designing with web standards third edition

jeffrey zeldman
with ethan marcotte

New Riders

VOICES THAT MATTER™

designing with web standards

third edition

Contents at a Glance

This page intentionally left blank

designing with web standards

third edition

jeffrey zeldman with ethan marcotte

1249 Eighth Street, Berkeley, California 94710
An Imprint of Pearson Education

Designing with Web Standards, Third Edition

Jeffrey Zeldman with Ethan Marcotte

New Riders
1249 Eighth Street
Berkeley, CA 94710
510/524-2178
510/524-2221 (fax)

Find us on the Web at: www.newriders.com
To report errors, please send a note to errata@peachpit.com
New Riders is an imprint of Peachpit, a division of Pearson Education

Project Editor: Michael J. Nolan
Development Editor: Erin Kissane
Production Editor: Tracey Croom
Copyeditor: Rose Weisburd
Proofreader: Liz Merfeld
Indexer: Fred Leise
Composition: Kim Scott, Bumpy Design
Cover designer: Aren Howell
Interior designer: Kim Scott and Charlene Will
Editorial Assistant: Krista Stevens
Editorial Interns: Henry Li and Nicole Ramsey

Notice of Rights

Notice of Liability

Trademarks

ISBN 13: 978-0-321-61695-1
ISBN 10: 0-321-61695-2

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

Once again, to Ava. —JZ

To Elizabeth, my impossibly wonderful wife.
We've got a world that swings. —EM

Acknowledgments from Jeffrey

This book is the product of many people to whom I give profoundest thanks.

My gratitude to our fabulous interns, Henry Li and Nicole Ramsey, Microsoft Word wranglers extraordinaire. Without them, this book would be full of burps and hiccups.

To Krista Stevens, who interrupted her frenetic schedule as editor of A List Apart to manage tasks and timings.

To Aaron Gustafson, technical editing colossus. Nobody knows more about anything. The man is a genius. And humble. And lovely. My thanks to him.

The brilliant Ethan Marcotte agreed to co-author this edition, and one mark of his greatness in that capacity is this: in rereading the galleys as this book goes to press, I can't tell where my writing stops and Ethan's begins. My thanks to him, his bride, and all his antecedents.

Despite all the talent mentioned here, this book would be a catastrophic flop if not for the invincible fortitude and strategic verbal cunning of Ms Erin Kissane, editor to the stars. Dear Erin, six words: thank you, thank you, thank you.

This edition and the HTML5 specification have benefited from the thinking of some extremely gifted individuals who flew to Happy Cog's New York studio to review the HTML5 spec, identify its virtues, and tone up its flab. Big thanks to my fellow HTML5 Super Friends (www.zeldman.com/superfriends/), Dan Cederholm, Tantek Çelik, Wendy Chisholm, Aaron Gustafson, Jeremy Keith, Ethan Marcotte, Eric A. Meyer, and Nicole Sullivan (www.flickr.com/photos/zeldman/3813120876/).

Many thanks are due to Michael Nolan, who brought me to New Riders in 2000, and who said to me in 2001, "You ought to write a book about web standards."

Angel trombones of gratitude to John Allsopp, author of Developing with Web Standards, this book's code-rich companion volume.

Hardly least, my thanks and praise to the crew at Peachpit who built the book: Liz Merfeld, Kim Scott, Rose Weisburd, and Tracey Croom.

Lastly, to first-timers and returning readers, thank you for sharing the journey.

—JZ

About Jeffrey Zeldman (author)

Dubbed King of Web Standards by *Business Week*, Jeffrey Zeldman was one of the first designers, bloggers, and independent publishers on the web, and one of the first web design teachers. In 1998, he co-founded and designed—and from 1999 to 2002 he directed—The Web Standards Project, a grassroots coalition that brought standards to our browsers and initiated the web standards movement.

Since 1998, Jeffrey has published and directed the industry-leading magazine *A List Apart* "for people who make websites" (www.alistapart.com). He co-founded the web design conference An Event Apart (www.aneventapart.com) with Eric Meyer, and founded and is executive creative director of Happy Cog™ (www.happycog.com), a high-end web design agency with studios in New York, Philadelphia, and San Francisco. Happy Cog clients include AIGA, Zappos, the U.S. Holocaust Memorial Museum, WordPress, Housing Works, the W3C, Fetch Software, Sundance Festival, W.W. Norton & Co., Mozilla Creative Collective, MICA, Brighter Planet, and The Amanda Project.

Jeffrey has written two books, including *Taking Your Talent to the Web*, a guide for transitioning designers and art directors (New Riders, 2001: now available as a free download at www.zeldman.com/talent). He serves on the Advisory Boards of the SXSW Interactive Festival and Rosenfeld Media, and is a co-founder of The Deck advertising network (www.decknetwork.net), the premier network for reaching creative, web, and design professionals.

Before becoming a web designer, he worked as a composer and performing musician, as a journalist for *The Washington Post* and *City Paper*, and as an advertising copywriter and art director. Jeffrey blogs at www.zeldman.com (since 1995) and www.twitter.com/zeldman. He lives in New York City with his daughter Ava and a small dog named Emile.

Acknowledgments from Ethan

My thanks to the talented Ian Adelman and the design team at New York Magazine, for being one of the finest clients I'll ever work with—finer still for letting me talk about their beautiful site in these pages.

Erin Kissane is the most careful, most thorough, most superlative-exhausting editor I've ever worked with. The English language needs more people like her. Thank you so very much, Erin.

Aaron Gustafson is a miracle of a technical editor, providing careful feedback and thoughtful criticism. He is, as the kids say, a true rockstar.

If I have a career today, it's because I picked up a little orange book several years ago, penned by a fellow whose writing I'd long admired. To work with that fellow on a new edition of that book has been a weird, wonderful dream come true, and I don't have the words to properly thank Jeffrey Zeldman for the opportunity. I don't know if I ever will.

About Ethan Marcotte (author)

Ethan Marcotte is a versatile user experience designer/developer, whose work demonstrates a passion for the intersection of quality code and compelling design. Prior to joining Happy Cog, Ethan worked with such clients as *New York Magazine*, Harvard University, and the W3C. A former steering committee member of The Web Standards Project, his work in the standards space has been covered in several magazines and online publications.

Ethan has acted as a contributing author to *Handcrafted CSS* (New Riders, 2009), *Web Standards Creativity* (friends of ED, 2007) and *Professional CSS* (Wrox, 2005). Ethan is an experienced technical editor, having edited the first edition of *Bulletproof Web Design* (New Riders, 2005), as well as the second edition of *Designing with Web Standards* (New Riders, 2006).

Ethan is a contributing author and technical editor at *A List Apart*, "for people who make websites." He is also a popular educator, and has been a featured speaker at An Event Apart, the SXSW Interactive festival, Harvard University, and AIGA's In Control conference. He spends entirely too much time online, and would like to be an unstoppablerobot ninja (www.unstoppablerobotninja.com) when he grows up. Beep.

Aaron Gustafson (technical editor, third edition)

After getting hooked on the web in 1996 and spending several years pushing pixels for the likes of IBM and Konica Minolta, Aaron Gustafson founded Easy! Designs, LLC (www.easy-designs.net), a boutique web consultancy. Aaron is a member of The Web Standards Project (WaSP), serves as Technical Editor for *A List Apart*, contributes to MSDN, and has written and edited books including *Accelerate DOM Scripting with Ajax, APIs, and Libraries* (Apress, September 27, 2007), *AdvancED DOM Scripting* (Friends of Ed, 2007) and *Web Design in a Nutshell* (3rd Edition, O'Reilly). He is a regular on the web conference circuit and provides web standards and JavaScript training in both the public and private sector. He blogs at www.easy-reader.net.

J. David Eisenberg (technical editor, first edition)

J. David Eisenberg lives in San Jose, California, with his cats Marco and Big Tony. He teaches HTML, XML, Perl, and JavaScript at Evergreen Valley College, and enjoys writing online tutorials. He is the author of *SVG Essentials* (O'Reilly & Associates) and *OASIS OpenDocument Essentials* (Friends of OpenDocument, Inc.). David attended the University of Illinois, where he worked with the PLATO computer-assisted instruction project. He has also worked at Burroughs and Apple.

Eric Meyer (technical editor, first edition)

Eric Meyer is an internationally recognized expert in CSS and the use of web standards, and has been working on the web since late 1993. He is the best-selling CSS author and best-recognized CSS authority in the world. His seven books have been translated into six languages and have sold in the hundreds of thousands. Eric is currently the principal of Complex Spiral Consulting (www.complexspiral.com), which focuses on helping clients use standards to cut costs and improve user experience. In that capacity, he has assisted organizations from universities to government laboratories to Fortune 500 companies; some recent and notable clients include America On-Line, Apple Computer, Macromedia, Sandia National Laboratory, and Wells Fargo Bank.

Table of Contents

Preface to the Third Edition

When my colleagues and I started The Web Standards Project (WaSP) in 1998, we did not know that valid, semantic markup would make your site's content attractive to Google. It does, but that's not what we were concerned about in those pre-Google days. In the late 1990s, a clever web designer was one who could code her client's site in the five ways necessary to make it work and look right in Netscape 3, Netscape 4, IE3, IE4, and whatever else.

If you wanted your site to do anything besides sit there in Netscape and Microsoft's 3.0 and 4.0 browsers, you had to author using two generations of two incompatible scripting languages—four incompatible scripts per page in all, and all of them inline. All those inline scripts sat atop complex HTML table layouts, resulting in pages that were at least 60% heavier than they needed to be. And when it came time to redesign—or when you just needed to repurpose your content—doing so was harder and more costly than it needed to be, because content and layout were all jumbled up together.

Pref 1.
A big benefit of web standards, from a site owner's perspective, is that they make content easy for people and search engines to find. (**www.twitter.com/zeldman/status/1137456194**)

To my fellow WaSP co-founders and me, all that browser-specific coding presented a serious threat to web development. It took at least 25% of every site's development time and thus added at least 25% to the cost of every site. Moreover, if designing the primitive websites of 1998 took five kinds of code, what did the future of web design look like? Would the most thrilling advance in publishing and communication since the printing press be held back by the deliberate incompatibilities of competing browsers? Ten years in the future, would even the simplest websites be burdened by the cost and effort of coding each site twelve ways?

Web standards—semantic (X)HTML markup, CSS layout, and unobtrusive scripting based on JavaScript and the DOM—were the solution to all these problems, and by hammering away at that idea, standards advocates at The Web Standards Project and elsewhere eventually persuaded browser makers to support standards—and many web professionals to use them.

The benefits of designing with web standards, which Part I of this book will explain in detail, are

- Appropriate, semantic markup makes your content easy for people and search engines to find. Merely by converting from nonsemantic table layouts to well-structured semantic markup, sites can often score much

higher in search engine listings, bump up their Alexa ranking, and show equal improvement along other findability metrics. (Findability is the *standardista's* preferred term for the promised benefit of *search engine optimization*, or SEO).

- Separating your site's presentation from its structure and behavior makes your site easier and less expensive to develop and test, lowering your overall budget or freeing cash for things like usability testing and content development.
- Separating your site's presentation from its structure and behavior makes your site lighter, thus improving performance.
- Separating your site's presentation from its structure and behavior and using appropriate semantic markup makes your site more accessible to different kinds of browsers and devices, including mobile devices *and* browsing tools used by people with disabilities—from screen readers to alternative input devices. Designing with standards doesn't mean there are no benefits to offering a mobile version. But it makes it easier to pull together a mobile version if you choose to; and in some cases, depending on your site's content, it may remove the need to do so.
- Designing with standards instead of browser-specific code future-proofs your site and its content. If your site is correctly authored in HTML 4.01 or XHTML 1.0 and laid out with CSS2, browsers will support it indefinitely. Even when HTML5, CSS3, and other emerging specifications have been long finalized, and even if they are brilliantly supported in future browsers, the site you author with today's standards will still work perfectly. Sites not authored to web standards have no such guarantee. At best, they will be stuck in time; at worst, they will cease to work.

Translated into fifteen languages from Bulgarian to Korean, the first two editions of this book introduced the notion and methods of standards-based design to hundreds of thousands of designers and developers and their clients worldwide. Since the first edition hit the shelves in 2003, there is scarcely an agency or in-house department that does not have at least one web standards and accessibility evangelist on staff. The industry has grown up, and it is no longer only the leading-edge designer who embraces web standards, no longer just the boutique agency that can claim profound expertise in standards-based design.

This third edition faces a world where web standards are becoming mainstream; where designers and developers argue about CSS3, and more than a few are trying their hand at microformats and HTML5; and where advanced standards compliance is a leading checkbox for software from Apple, Google, Opera, Adobe, Microsoft, and the open source community.

This substantially revised and rewritten edition has been restructured in two parts:

- Part I explains the problems created by old methods of web design and how web standards solve them. It also provides arguments in defense of standards-based design for those who must "sell" these practices to skeptical clients, colleagues, and employers. We next survey the expanding world of web standards, showing how old and new standards are turning the web into a dynamic platform of robust applications and accessible, easy-to-find (and beautifully styled) content. The section ends with a look at the web's future.
- Part II introduces the reader to XHTML, HTML5, and CSS, and (more importantly) to the principles of structured, semantic markup; clean, robust, optimal CSS layout; and unobtrusive scripting. After significant sidebars on typography and accessibility, it ends by taking us deep inside selected web design projects to reveal their standards secrets.

I am beholden to my new coauthor Ethan Marcotte, to our editor Erin Kissane, and to technical editor Aaron Gustafson. No finer team exists. If there is good in this book, thank these folks.

To new friends discovering web standards for the first time, welcome! And to those companions who have journeyed with us before, welcome back.

Part I

This page intentionally left blank

Before You Begin

This book is for designers, developers, owners, and managers who want their sites to cost less, work better, and reach more people—not only in today's browsers, screen readers, and wireless devices, but in tomorrow's, next year's, and beyond.

Most of us have gone a few rounds with the obsolescence that seems to be an inescapable part of the web's rapid technological advancement. Every time an improved browser version or new internet device comes onto the scene, it seems to break the site we just finished producing (or paying for).

We build only to rebuild. Too often, we rebuild not to add visitor-requested features or increase usability, but merely to keep up with browsers and devices that seem determined to stay one budget-busting jump ahead of our planning and development cycles. Even on those rare occasions in which a new browser or device mercifully leaves our site unscathed, the so-called "backward-compatible" techniques we use to force our sites to look and behave the same way in all browsers take their toll in human and financial overhead.

We're so used to this experience that we consider it normative—the price of doing business on the web. But it's a cost most of us can no longer afford, if we ever could.

Ending the Cycle of Obsolescence

Technologies created by the World Wide Web Consortium (W3C) and other standards bodies and supported by most current browsers and devices make it possible to design sites that will continue to work, even as those standards and browsers evolve.

What Is the W3C?

Created in 1994, the World Wide Web Consortium (`www.w3.org`) creates specifications and guidelines that are intended to promote the web's evolution and ensure that web technologies work well together. Roughly 500 member organizations belong to the consortium. Its director, Tim Berners-Lee (`www.w3.org/People/Berners-Lee`), invented the web in 1989. Specifications developed by the W3C include HTML 4, CSS2, CSS3, XML, XHTML 1.0, XHTML 1.1, HTML5 (created in cooperation with browser makers in the Web Hypertext Application Technology Working Group, or WHATWG), and the standard Document Object Model (DOM), among many others.

Other standards bodies include the European Computer Manufacturers Association (ECMA), which is responsible for the language known as ECMAScript and more familiarly referred to as "standard JavaScript."

This book will teach you how to escape the "build, break, rebuild" cycle without excluding potential visitors or wasting time and money on short-sighted, proprietary "solutions" that contain the seeds of their own doom. It will tell you what you need to know to work around the occasional compliance hiccup in Internet Explorer and other modern browsers, and it will offer strategies for coping with the bad old browsers that might be used by some in your audience, including that stubborn guy in the corner office.

