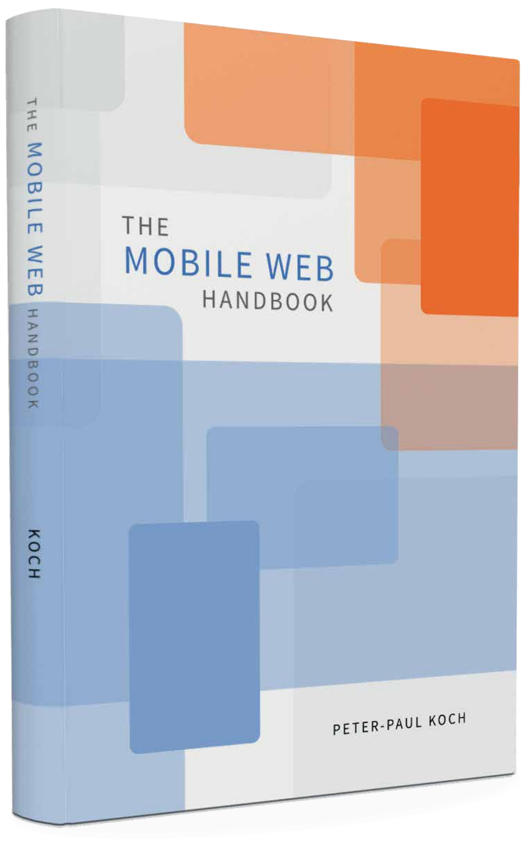
Although your country's stats are much more useful than global ones, there might still be factors affecting your site that influence the exact browser make-up. But if you don't have stats for that site, you're forced to use country stats instead.

In any case, you should now have some idea of which browsers you need or want to target, even if it's only elaborate guesswork. This will inform your device purchases.

***

Now that we've gone through the simple stuff it's time to look at the more complicated part of the story: Android.

Index


THE MOBILE WEB HANDBOOK
PETER-PAUL KOCH
THE MOBILE WEB HANDBOOK KOCH