Armed with this book, designers and developers will be able to modify their practices to create websites that work in many browsers and devices instead of a handful, while avoiding perpetual obsolescence born of proprietary markup and coding techniques. Site managers who read this book will be able to stop wasting money on specs that only perpetuate the wasteful cycle and learn instead how to write requirements documents that lead to forward-compatible sites.

No Dogma

This is not a dogmatic book. There is no best way to design a website, no one right way to incorporate standards into your workflow. This book will not advocate strict standards compliance at the expense of transitional approaches that might be better suited to particular sites and tasks. Moreover, this is not a book for theorists or purists, but for people who need to get work done.

I have nothing against the purists whose passion drives the creation of web standards; I admire such people immensely and am lucky enough to have befriended and learned from a number of them. But this book is for working designers and developers and the clients and employers who pay for their expertise, and its exploration of web standards will be rooted in the context of design, content, and marketing issues. Where W3C standards are fuzzy and implementation practices are controversial, I'll share any consensus the standards community has reached and help you make your own decision.

If this book contains one kernel of dogma, or holds one fixed, inflexible view, it's this: the cost of business as usual is too high.

No one reading this book can afford to design today's websites with yesterday's piecemeal methods. Coding every site six ways might have seemed a reasonable practice when the internet boom and grotesquely over-inflated budgets were at their height, but that day is gone. HTML, XHTML, XML, CSS, JavaScript, and the DOM are here to stay—not as ends in themselves, but as components of a rational solution to problems that have plagued site owners and builders since the `blink` tag.

A Continuum, Not a Set of Inflexible Rules

As this book will emphasize, web standards are a continuum, not a set of inflexible rules. In moving to web standards, you might not achieve perfect separation of structure from presentation in your first site or even your fifth. Your first efforts at accessibility might deliver only the minimum required by Web Content Accessibility Guidelines (WCAG) 1.0 Priority 1, and you might not get all of it exactly right. (You'd be in good company; the difficulty of knowing for sure that you've gotten WCAG 1.0 right led the W3C to issue a very different WCAG 2.0 specification in December 2008.)

The point is to begin. Fear of imperfection can immobilize the unwary the same way that shame about our flab might keep us from going to the gym. But we won't begin to lose the excess avoirdupois until we make our first fumbling efforts at physical fitness. Likewise, our sites won't attain forward compatibility if we don't start somewhere. Deleting `font` tags might be where you start. Or you might replace nonsemantic markup with meaningful `h1` and `p` tags. This is often an excellent place to begin, and as a consequence, this book will spend a fair amount of its time and yours considering modern markup.

A Few Important Definitions

In this book from time to time I refer to *forward compatibility*. What exactly do I mean by that? I mean that, designed and built the right way, any page published on the web can work across multiple browsers, platforms, and internet devices—and will continue to work as new browsers and devices are invented. Web standards make this possible.

What do I mean by web standards? I mean the same thing The Web Standards Project means: namely, structural languages like HTML, XHTML, and XML, presentation languages like CSS, object models like the W3C DOM, and scripting languages like JavaScript, all of which will be explained in this book. (Site owners and managers: Don't worry yourselves over the technical chapters in this book. Just make sure your employees or vendors understand them.)

Hammered out by experts in working groups, these technologies are designed to deliver the greatest benefits to the largest number of web users. Taken together, web standards form a roadmap for rational, sophisticated, and cost-effective web development. As an added attraction, designing with standards

also makes sites more accessible to those who have special needs. (Translation: you'll have more customers, lower costs, improved public relations, and a decreased likelihood of accessibility-related litigation.)

What Is The Web Standards Project?

For years, the W3C referred to its specs as "Recommendations," which might have inadvertently encouraged member companies such as Netscape and Microsoft to implement W3C specs less than rigorously. On its launch in 1998, The Web Standards Project (WaSP)—a grassroots coalition of web designers and developers—relabeled key W3C Recommendations "web standards," a guerrilla marketing maneuver that helped reposition accurate and complete support for these specs as a vital ingredient of any browser or internet device.

WaSP (`www.webstandards.org`) advocates standards that reduce the cost and complexity of site creation and ensure simple, affordable access for all. Thanks to its efforts, today every browser supports web standards as a matter of course. The group also works with the makers of site development tools like Dreamweaver, and with site owners and designers. Today, The Web Standards Project also provides web design and web standards education, as well as outreach beyond the English-speaking Western world.

What do I mean by standards-compliant browsers? I mean Apple's WebKit-based Safari 3 and 4 (which also powers the iPhone's web browser); the open source, Mozilla-powered Firefox 3 and higher; Opera Software's Opera 9+ (and the Opera Mini browser found in many great phones); the WebKit-based Google Chrome; and Microsoft Internet Explorer 8 (and to a lesser extent, IE7—and to a still lesser extent, IE6). What do these products have in common? To varying degrees, they understand and correctly support such standards as XHTML 1.0, CSS2, JavaScript, and the DOM. (All but IE also support tasty bits of CSS3 and some advanced features of HTML5.)

Are these browsers perfect in their support for every one of these standards? Of course they're not. No software yet produced in any category is entirely bug-free. Moreover, standards are sophisticated in themselves, and the ways they interact with each other are complex.

Are some of these browsers better than others where standards compliance is concerned? Every working developer knows the answer to that one. Most of us agree that even the latest version of Internet Explorer, while very good, is not as advanced as Safari, Firefox, and Opera. And of course, many people view the web through old versions of IE that are even less compliant.

But even if your audience hasn't upgraded their PCs since IE6 and Vanilla Ice were hot, cross-browser support is now solid enough that we can discard outdated methods, work smarter, and satisfy more users by designing with standards. And because standards are inclusive by nature, we can even accommodate folks who use older browsers and devices—in a forward-compatible way.

One Size Does Not Fit All

This book is large and has been crafted with care over three editions, yet it barely scratches the surface of what standards mean to the web. There is more to CSS, more to accessible, structured markup, and far more to the DOM and JavaScript than what this book or any single reference could convey. And as I've already mentioned, there are more ways to view the issues covered than the way this author looks at them.

Since the first edition of this book introduced standards to a worldwide audience, a new book with the words "web standards" in its title seems to come out each week. Many of these books offer the same advice as the volume you hold in your hands; others contradict some things said here. Different techniques advance different agendas.

Put two designers in a room and you will hear three opinions. No two designers agree on typography, branding, layout, or color. The same is true with web standards. Not every reader will immediately use every idea discussed in this book. But any thinking designer, developer, or site owner should be able to endorse the general notions advanced in this book. Doing so saves time and money, reduces overhead, extends the usable life of our sites, and provides greater access to our content.

This book will have done its job if it helps you understand how standard technologies can work together to create forward-compatible sites—and provides

a few tips to help you along your way. You may disagree with some of what I have to say here, but the point is not to bog down in differences or reject the whole because you're uncertain about one or two small parts. The point is to begin making changes that will help your web projects reach the most people for the longest time, and often at the lowest cost.

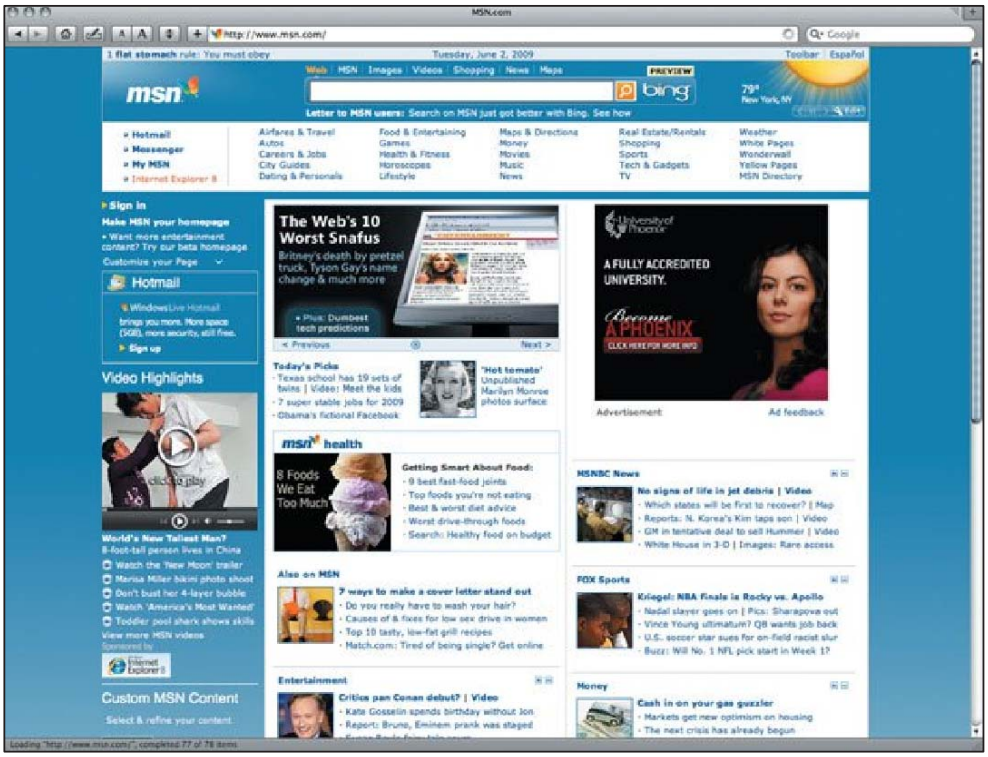
Welcome to the Winning Team

After years in which nothing new seemed to be going on at the W3C, the latest browsers support exciting (and still unfinished) specifications like HTML5 and CSS3—and companies as powerful as Google are betting the farm on web applications powered by these technologies (`radar.oreilly.com/2009/05/google-bets-big-on-html-5.html`). Progressive enhancement techniques made possible by CSS3 are bringing new beauty—and new design thinking—to our medium. Real typography is a heartbeat away (`www.zeldman.com/x/16`). And organizations as diverse as Apple, MSN, Wikipedia, and WordPress have embraced web standards in their DNA, even if they don't always achieve perfect validation or 100% pure semantic markup (**0.1**-**0.4**).

0.1
Known for the sophisticated design of its products, Apple has a secret—its site is powered by web standards (**`www.apple.com`**).

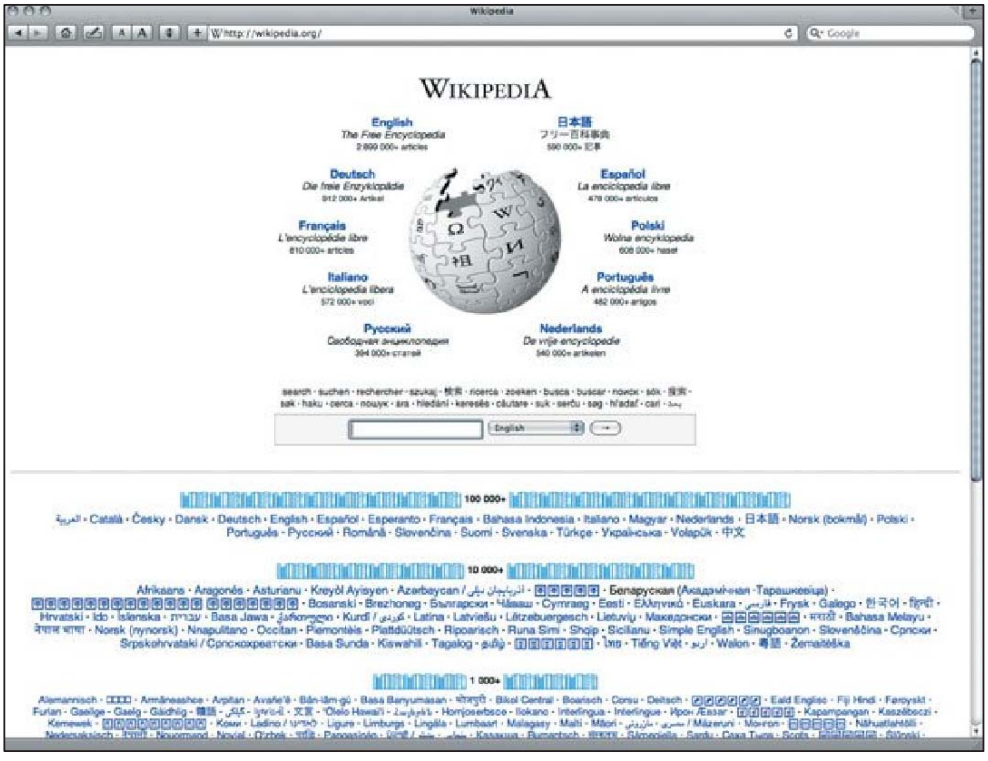
0.2

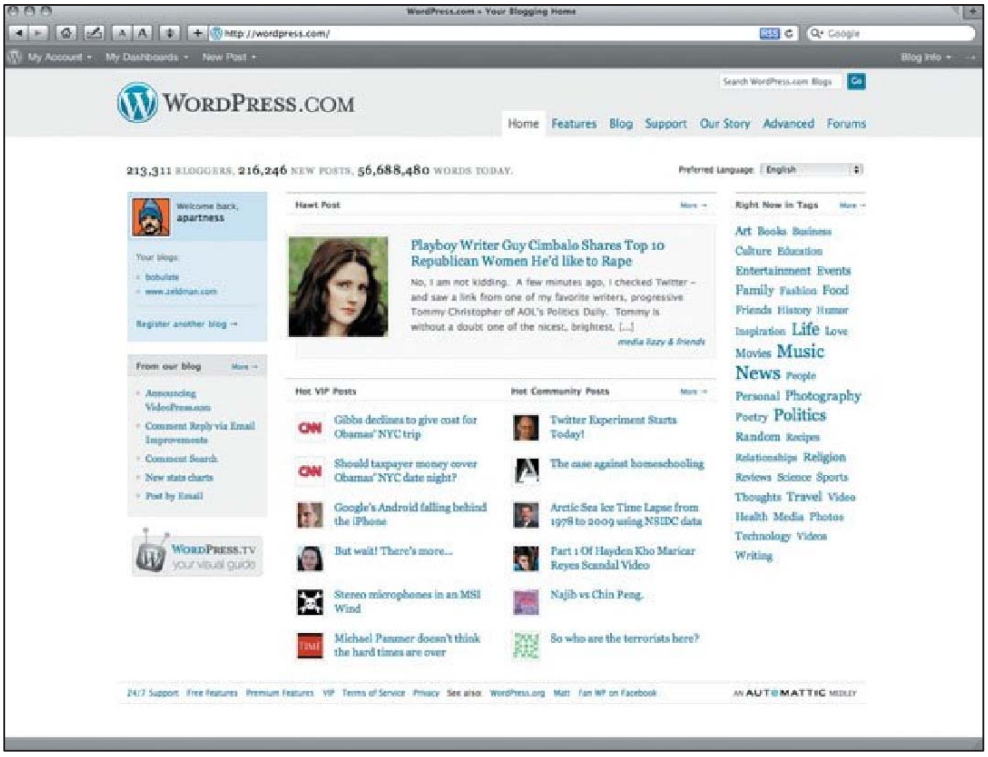
MSN, #7 in Alexa's Top 100 sites, validates as XHTML Strict (**www.msn.com**).



0.3

When you want answers, you go to Wikipedia. When Wikipedia wants stability, it goes to XHTML 1.0 Strict markup and CSS layout (**www.wikipedia.org**).





0.4
Open source blogging platform WordPress, #22 in Alexa's Top 100 sites, validates as XHTML 1.0 Transitional (**`www.wordpress.com`**).

Web standards are the tools with which all of us can design and build sophisticated, beautiful sites that will work as well tomorrow as they do today. In this book, I'll explain the job of each standard and how all of them can work together to create forward-compatible sites. The rest is up to you.

This page intentionally left blank

chapter one

99.9% of Websites Are Obsolete

An equal opportunity disease afflicts nearly every site now on the web, from the humblest personal home pages to the multi-million-dollar sites of corporate giants. Cunning and insidious, the disease goes largely unrecognized because it is based on industry norms. Although their owners and managers may not know it yet, 99.9% of all websites are obsolete.

These sites might look and work all right in mainstream, desktop browsers. But outside these fault-tolerant environments, the symptoms of decay have already started to appear.

In modern versions of Microsoft Internet Explorer, Opera Software's Opera browser, Apple's Safari, Google's Chrome, and Mozilla (the open source, Gecko-based browser whose code drives Firefox and Camino), carefully constructed layouts have begun to fall apart and expensively engineered behaviors have stopped working. As these leading browsers evolve, site performance continues to deteriorate.

In "off-brand" browsers, in screen readers used by people with disabilities, and in too many mobile phones, large numbers of these sites have never worked and still don't, while others function marginally at best. Depending on their needs and budget, site owners and developers have either ignored these off-brand browsers and devices, or supported them by detecting their presence and feeding them customized markup and code, just as they do for "regular" browsers.

To understand the futility of this outdated industry practice and to see how it increases the cost and complexity of web development without achieving its intended goal, we must consider how modern, standards-compliant browsers differ from the browsers of the past.

Modern Browsers and Web Standards

Throughout this book, when I refer to "modern" or "standards-compliant" browsers, I mean those that understand and support HTML and XHTML, Cascading Style Sheets (CSS), ECMAScript (more commonly referred to as "standard JavaScript"), and the W3C Document Object Model (DOM). Taken together, these are the standards that allow us to move beyond presentational markup and incompatible scripting languages and the perpetual obsolescence they engender.

Such browsers include the award-winning, open source Firefox 3.5+; Microsoft Internet Explorer 7/8 and higher for Windows; Apple's Safari 3.0/4.0+ for Macintosh OS X; Google's Chrome; and Opera Software's Opera 9/10+. (Have I left out your favorite standards-compliant browser? I mean it no disrespect. Any attempt to list every standards-compliant browser would date this book faster than the Macarena.) Although I use the phrase "standards-compliant," please remember that no browser is, or can be, *perfectly* standards compliant.

But lack of browser perfection is no reason to avoid standards. Millions of people still use Internet Explorer 6 for Windows, and that browser's support for standards is decidedly inferior to IE7+, Firefox, Opera, and Safari. Does that mean if your audience includes IE6 users you should forget about web standards? Does it mean you should tell IE6 users to upgrade or get lost? Of course not! Standards-oriented design and development need not and should not mean "designing for the latest browsers only."

Likewise, using XHTML and CSS need not necessitate telling users of older browsers to go take a hike. But a site properly designed and built with standards is unlikely to look the same, pixel-for-pixel, in IE6 as it does in more compliant browsers. In fact, depending on your design method, it might look entirely different—and that's probably OK. Indeed, some designers have gone so far as to recommend serving IE6 users core styles that enhance readability while making no attempt to ape the look and feel of the site as viewed in more compliant browsers. See Andy Clarke's "Universal Internet Explorer 6 CSS" (`www.forabeautifulweb.com/blog/about/universal_internet_explorer_6_css`) for more about this approach.

New Code for a New Job

Modern browsers are not merely newer versions of the same old thing. They differ fundamentally from their predecessors, and in many cases, they've been rebuilt from the ground up. Mozilla Firefox, Camino, and related Gecko-based browsers are not new versions of the long-extinct Netscape 4. Opera 10 is not based on the same code that drove earlier versions of the Opera browser. These products have been built with new code to do a new job: namely, to comply as well as possible with the web standards discussed in this book.

In contrast, the browsers of the 1990s focused on proprietary (Netscape-only, Microsoft-only) technologies and paid little heed to standards.

Old browsers ignored some standards altogether, but paradoxically this lack of support wasn't much of a headache. If browsers didn't support the Portable Network Graphic (PNG) standard, for example, developers didn't use PNG images. No problem. But these old browsers also provided partial and incorrect support for some standards. This slipshod, inconsistent support for standards as basic as HTML created an untenable web-publishing environment.

When a patient's appendix bursts, a qualified surgeon performs an appendectomy. But what if, instead, a drunken trainee were to remove half the appendix, randomly stab a few other organs, and then forget to sew the patient back up? That's what standards support was like in old browsers: incompetent, incomplete, and hazardous to the health of the web.

If Netscape 4 ignored CSS rules applied to the `body` element and added random amounts of whitespace to every structural element on your page, and if IE4 got `body` right but bungled padding, what kind of CSS was safe to write? Some developers chose not to write CSS at all. Others wrote one style sheet to compensate for IE4's flaws and a different style sheet to compensate for the blunders of Netscape 4. To compensate for cross-platform font and UI widget differences, some developers also served different style sheets for different operating systems.

CSS wasn't the only trouble spot: browsers couldn't agree on HTML, on table rendering, or on scripting languages used to add interactivity to the page. There was no fully supported correct way to structure a page's content, produce the design of a page, or add sophisticated behavior to a site.

Struggling to cope with ever-widening incompatibilities, designers and developers authored customized versions of (nonstandard) markup and code for each differently deficient browser that came along. It was all we could do at the time if we hoped to create sites that would work in more than one browser or operating system. It's the wrong thing to do today, because all modern browsers support the same open standards. Yet the practice persists, needlessly gobbling resources, fragmenting the web, and leading to inaccessible and unusable sites.

The "Version" Problem

The creation of multiple versions of nonstandard markup and nonstandard code is the source of the perpetual obsolescence that plagues most sites and their owners. But although it's costly, futile, and unsustainable, the practice persists.

Faced with a browser that supports web standards, many developers treat it like one that doesn't. Thus, they'll write detection scripts that sniff out IE6 and feed it Microsoft-only code even though IE6 can handle standard JavaScript and the DOM. They then feel compelled to write separate detection scripts (and separate code) for modern Mozilla-based browsers that can also handle standard JavaScript and the DOM.

The Perils of Browser Sniffing

All browsers have a user agent (UA) string, a short string of text that contains the browser's name, version, and platform, and acts as a digital fingerprint for the browser. Web servers frequently log the UA, providing valuable information to site owners hoping to better understand their audience. But all too frequently, site owners use JavaScript or server-side code to parse the UA string so they can serve platform-appropriate or version-specific content. The problem with this approach is that the user agent string is unreliable. Depending on the user's security or network settings, a browser may not relay its UA to the server, causing these browser-sniffing scripts to fail, preventing users from reaching their destination.

Furthermore, all modern browsers allow the user to change the UA, whether through user-installed extensions (Firefox) or as part of its native feature set (Safari and Opera). In fact, prior to version 8.02, Opera identified itself as Internet Explorer *by default* to avoid being blocked by the many sites that lock out everyone who doesn't use IE.

The fallibility of UA sniffing was underscored during the recent release of Opera 10, the first browser release to reach double digits. This happy milestone was soured during the browser's beta release, when a legion of shoddily written browser detection scripts failed to properly register the second digit in Opera 10's version number, instead misidentifying the browser as "Opera 1." This problem was so widespread that Opera changed its browser to identify itself not as "Opera 10.00", but "Opera 9.80", the number of its last single-digit release (`dev.opera.com/articles/view/opera-ua-string-changes`).

Unfortunately, this Y2K-like version rollover fate awaits all browsers that have yet to reach version 10, unless we do away with browser sniffing, and adopt a more standards-aware approach.

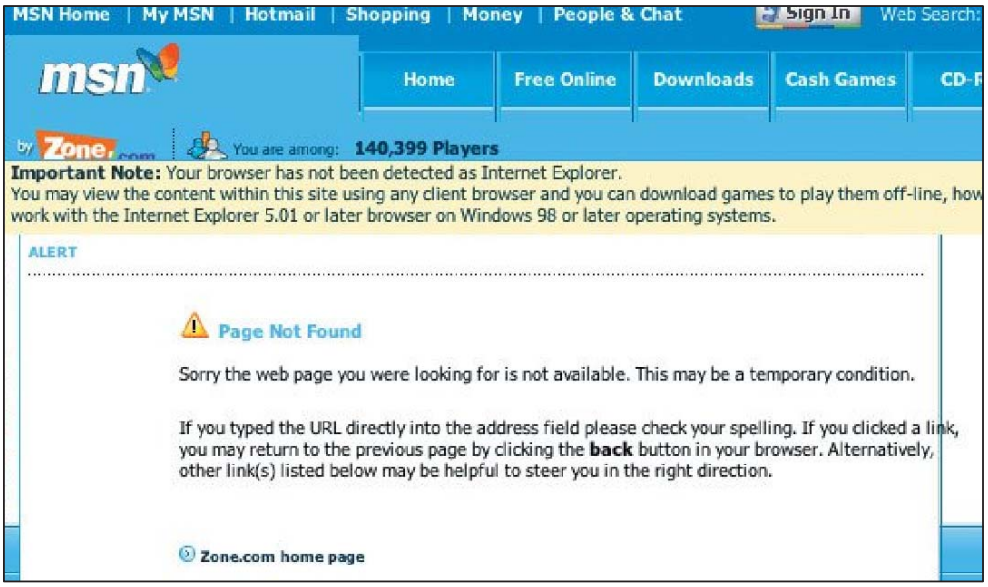
With multiple versions come ever-escalating costs and conundrums. Sites produced to the proprietary scripting specifications of long-extinct browsers don't work in modern browsers and devices. Should the site owner throw more

money at the problem, asking developers to create a fifth or sixth version of the site? What if there's no budget for such a version? Many users will be locked out.

Even when they embrace standard web technologies like XHTML and CSS, designers and developers who cut their teeth on old-school methods often miss the point. Instead of using standards to *avoid* multiple versions, many old-school developers create multiple browser- and platform-specific CSS files that are almost always self-defeating [**1.1**, **1.2**].

1.1
In 1999, the MSN Game Zone (**zone.msn.com/blog.asp**) sported three external style sheets and still didn't render properly in most modern browsers. It also contained multiple scripts (most of them inline), including heavy-duty browser detection. And it still didn't work. Throwing more versions of code at a problem rarely solves it.

1.2
To be fair, the preceding screen shot is four years old. Today, the same web page is even worse, as yet more code gets thrown at it. Six years after Microsoft brought the first standards-compliant browser to market, parts of the Microsoft site still don't know the first thing about designing with standards.



These practices waste time and money. Neither commodity has ever been abundant. Creating multiple site versions nobody can afford does nothing to advance the cause for which so much time and treasure have been squandered. Sites are still broken, and users are still locked out.

The Junkman Cometh

Peel the skin of major commercial sites like Amazon.com and eBay.com. Examine their tortuous nonstandard markup, their convoluted JavaScript (often including broken detection scripts), and their ill-conceived use of CSS. It's a wonder such sites work in any browser.

Often, non-standards-compliant sites work in yesterday's browsers because their owners have invested in costly publishing tools that accommodate browser differences by generating multiple, nonstandard versions tuned to the biases of specific browsers and platforms, as described earlier in "The 'Version' Problem." The first four generations of Netscape Navigator and Microsoft Internet Explorer did not merely tolerate nonstandard markup and browser-specific code; they actually encouraged sloppy authoring and proprietary scripting in an ill-conceived battle to own the browser space.

Bad Markup: The First Bag Is Free

Early in a computer programmer's education, he or she learns the phrase "Garbage In, Garbage Out." Likewise, among the first things a graphic designer learns is that the quality of source materials determines the effectiveness of the end product. Start with a high-resolution, high-quality photograph, and the printed piece or web graphic will look good. Try to design with a low-quality snapshot or low-resolution web image, and the end result won't be worth viewing. Garbage in, garbage out.

But old browsers don't work that way. Lax to the point of absurdity, they gobble up broken markup and bad links to JavaScript source files without a hiccup, in most cases displaying the site as if it were authored correctly. This laxity has encouraged front-end designers and developers to develop bad habits—and also persuaded middleware and backend developers to view technologies like XHTML, CSS, and JavaScript as contemptibly primitive.

Those who do not respect a tool are unlikely to use it correctly. Consider the following snippet, lifted from the costly e-commerce site of a company competing in a tough market, and reprinted here in all its warty glory:

```
<td width="100%"><ont face="verdana,helvetica,arial" size="+1"
color="#CCCC66"><span class="header"><b>Join now!</b></span>
</ont></td>
```

The nonsensical `ont` tag is a typo for the deprecated `font` tag—a typo that gets repeated thousands of times throughout the site, thanks to a highly efficient publishing tool. That error aside, this markup might look familiar to you. It might even resemble the markup on your site. In the context of this web page, all that's actually necessary is the following:

```
<h3>Join now!</h3>
```

Combined with an appropriate rule in a style sheet, the preceding, simpler markup will do exactly what the cumbersome, nonstandard, invalid markup did, while saving server and visitor bandwidth and easing the transition to a more flexible site powered by semantic markup (possibly including machine-readable code such as microformats, discussed later in this book). The same e-commerce site includes the following broken JavaScript link:

```
<script language=JavaScript1.1src=
"http://foo.com/Params.richmedia=yes&etc"></script>
```

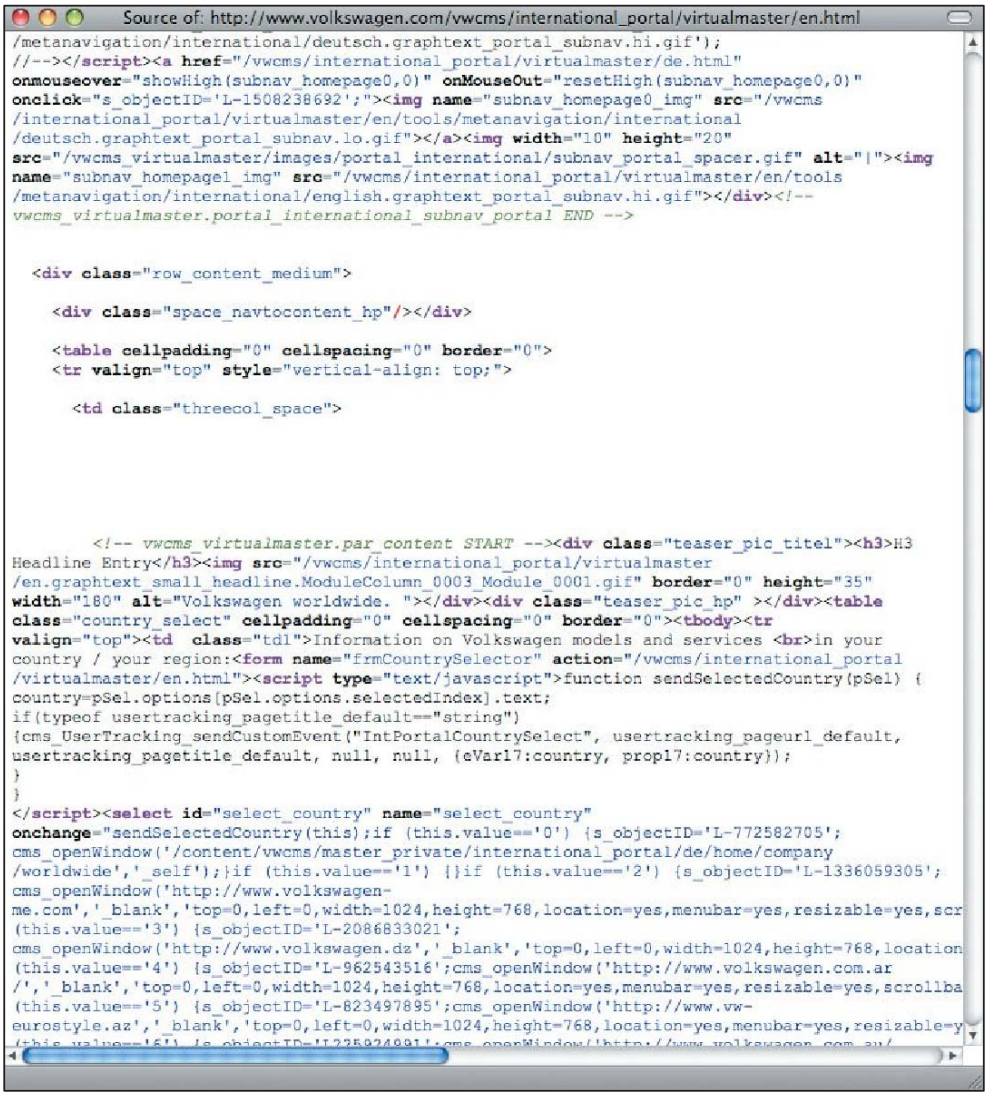
Among other problems, the unquoted language attribute erroneously merges with the source tag. In other words, the browser is being told to use a nonexistent scripting language ("`JavaScript1.1src`").

By any rational measure, the site should fail, alerting the developers to their error and prompting them to fix it pronto. Yet until recently, the JavaScript on this site worked in mainstream browsers, thus perpetuating the cycle of badly authored sites. Little wonder that skilled coders often view front-end development as brain-dead voodoo unworthy of respect or care.

As newer browsers comply with web standards, they are becoming increasingly intolerant of broken code and markup. Garbage in, garbage out is beginning to take hold in the world of browsers, making knowledge of web standards a necessity for anyone who designs or produces websites.

Code Forking Can Be Hazardous to Your Site's Long-term Health

More than one company I know has spent over a million dollars on an overly complex, not-terribly-usable content management system (CMS). The makers of this software behemoth partially justify its sickening cost by pointing to its ability to grind out all manner of nonstandard code versions. In addition to wasting obscene truckloads of cash, the practice hurts findability by drowning meaningful content in a sea of nonsemantic tags [1.3]. Unfairest of all, it taxes the dial-up (or smart phone) user's patience by wasting bandwidth on code forking, deeply nested tables, spacer pixels and other image hacks, and outdated or invalid tags and attributes.



1.3
Quick! Find "Information on Volkswagen models and services" in the convoluted, nonstructural HTML markup of Volkswagen.com. Tip: If you have a hard time finding it, so do readers and search engines. (**www.volkswagen.com**)

What Is Code Forking?

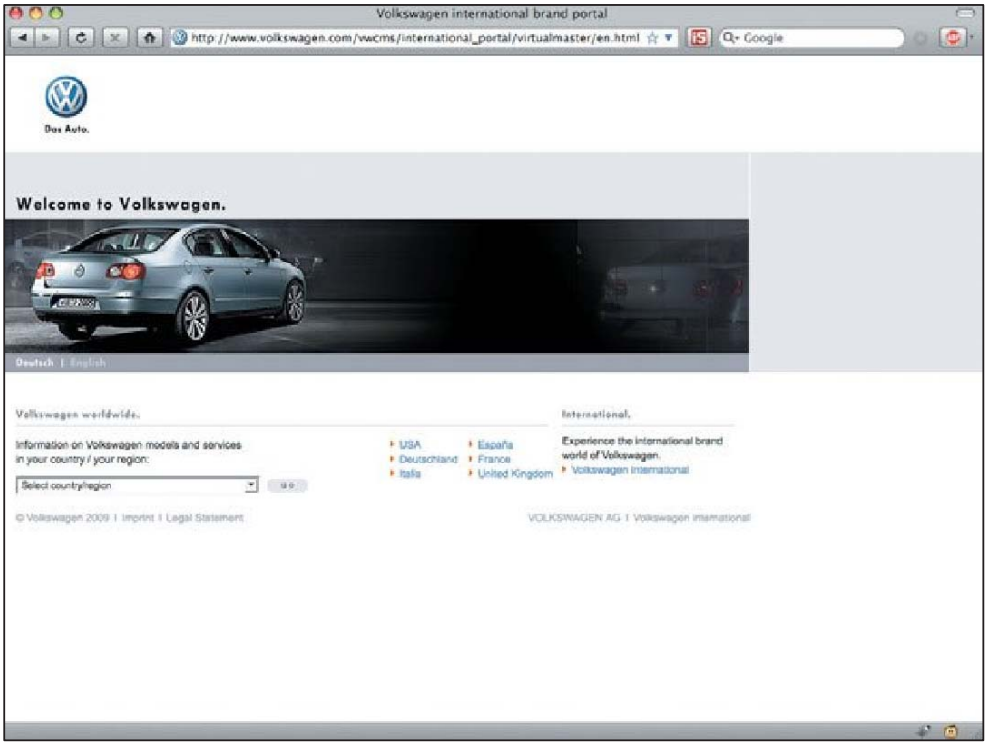
Code is the stuff programmers write to create software products, operating systems, or pretty much anything else in our digital world. When more than one group of developers works on a project, the code might "fork" into multiple, incompatible versions, particularly if each development group is trying to solve a different problem or bend to the will of a different agenda. This inconsistency and loss of centralized control is generally regarded as a bad thing.

As used in this book, *code forking* refers to the practice of creating multiple versions of incompatible code to cope with the needs of browsers that do not support standard JavaScript and the DOM (see "The 'Version' Problem," above).

At the same time, code forking squanders the site owner's bandwidth at a cost even the bean counters might be at a loss to calculate. The bigger the site and the greater its traffic, the more money is wasted on server calls, redundancies, image hacks, and unnecessarily complex code and markup.

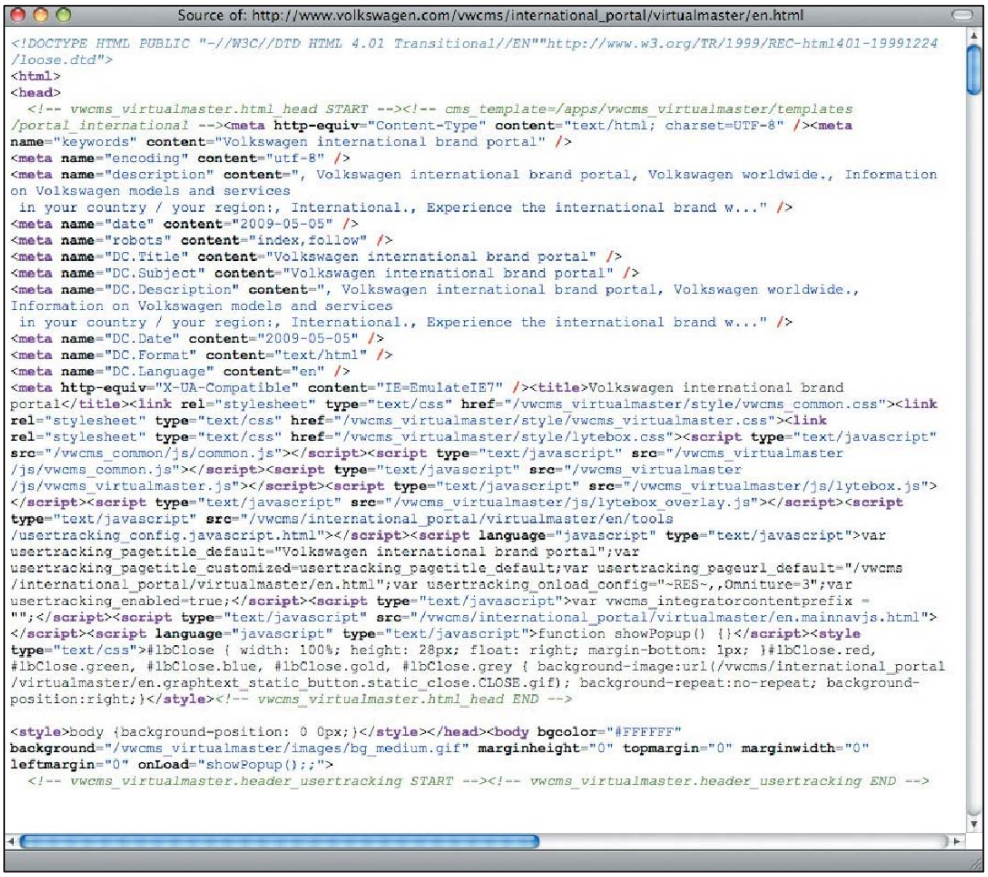
Hard numbers are hard to come by, but in general, if a site reduces its markup weight by 35%, it reduces its bandwidth costs by the same amount. An organization that spends $2,500 a year on bandwidth would save $875. One that spends $160,000 a year would save $56,000.

Consider the front page of Volkswagen.com [**1.4**]. Despite a straightforward, elegant design, the page's markup is weighted down by antiquated, table-bloated markup, inefficient JavaScript, and pages of inline CSS. Now, this is no amateur website: the international portal for one of the world's most recognizable brands is served millions of times per day. Multiply each wasted byte of outdated HTML design hacks by the site's staggering number of page views, and the result is gigabytes of bandwidth that both tax Volkswagen's servers and add Pentagon-like costs to its overhead. If Volkswagen simply replaced its inefficient, bandwidth-gobbling markup [1.3, **1.5**] with lightweight, CSS-styled semantic HTML, the cost of serving each page would greatly diminish—and this would, in turn, increase the company's profits.



1.4

What Volkswagen (**www.volkswagen.com**) looks like—stark, compelling, beautiful.



1.5

Beautiful, that is, until you look under the hood. Peel back Volkswagen's skin (with your browser's "view source" feature), and you'll discover that the code and markup used to create this simple-looking website are unbelievably convoluted and perplexing.

So why hasn't Volkswagen made the switch? Presumably, the company wishes its site to look exactly the same in long-extinct browsers that don't support CSS as it does in modern, standards-savvy browsers that do. The irony is that no one using a ten-year-old browser is likely to care what Volkswagen's website looks like. And they're certainly not going to compare how the Volkswagen site looks in IE5 with how it looks in Firefox 3 or IE8. By constructing their pages using techniques outlined in this book, Volkswagen's design team could ensure that their content is accessible to all devices and browsers, even when niceties of the site design are not.

That a company as smart as Volkswagen misses this opportunity says everything you need to know about the entrenched mindset of brand managers who treat the web as if were a print ad, and of developers who hold "backward compatibility" in higher esteem than reason, usability, or their own profits.

The Hidden Cost of Bloated Markup

Suppose the code and markup on one old-school web page weighs in at 60K. Say that, by replacing outdated `font` tags and other presentational and proprietary junk with clean, structural markup and a few CSS rules, that same page can weigh 30K. (In our agency's practice, we can often replace 60K of markup with 22K or less. But let's go with this more conservative figure, which represents bandwidth savings of 50%.) Consider two typical scenarios, detailed next.

T1 Terminator

Scenario: A self-hosted business or public sector website serves a constant stream of visitors—several hundred at any given moment. After cutting its page weight in half by converting from presentational markup to lean, clean, structural XHTML, the organization saves $1,000 a month.

How it works: To serve its audience prior to the conversion, the self-hosted site requires four T1 lines, each of which is leased at a cost of U.S. $500 per month (a normal cost for a 1.544-megabit-per-second T1 line). After shaving file sizes by 50%, the organization finds it can get by just as effectively with two T1 lines, thus removing $1,000/month from its operating expenses. In addition to savings on bandwidth, there will also be fewer hardware expenses,

plus lowered storage costs and energy use for large data centers, both in running the machines and cooling the room. That's right: web standards can help save the earth.

The simpler the markup, the faster it's delivered to the user. The faster it's delivered, the less stress is placed on the server—and the fewer servers you need to buy, service, and replace. This is particularly true for servers that must cope with dynamic, database-driven content—that is, all commerce and most modern content sites (even most blogs).

Metered Megabytes

Scenario: As a commercially hosted site grows popular, its owners find themselves paying an unexpected file transfer penalty each month, to the tune of hundreds, or even thousands, of unexpected dollars. Cutting file sizes in half restores the monthly bill to a manageable and reasonable fee.

How it works: Many commercial hosting services allot their users a set amount of "free" file transfer bandwidth each month—say, up to 3GB. Stay below that number, and you'll pay your usual, monthly fee. Exceed it, and you must pay more. Sometimes *much* more.

In one infamous case, a hosting company slapped designer Al Sacui with $16,000 in additional fees after his noncommercial site, Nosepilot.com, exceeded its monthly file transfer allowance. It's an extreme case, and Sacui was able to avoid paying by proving that the host had changed the terms of service without notifying its customers, but only after a lengthy legal fight. Who wants to risk outrageous bills or protracted legal battles with an ornery hosting company?

Not every hosting company charges outrageous amounts for excess file transfers, of course. Whether your site is large or small, visited by millions or just a handful of community members; the smaller your files, the lower your bandwidth. (By the way, it's best to choose a hosting company that permits unlimited or "unmetered" file transfers rather than one that penalizes you for creating a popular site.)

Condensed Versus Compressed Markup

After delivering a lecture on web standards, I was approached by a developer who claimed that the bandwidth advantages of clean, well-structured markup didn't amount to a hill of beans for companies that compress their HTML.

In addition to *condensing* your markup by using semantic structures, you can digitally *compress* your markup in some server environments. For instance, the Apache web server includes a `mod_gzip` module that squeezes HTML on the server side. The HTML expands again in the user's browser.

The developer I spoke with gave this example: If Amazon.com wastes 40K on outdated font tags and other junk but uses `mod_gzip` to compress it down to 20K, Amazon's bloated markup represents less of an expense than my lecture (and this book) would suggest.

As it turns out, Amazon does not use `mod_gzip`. In fact, the tool is used little on the commercial web, possibly due to the extra load required to compress pages before sending them. But that quibble aside, the smaller the file, the smaller it will compress. If you save money by compressing an 80K page down to 40K, you'll save even more by compressing a 40K page down to 20K. Savings in any given page-viewing session might seem small, but their value is cumulative. Over time, they can substantially reduce operating costs.

Bandwidth savings are only one advantage to writing clean, well-structured markup, but they're one that accountants and clients appreciate, and they hold as true for those who compress their HTML as they do for the rest of us.

Backward Compatibility Is a Lie

What do developers mean by "backward compatibility"? If you ask them, they'll say they mean "supporting all our users." And who could argue with a sentiment like that?

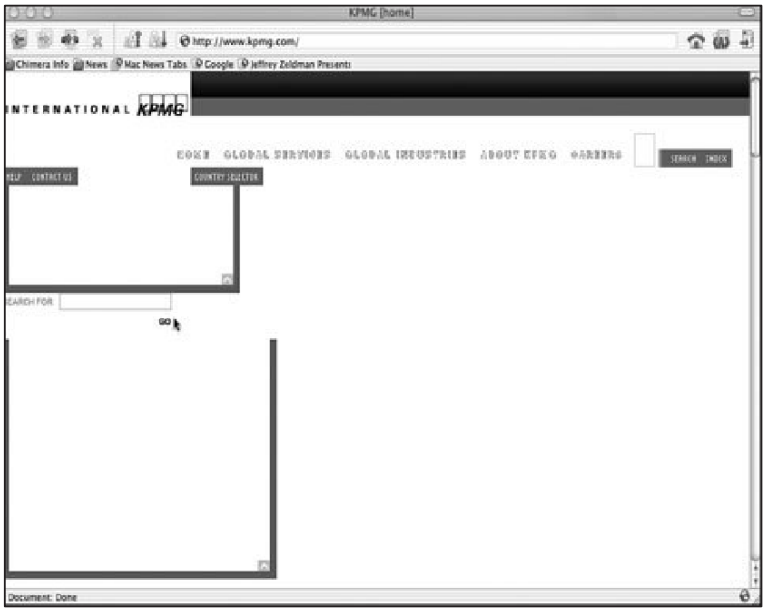
In practice, however, backward compatibility means using nonstandard, proprietary (or deprecated) markup and code to ensure that every visitor has the same experience, whether they're sporting IE2 or Opera 10. Backward compatibility sounds great in theory. But the cost is too high and the practice has always been based on a lie. The truth is, there is no real backward compatibility. There is always a cut-off point. For instance, neither Mosaic (the first visual browser) nor Netscape 1.0 supports HTML table-based layouts. By definition, then, those who use these ancient browsers cannot possibly have the same visual experience as folks who view the web through slightly less ancient browsers like Netscape 1.1 or MSIE2.

Developers and clients who claim to strive for backward compatibility inevitably specify a "baseline browser" such as Internet Explorer 6 and agree that that's the earliest browser their site will support. (In this scenario, Internet Explorer 5.5 users are out of luck.) To achieve baseline browser support, developers layer their markup with browser-specific, nonstandard hacks, write multiple scripts to accommodate "supported" browsers, and use UA sniffing to feed each browser the code it likes best. In so doing, they further increase the girth of their pages, pump up the load on their servers, and ensure that the race against obsolescence will continue.

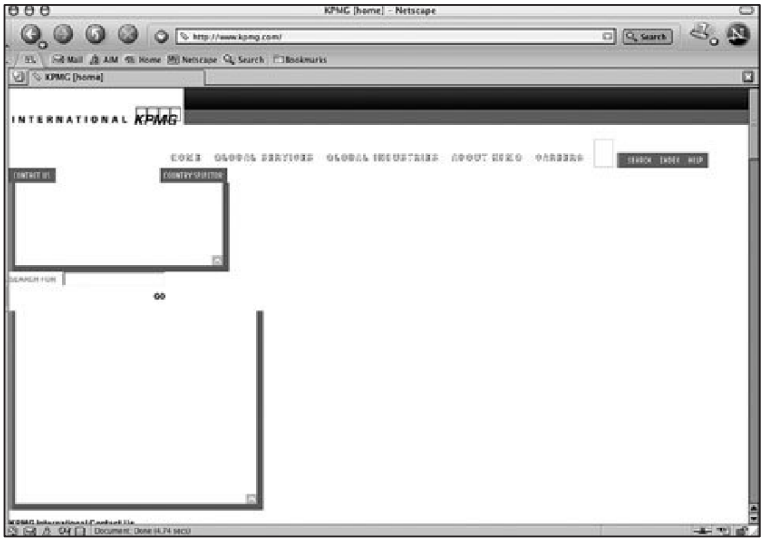
Blocking Users Is Bad for Business

While some companies undercut their own profitability trying to ensure that even the oldest browsers display their sites exactly as new browsers do, others have decided that only one browser matters. In a misguided effort to reduce expenses, many sites are designed to work only in Internet Explorer, and sometimes only on the Windows platform, thus locking out 15–25% of their potential visitors and customers [1.6, 1.7, 1.8, 1.9, 1.10].

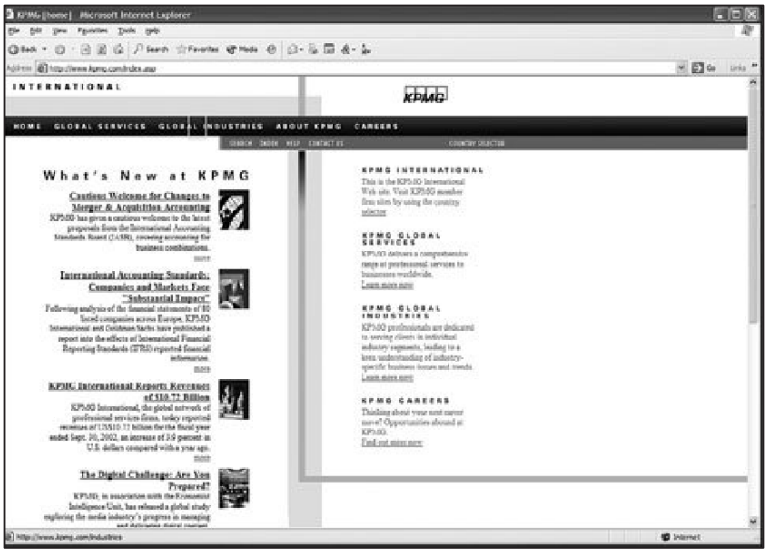
1.6
The home page of KPMG (www.kpmg.com), circa 2003, as seen in Netscape Navigator. Or rather, as not seen in Navigator, thanks to IE-only code.



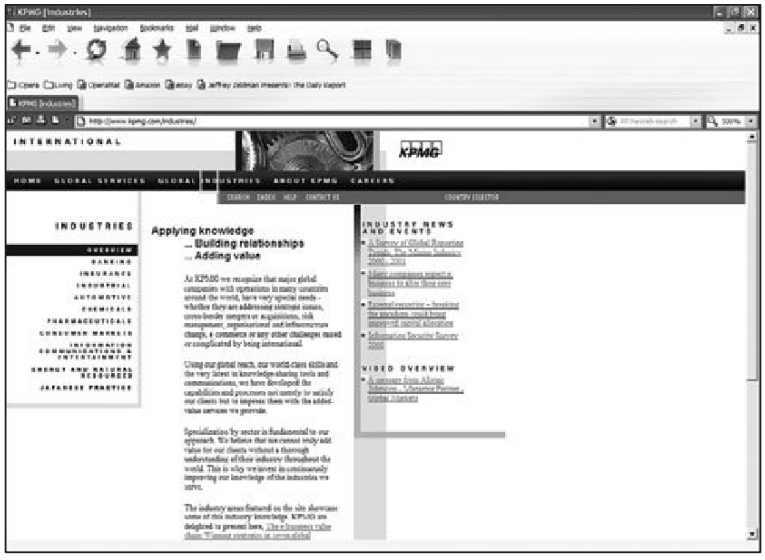
1.7
The site was equally useless in Netscape 7.



1.8
Well, if the site was for IE only, how did it work in IE5/Mac? Apparently, not well at all.

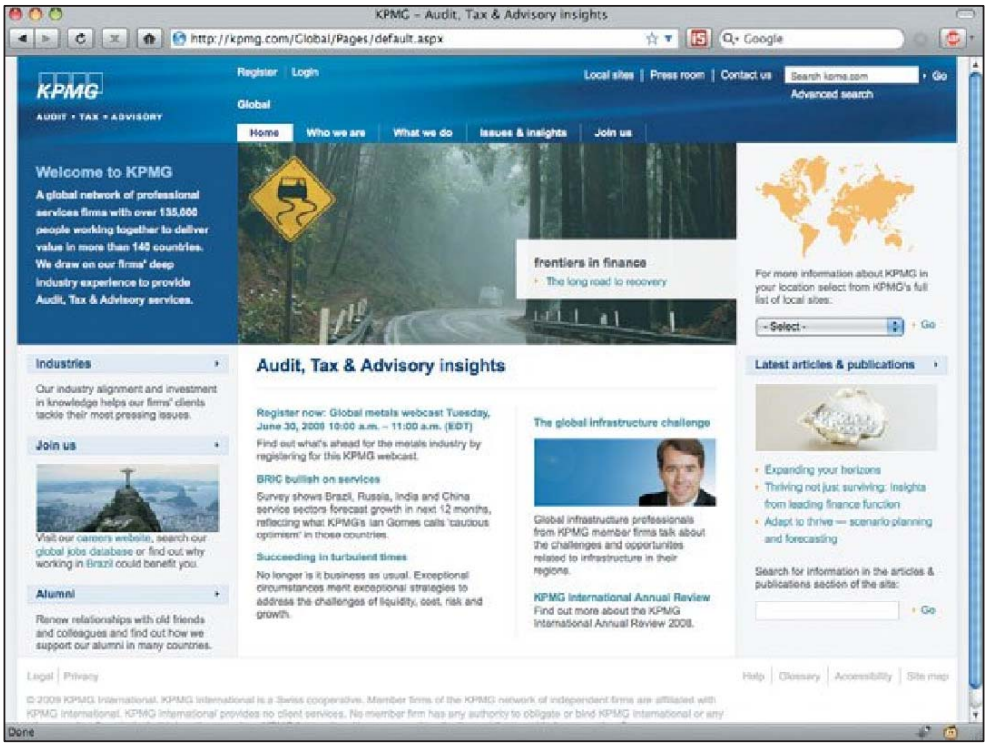


1.9
The same site as seen in IE6/Windows, where it finally deigned to work.



1.10
To be fair, the site kind of worked in Opera 7 for Windows when Opera was set to identify itself as IE. (When Opera identified itself as Opera, the site failed.)

1.11
After a redesign, today KPMG's site looks OK and works right across multiple browsers and platforms. There's still much room for improvement under the hood, but updated, non-IE-only markup makes all the difference.



I won't pretend to understand the business model of a company that would say no to up to a quarter of its potential customers. And the sheer number of customers lost by this myopic approach should boggle the mind of any rational business owner or noncorporate agency with a mandate to serve the public. According to recent statistics (`www.internetworldstats.com`), nearly 1.6 billion people used the web in May 2009. You do the math.

Say you don't mind losing up to 25% of the people who choose to visit your site. The "IE-only" approach still makes no sense because there's no guarantee that IE (or even desktop browsers as a category) will continue to dominate web space. As I write these words, Firefox continues to take market share away from IE, and ever more people are getting their internet fix via mobile devices powered by Webkit and Opera Mini. As ubiquitous computing gains acceptance and creates new markets, the notion of designing to the quirks of *any* individual desktop browser seems more and more 20th century and less and less intelligent.

Besides, as this book will show, standards make it possible to design for *all* browsers and devices as easily and quickly as for just one.

In our efforts to deliver identical experiences across incompatible browsing environments, we've lost sight of its true potential as a rich and multilayered medium accessible to all. We lost it when designers and developers, scrambling to keep up with production demands during short-lived internet booms, learned deeply flawed ways of creating sites, thus bringing us to our current pass, whose name is obsolescence.

But the obsolescent period of web development is dying as you read these words, taking countless sites down with it. If you own, manage, design, or build websites, the bell tolls for you.

The Cure

"Write once, publish everywhere," the promise of web standards, is more than wishful thinking; it is being achieved today, using methods we'll explore in this book. Although today's leading browsers finally support these standards and methods, the message has not yet reached many designers and developers, and new sites are still being built on the quicksand of nonstandard markup and code. This book hopes to change that.

Crafted by the members of the World Wide Web Consortium (W3C) and other standards bodies and supported in all post-2000 browsers, technologies like CSS, XHTML, standard JavaScript, and the W3C DOM enable designers to do the following:

- Attain more precise control over layout, placement, and typography in graphical desktop browsers while allowing users to modify the presentation to suit their needs.
- Develop sophisticated behaviors that work across multiple browsers and platforms.
- Comply with accessibility laws and guidelines without sacrificing beauty, performance, or sophistication.
- Redesign in hours instead of days or weeks, reducing costs and eliminating grunt work.
- Support multiple browsers without the hassle and expense of creating separate versions, and often with little or no code forking.

- Support nontraditional and emerging devices, from wireless gadgets and smart phones to Braille output devices and screen readers used by those with disabilities—again without the hassle and expense of creating separate versions.
- Deliver sophisticated printed versions of any web page, often without creating separate "printer-friendly" page versions or relying on expensive proprietary publishing systems to create such versions.
- Transition from the tag soup of the past to the real semantic web of the present and future.
- Ensure that sites so designed and built will work correctly in today's standards-compliant browsers and perform acceptably in old browsers, even if they don't render pixel-for-pixel the same way in old browsers as they do in newer ones.
- Ensure that sites so designed will continue to work in tomorrow's browsers and devices, including devices not yet built or even imagined. This is the promise of forward compatibility.
- … and more, as this book will show.

Before we can learn how standards achieve these goals, we must examine the old-school methods they're intended to replace and find out exactly how the old techniques perpetuate the cycle of obsolescence. Chapter 2 reveals all.

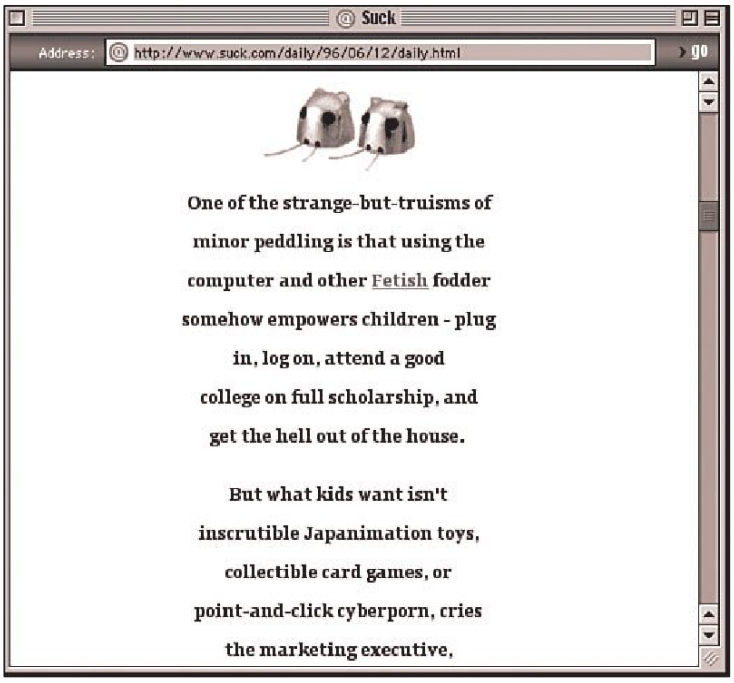
Not in the mood for a history lesson? Skip ahead to Chapter 3 for a quick blast of fresh air.

chapter two

Designing and Building with Standards

How did designers and developers produce sites before web standards were created and before browsers supported them? Any which way they could. Consider the late Suck.com, one of the web's earliest and wittiest independent periodicals [**2.1**]. Suck possessed a sharp writing style and had the smarts to slap its daily content right on the front page, where readers couldn't miss it. It sounds obvious today, when everyone and their brother has a blog featuring continually updated front-page stories, but in the mid-1990s when Suck debuted, most sites buried their content behind splash screens, welcome pages, mission statements, and confusing "Table of Contents" pages.

Suck's straight-ahead emphasis on text felt refreshingly direct in an era when most commercial sites wrapped their content in overwrought, literal metaphors ("Step Up To Our Ticket Counter," "Enter the Web Goddess's Lair"). Likewise, Suck's spare, minimalist look and feel stood out at a time when many sites were over-designed

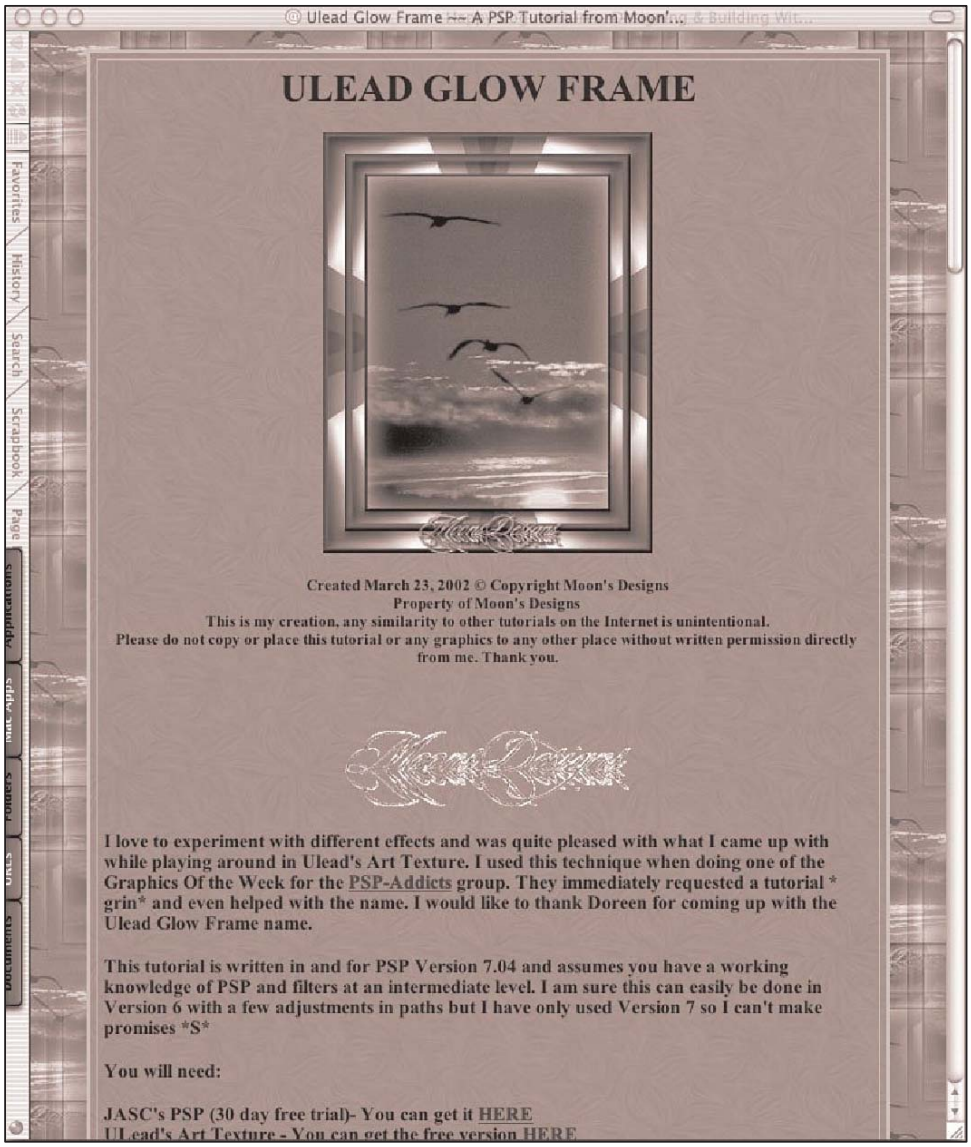


2.1
Suck didn't. A decidedly bright site from the pioneering days of the commercial web (**www.suck.com**).

exercises in metallic bevels and high-tech, Gothy glows, or non-designed messes flung together by systems administrators and self-taught HTML *auteurs*. Many site builders at the time used every primitive device Netscape 1.1 offered the would-be layout artist, from the repeating background tile to the proprietary `center` tag, and some web pages still showcase these techniques [**2.2**]. In a web where more was more, Suck stood out by daring to do less.

To achieve Suck's distinctively spare, content-focused appearance, co-creators Carl Steadman and Joey Anuff had to jump through hoops. HTML lacked design tools, and for good reason: as conceived by Tim Berners-Lee, the physicist who invented the web, HTML was a structured markup language (`www.w3.org/MarkUp/html-spec`) derived from SGML, not a design language like Adobe's PostScript or the Cascading Style Sheets standard. (CSS had yet to be approved as a W3C standard, and once approved, it would take four long years before browsers supported the standard with anything close to accuracy.)

So how did Steadman and Anuff control their site's presentation? They did it with creativity, invention, and many rolls of digital duct tape.



2.2
The Ulead Glow Frame tutorial at Moon's Designs exemplifies mid-1990s web design (**`www.moonsdesigns.com/tutorials/frames/glow.html`**). Content is centered in an HTML table that is also centered. One repeating background tile is applied to the table and another to the page that contains it.

Jumping Through Hoops

To create the look of Suck, Steadman and Anuff wrote a Perl script that counted the characters in their text, inserting a `p` paragraph tag as a carriage return when a set number of characters had elapsed:

```
<p>One of the strange-but-truisms of
<p>minor peddling is that using the
<p>computer and other Fetish fodder
<p>somehow empowers children - plug
<p>in, log on, attend a good
<p>college on full scholarship, and
<p>get the hell out of the house.
```

The entire production was then wrapped in "typewriter" `tt` tags to force early graphical browsers (mainly Netscape 1.1) into styling the text in a monospace font like Courier or Monaco.

The result was rudimentary typographic control and a brute-force simulation of leading. Such HTML hacks offered the only way to achieve design effects in 1995. (The visual example shown in Figure **2.1** is from 1996, after a somewhat more graphic-intensive Suck redesign—still fairly minimalist. The original design is lost to the sands of internet time.)

Equally creative methods of forcing HTML to produce layout effects were widely practiced by web designers and were taught in early web design bibles by authors like Lynda Weinman and David Siegel—pioneers to whom all web designers owe a lasting debt. The creators of HTML clucked their tongues at this wholesale deformation of HTML, but designers had no choice as clients clamored for good-looking web presences.

Many designers still use methods like these, and many books still teach these outdated—and in today's web, counterproductive—techniques. One otherwise excellent recent web design book straight-facedly advised its readers to control typography with `font` tags and "HTML scripts." `font` tags have long been deprecated (W3C parlance for "please don't use this old junk") and HTML is not a scripting language. But bad or nonsensical advice of this kind continues to appear in best-selling web design books, perpetuating folly and ignorance.

The Cost of Design Before Standards

By creatively manipulating HTML, Suck had achieved a distinctive look, but at a double cost: the site excluded some readers and was tough for its creators to update.

In early Mom-and-Pop screen readers (audio browser interfaces for the visually disabled), the voice that read Suck's text aloud would pause every few words in deference to the ceaseless barrage of paragraph tags, disrupting the flow of Suck's brilliantly argumentative editorials:

```
One of the strange-but-truisms of ... [annoying pause]
minor peddling is that using the ... [annoying pause]
computer and other Fetish fodder ... [annoying pause]
somehow empowers children—plug ... [annoying pause]
in, log on, attend a good ... [annoying pause]
college on full scholarship, and ... [annoying pause]
get the hell out of the house.
```

Hard enough to parse under ideal conditions, Suck's convoluted sentence structures devolved into Zen incomprehensibility when interrupted by nonsemantic paragraph tags. These audio hiccups presented an insurmountable comprehension problem for screen reader users and made the site unusable to them.

If the HTML tricks that made Suck's layout work in graphical browsers thwarted an unknown number of readers, they also created a problem for Suck's authors each time they updated the site.

Because their design depended on Perl and HTML hacks, it was impossible to template. Hours of production work had to go into each of Suck's daily installments. As the site's popularity mushroomed, eventually leading to a corporate buyout, its creators were forced to hire not only additional writers but also a team of producers. The manual labor involved in Suck's production was inconsistent with the need to publish on a daily basis.

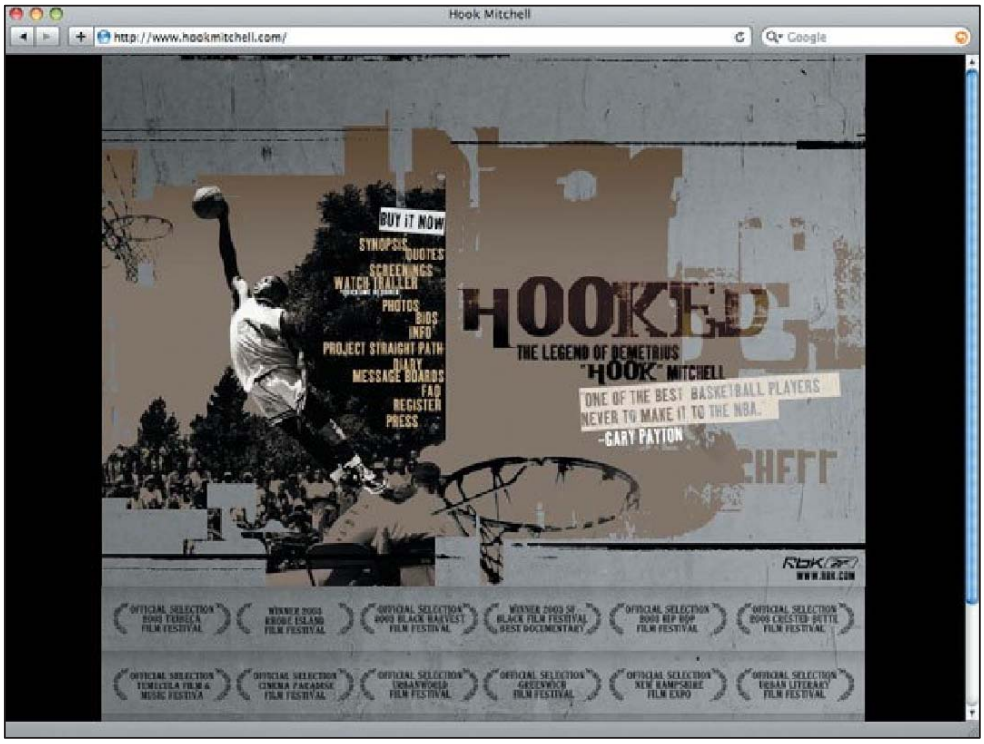
In a more perfect world, these difficulties would have been confined to the era in which they occurred. They would be anecdotes of early commercial web development. While admiring pioneering designers' ingenuity, we'd smile at the thought that development had ever been so screwy. But in spite of the emergence of standards, most commercial production still relies on bizarrely

labor-intensive workarounds and hacks and continues to suffer from the problems these methods engender. The practice is so widespread that many designers and developers never even stop to think about it.

Modern Site, Ancient Ways

Leap from 1995 to 2003 and consider a little movie called *Hooked: The Legend of Demetrius "Hook" Mitchell*. While *Hooked* might not be on your top ten list, the independent documentary was featured at over twenty film festivals, and garnered not a few awards. As part of the film's marketing campaign, the makers of *Hooked* built a website to help them better promote the film (`www.hookmitchell.com`) [**2.3**]. It was (and is) a well-designed site, but was assembled using labor-intensive table layout techniques that had outlived their usefulness on the modern web.

Hooked told the story of a legendary amateur basketball player whose hard life on the streets eventually derailed his career. And while the site's design effectively captured the film's grittiness, the *Hooked* website had plenty of grime under the hood, too. Its table-heavy markup and spacer-GIF–ridden layout had more than a few drawbacks:



2.3
The Hook Mitchell website (**`www.hookmitchell.com`**), built in 2003 with 1998-era markup. A compelling design powered by contorted code.

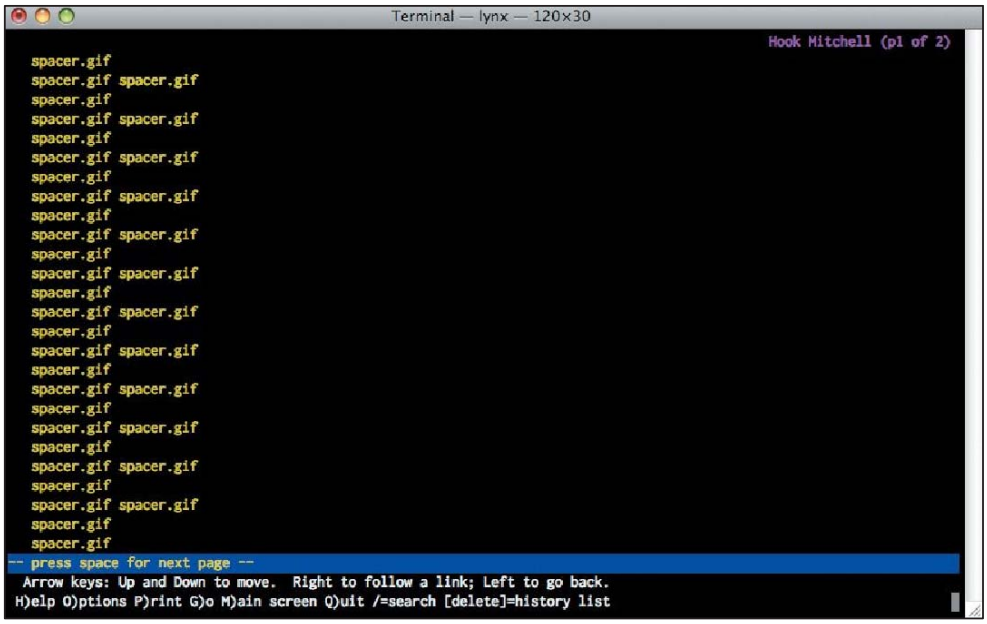
- If the filmmakers needed to make any changes to the site—even something as small as the addition of a single link—the markup for the homepage would need to be completely scrapped and rebuilt. Changing the size of even one component image would cause the entire HTML table that combined the various image slices [**2.4**] to burst apart.
- Thus, even the smallest changes to the site would incur a significant cost. The graphics would have to be redesigned, resliced, and reoptimized, and the table markup would have to be rewritten, along with the JavaScript that powered the rollovers. When a task as basic as adding a link requires hours of work, something is wrong.
- The site excluded many potential visitors. As implemented, the site was inaccessible to users of screen readers, text browsers, and mobile devices. Viewed in a nongraphical browsing environment [**2.5**], the page's total content read as follows:

```
spacer.gif
spacer.gif spacer.gif
spacer.gif
spacer.gif spacer.gif
[And so forth...]
spacer.gif spacer.gif
bottom.jpg
```

2.4
The same film site, with table borders activated to reveal construction methods (and potential maintenance headaches).

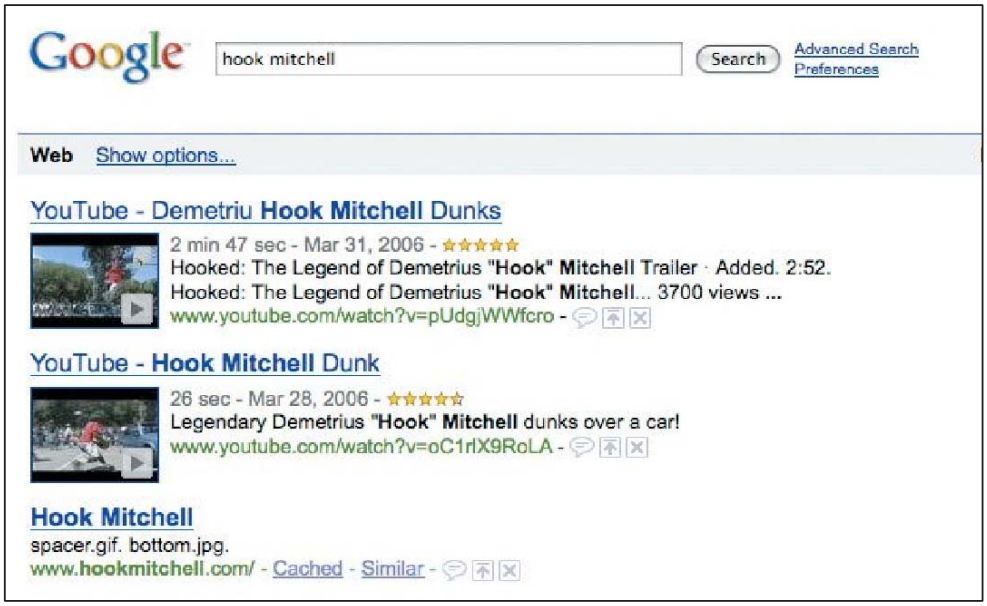
2.5

In a text browser like Lynx (**lynx.isc.org**), the Hook Mitchell website provides no information: images are missing their required `alt` attributes, so filenames are displayed instead of human-readable link titles. To see how your web pages appear in a nongraphical environment, test them in Lynx, a Lynx emulator, or a screen reader like JAWS or OS X's VoiceOver. A free online Lynx emulator (**www.delorie.com/web/lynxview.html**) is available but can be used only on sites for which you are the "webmaster."



2.6

Because there is no content to be found on the *Hooked* homepage beyond nonsensical image filenames, its search engine results are polluted, and its ranking no doubt suffers. If you were searching for information on the movie, would "spacer.gif. bottom.jpg" compel you to visit? (I didn't think so.)



Without a lick of nongraphical content to be found on their homepage, even the site's search results return nothing useful about the movie it was designed to promote [**2.6**]. Was "spacer.gif" the message that the producers of *Hooked* hoped to convey to the site's visitors? We doubt it.

I'm not saying images are bad, or intricate design is a liability. To the contrary, images are vital, beauty is needed, and the *Hooked* site is indeed a well-crafted aesthetic online experience. But that experience need not have been inaccessible.

Though visually attractive, the *Hooked* site epitomizes the secret shame of old-school page layouts: their code reveals a hidden ugliness in the form of inaccessibility and a reliance on inefficient and expensive production techniques. Thankfully, much of the web has long abandoned these techniques in favor of a more standards-driven approach. But despite the progress we've made, many sites are still being built with outdated techniques (see sidebar "No Room at the Inn"). Those of us forced to rely on broken markup will find that our carefully constructed layouts break when the client wishes to make changes, as all clients do all the time. We either bill the client or eat the cost—or implement visually confusing workarounds that damage the site's credibility and usefulness. (For example, we might add three plain links to the top of the *Hooked* site to make newly added sections accessible, but this addition would degrade the site's carefully controlled aesthetic effect, and could confuse users as well.)

No Room at the Inn

Sadly, nonstandard production techniques like those used to build the Hook Mitchell site are still in use today.

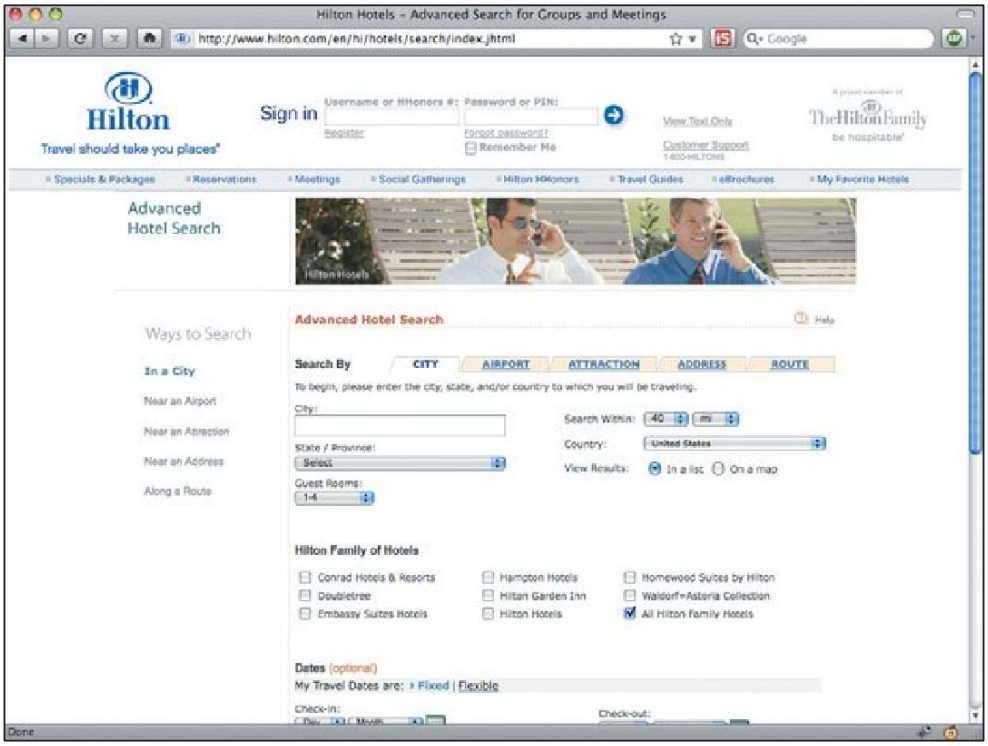
Onward but not upward we fly to present day, to Hilton's hotel search (`www.hilton.com/en/hi/hotels/search/index.jhtml`). Looking to make a reservation? Good luck. Over a decade after the introduction of CSS and in the era of the standards-compliant browser, Hilton's search form still relies on nonsemantic table layouts, obsolete font tags, 1995-style spacer GIF images, and other long-stale crusts. As if all that nonstructural markup didn't waste enough bandwidth, whopping JavaScripts embedded into each page (instead of being externalized) further swell the site's bloat, slowing

(continues on next page)

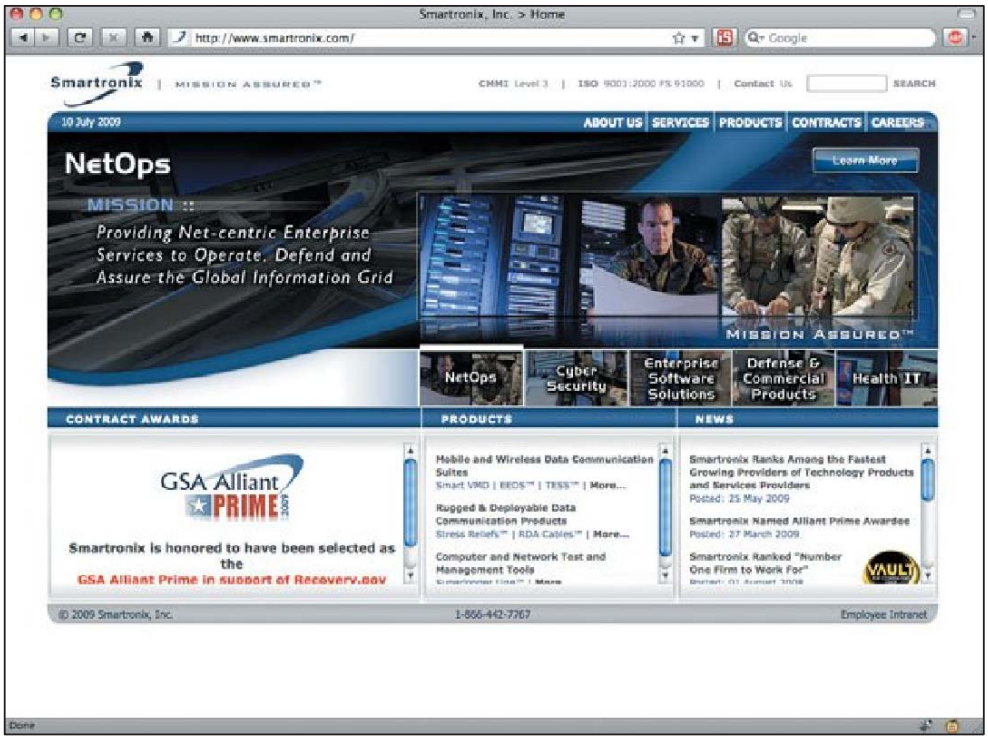
(continued from previous page)

page display. Worst of all, the content is an accessibility nightmare. Missing `alt` attributes on images will cause audio browsers to read gibberish. And if JavaScript is disabled, large swaths of the page will cease to work. So if you don't meet Hilton's stringent technical requirements, you'll have to find another hotel to put you up.

Model of worst practices though it may be, Hilton's poorly produced search engine is scarcely unique. Plenty of dinosaurs just as big and just as dumb still roam the cooling planes of cyberspace [**2.8**]. All the more reason to advocate for smarter techniques based on web standards, so that mistakes like these stay in the past.



2.7
Hilton's hotel search (**`www.hilton.com/en/hi/hotels/search/index.jhtml`**), directly accessible from hilton.com, is a simple form weighted down by overwrought presentational markup: spacer GIFs, nested tables, and—can it be?—heaps of obsolete tags. So make your reservation, and 1999'll leave a light on for you.



2.8
In 2009, Smartronix, Inc. (**www.smartronix.com**) was awarded $9.5 million (**www.bizjournals.com/washington/stories/2009/07/06/daily78.html**) to oversee the redesign of recovery.gov, a site dedicated to preventing government waste. Judging from Smartronix's invalid markup, much of that budget will be spent on bandwidth.

FIGHT BACK!

Tired of outdated, non-standards-based redesigns that make sites harder for customers to use while also making web standards harder to sell to your clients? Accessibility expert and standardista *Joe Clark, whose name will pop up more than once in this book, created a Failed Redesigns campaign (*`blog.fawny.org/category/web-standards/failed-redesigns`*) to spread standards awareness while shaming those who produce new or redesigned sites that act as if "the 21st century is frozen in the amber of the year 1999."*

As the creators of the sites above might have learned by now, desktop-centric layouts don't travel well. They might look fine in aging browsers under "average" conditions. But beyond those borders, our sites might cease to communicate. At the very least, such inaccessibility cuts off potential customers.

When they drowned their content in tag soup and locked out users and browsers, these designers were using long-established (if painfully outdated) techniques. But why would they do so? It may be a matter of education: perhaps they were using the most modern technologies they were familiar with. The designers may also have been hampered by outdated content management systems, or by a client's equally outmoded respect for long-dead browsers.

But whatever the motive, the problems generated by these techniques are not unique to Hilton or Hook Mitchell. They're the problems faced by any site crafted to the quirks of a few visual browsers instead of being designed to facilitate universal access via web standards. They are also the problems of any site that yokes presentation to structure by forcing HTML to deliver layouts, a job it was not built to do.

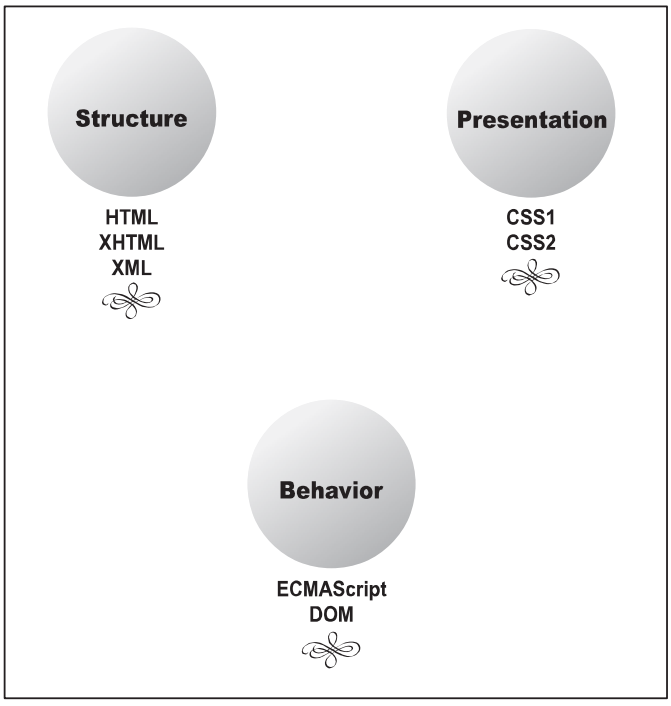
Fortunately, there's another way to design and build websites—a way that solves the problems created by old-school methods without sacrificing the aesthetic and branding benefits the old methods deliver: web standards.

The Trinity of Web Standards

Figure **2.9** indicates how web standards solve the problems we've been discussing by breaking any web page into three separate components: structure, presentation, and behavior.

Structure

A *markup language* (HTML 4.01: `www.w3.org/TR/html401`; XHTML 1.0: `www.w3.org/TR/xhtml1`) contains text data formatted according to its structural (semantic) meaning: headline, secondary headline, paragraph, numbered list, definition list, and so on.



2.9 Structure, presentation, and behavior: the three components of any web page in the world of web standards.

On the web, this text would likely be part of a definition list `dl`. The subhead, "Structure," would be marked up as a definition title `dt`. The paragraphs you're now reading would be wrapped in definition data `dd` tags:

```
<dl>
<dt>Structure</dt>
<dd>A <em>markup language</em> (<a href="http:// www.w3.org/
TR/html401">HTML 4.01</a>; <a href="http://www.w3.org/TR/
xhtml1">XHTML 1.0</a>) contains text data formatted according
to its structural (semantic) meaning: headline, secondary
headline, paragraph, numbered list, definition list, and so
on.</dd>
<dd>On the web, this text would likely be part of a definition
list &lt;dl&gt;. The subhead, &#8220;Structure,&#8221; would be
marked up as a definition title. The paragraphs you&#8217;re
now reading would be wrapped in definition data tags.
</dd>
</dl>
```

Alternately, if it made better semantic sense for your document, the two paragraphs might simply follow a single headline element:

```
<h3>Structure</h3>
<p>A <em>markup language</em> (<a href="http:// www.w3.org/
TR/html401">HTML 4.01</a>; <a href="http://www.w3.org/TR/
xhtml1">XHTML 1.0</a>) contains text data formatted according
to its structural meaning: headline, secondary headline,
paragraph, numbered list, definition list, and so on.</p>
<p>On the web, this text would likely be part of a definition
list &lt;dl&gt;. The subhead, &#8220;Structure,&#8221; would
be marked up as a definition title. The paragraphs you&#8217;re
now reading would be wrapped in definition data tags.</p>
```

THE 411 ON ’

If you're one of the five people who will actually take the time to read the text in the XHTML examples, you might wonder what `&#8217;` *means. Quite simply, it is the standard Unicode character sequence encoding a typographically correct apostrophe. Likewise,* `&#8220;` *is Unicode for typographically correct double open quotation marks, while* `&#8221;` *is the code that closes those double quotation marks.*

XML (`www.w3.org/TR/REC-xml`), the extensible markup language, provides considerably more options than this, but for now we'll limit ourselves to XHTML 1.0, a transitional markup language and stable W3C standard since 2000 that works just like HTML in nearly every browser or internet device.

When authored correctly (containing no errors, and no illegal tags or attributes), standards-compliant XHTML or HTML is completely portable. It works in web browsers, screen readers, text browsers, wireless devices—you name it.

The markup can also contain additional structures deemed necessary by the designer. For instance, content and navigation might be marked as such and wrapped in appropriately labeled tags:

```
<div id="content">[Your content here.]</div>
<ul id="navigation">[Your navigational menu here.]</ul>
```

Markup can also contain embedded objects such as images, Flash presentations, or video clips, along with tags and attributes that present text equivalents for those who cannot view these objects in their browsing environment.

Valid? Semantic? Say what?

- Markup is *valid* when it contains no errors (example: forgetting to close a tag's bracket) and no illegal tags or attributes (example: the `height` attribute, applied to a table, is not legal in XHTML). Validation can be tested via free online software (`validator.w3.org`).
- Markup is *semantic* when tags are chosen according to what they mean. For example, tagging a headline `h1` because it is the most important headline on the page is a semantic authoring practice. Tagging a headline `h1` "to make it look big" is not. In this book I sometimes use the phrase "structural markup" to mean pretty much the same thing as "semantic markup." ("Structural markup" takes its name specifically from the idea that each web document has an outline-like structure.)

A web page can be valid yet not be semantic. For example, an HTML page could be layed out with table cells and no structural markup. If the table markup contains no errors and no illegal tags or attributes, the page is valid. Likewise, a page can be semantic and invalid. Typically, professionals who practice standards-based design strive to create pages whose markup is both valid and semantic.

Presentation

Presentation languages (CSS 2.1: `www.w3.org/TR/CSS21`; CSS 3: `www.w3.org/Style/CSS/current-work`) format the web page, controlling typography, placement, color, and so on. CSS can take the place of old-school HTML table layouts. In all cases, it replaces nonstandard font tags and bandwidth-wasting, outdated junk like this:

```
<td bgcolor="#FFCC00" align="left" valign="top"><br><br><br>&nb
sp;</td>
```

We could strike such junk from our markup by applying the `border` property to the element via a single CSS rule.

Because presentation is separated from structure, it is possible to change one without negatively affecting the other. For instance, you can apply the same layout to numerous pages or make changes to text and links without breaking the layout. You or your clients are free to change the XHTML at any time without fear of breaking the layout because the text is just text; it does not serve double duty as a design language.

Likewise, you can change the layout without touching the markup. Have readers complained that your site's typeface is too small? Change a rule in the global style sheet, and the entire site will reflect these changes instantly. Need a printer-friendly version? Write a print style sheet, and your pages will print beautifully, regardless of how they appear on the screen.

Behavior

A standard object model (the W3C DOM at `www.w3.org/DOM`) works with CSS, XHTML, and ECMAScript 262 (`www.ecma-international.org/publications/standards/Ecma-262.htm`), the standard version of JavaScript, enabling you to create sophisticated behaviors and effects that work across multiple platforms and browsers. No more browser-specific scripting.

Standards In Action

If Suck were produced today, web standards like (X)HTML and CSS would allow the staff to concentrate on writing. A basic (X)HTML template would semantically describe the document. CSS would control the look and feel without requiring additional design work for every issue. Paragraph tags would denote the beginnings and ends of paragraphs rather than force vertical gaps between each line of text. (CSS would do that job.)

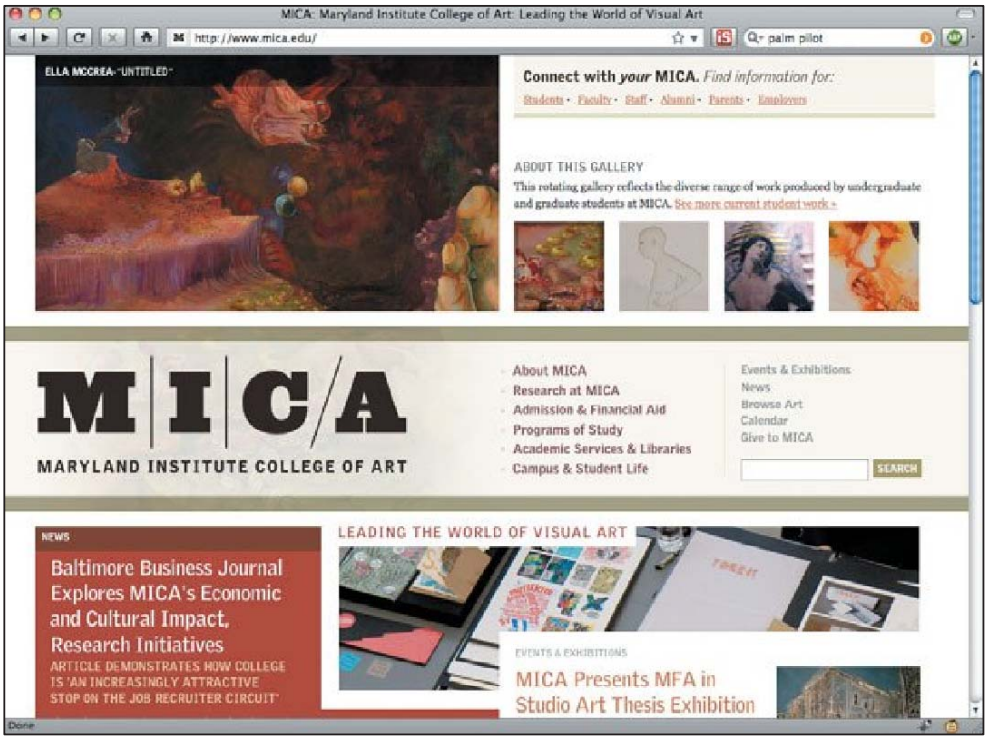
In graphical browsers like IE, Firefox, Opera, and Safari, style sheets would ensure that Suck looked as its designer intended. Semantic markup would deliver Suck's content not only to these browsers but also to mobile devices, screen readers, and text browsers without the accompanying hiccups of fake paragraphs and similar hostages to markup-as-a-design-tool.

As a content-focused site, Suck.com would be a prime candidate for a strict XHTML/CSS makeover in which substance and style would be delivered via the appropriate technology: CSS for layout and XHTML for structured content.

Similarly, the Hook Mitchell site could deliver their layout with CSS, conserving bandwidth while enabling the design team to change one section of the page without recooking the whole tamale. The site's designers could use the CSS `background` property to position its primary image as a single JPEG or PNG file instead of a dozen image slices [2.4] and could easily overlay one or more menu graphics using any of several time-tested CSS positioning methods.

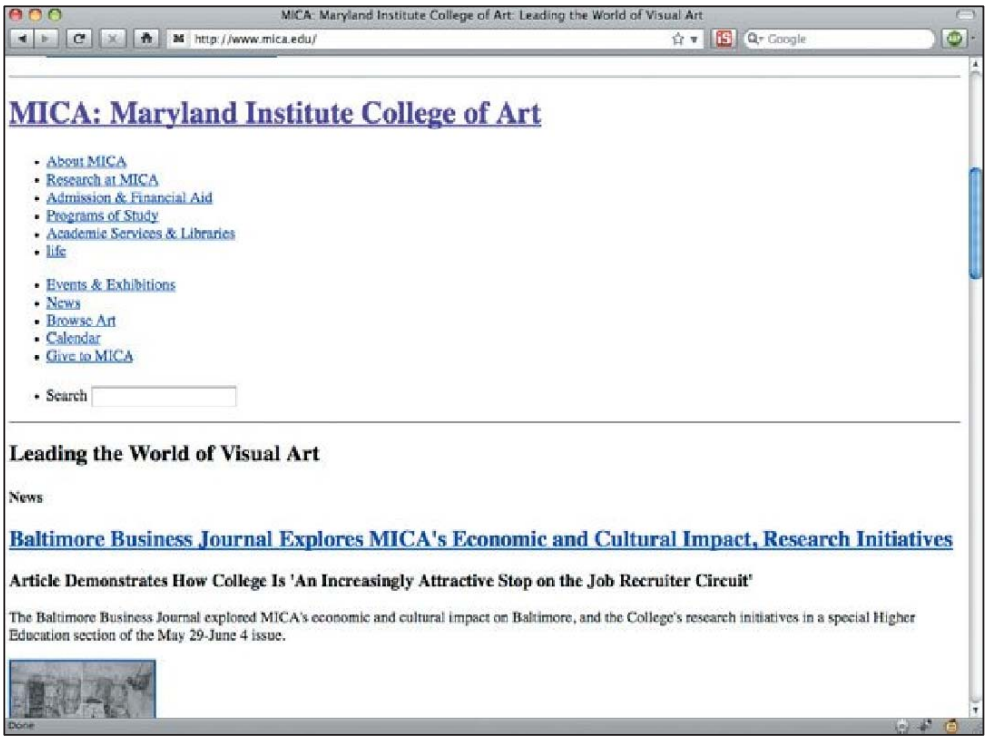
In addition to abstracting the presentational logic out of their markup and into their CSS, the designers could also use valid XHTML or HTML and accessibility attributes such as `alt` and `title` to ensure that the site's content would be accessible to all instead of meaningless to many [2.5]. And a single style sheet could control the design on countless interior pages, lowering maintenance overhead, as well as the cost of future design tweaks.

Thankfully, since the first edition of this book went to press, countless sites have embraced these technologies. Consider the website for the Maryland Institute College of Art (MICA), a recent redesign by Happy Cog. Underneath its striking design you'll find some equally attractive code, as MICA uses web standards to deliver its content to all [**2.10**, **2.11**].



2.10

The homepage for MICA (`www.mica.edu`), an art school that places the striking work of its students front and center. And web standards like XHTML and CSS make it happen.



2.11

With style sheets disabled, MICA's content is still perfectly accessible, its semantically rich markup easily read. The experience in a text-only browser would be similar.

The Web Standards Project: Portability in Action

The Web Standards Project (WaSP) [**2.11**] launched in 1998 to persuade Netscape, Microsoft, and other browser makers to thoroughly support the standards discussed in this book. It took time, persistence, and strategy (aka yelling, whining, and pleading), but eventually browser makers bought into WaSP's view that interoperability via common standards was an absolute necessity if the web was to move forward.

LIP SERVICE...

One of the ironies of the struggle for standards compliance in browsers was that Microsoft, a W3C member that has contributed significantly to the creation of web standards, had to be bullied into fully supporting the very technologies it helped to create. Go figure.

After browsers finally began meaningfully supporting standards, The Web Standards Project relaunched in 2002 to encourage designers and developers to learn about and harness the power of these hard-won technologies. To denote the enlargement of the group's mission from bully pulpit to educational resource, the site was rewritten and redesigned.

As expected, the site looked nice in standards-compliant browsers [**2.12**]. It also looked acceptable in older, less-compliant browsers [**2.13**]. But the site transcended the PC-based browsing space without requiring additional or alternative markup, code, or device detection. (Look, Ma, no versions!)

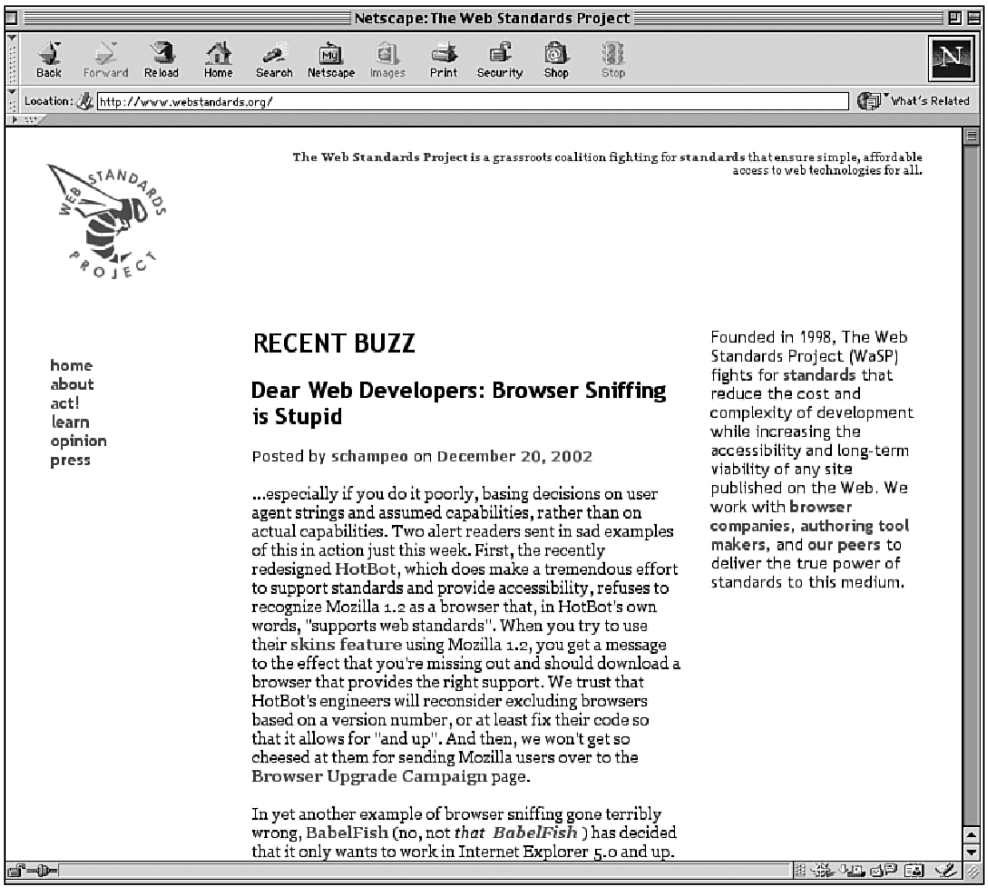
One Document Serves All

The Web Standards Project was built with XHTML 1.0 Strict. CSS was used for layout. There is no Palm version or WAP version, although creating such versions was common practice at the time. Multiple versions were not needed; when you design and build with standards, one document serves all.

Figure **2.14** shows `webstandards.org` as seen in a Palm Pilot. Figure **2.15** shows how it looks in Microsoft's PocketPC. Most uncannily of all, Figure **2.16** shows the site working just as fine as you please on a Newton handheld, Apple's long-discontinued proto-PDA. (Think of it as the iPhone's great-grandfather.) Grant Hutchinson, who captured the Newton screenshot, told us: "There's nothing like viewing a modern site using a piecemeal browser on a vintage operating system."

2.12

The Web Standards Project's CSS-powered homepage in 2002, as seen in an early version of the Camino (Mozilla) browser for Mac OS X (**www.webstandards.org**).

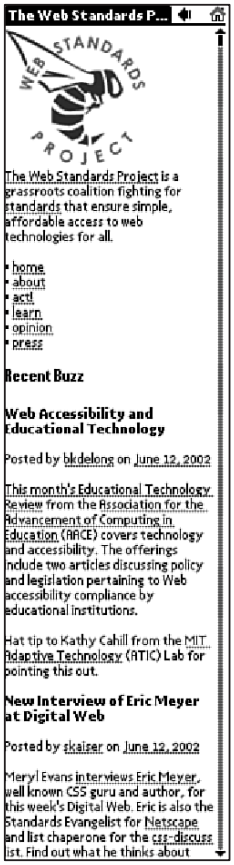


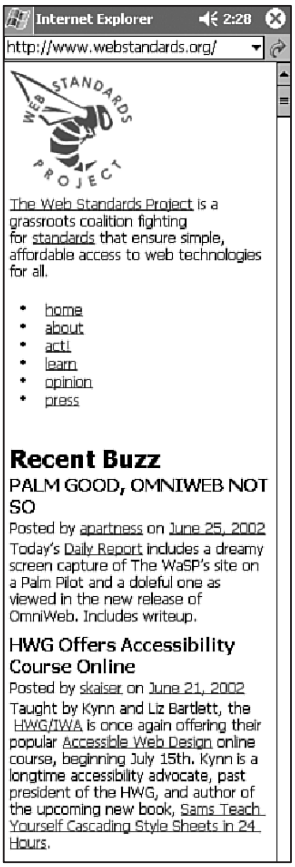
2.13

The same site looks decent and works acceptably in creaky old Netscape 4, our poster child for non-standards-compliance. A special "Netscape 4 version" was not needed.

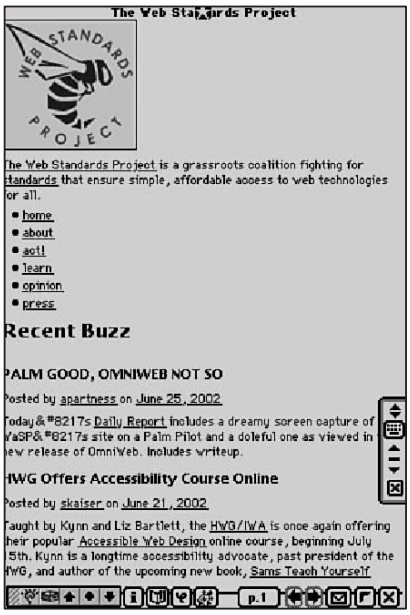
2.14
(left) The same site on a different day, as seen in a Palm Pilot. Look, Ma, no WAP! Screenshot courtesy of Porter Glendinning (**www.g9g.org**).

2.15
(right) The same site as seen in Microsoft's PocketPC. Screenshot courtesy of Anil Dash (**www.dashes.com/anil**).





2.16
Webstandards.org again, this time as viewed in Apple's long-discontinued Newton handheld. Screenshot courtesy of Grant Hutchinson (**www.splorp.com**).



That should be music to the ears of any designer or site owner who wants to reach the greatest number of visitors with the least effort. Strict compliance with (X)HTML and intelligent use of CSS frees designers and developers from the need to create multiple versions.

By the time you see this book, The Web Standards Project will have redesigned again, and these screenshots will be obsolete—although the points they prove will still be true.

A List Apart: One Page, Many Views

Published by our agency Happy Cog since 1998, *A List Apart* (`www.alistapart.com`) "for people who make websites" has long taught and promoted standards-based design. In February 2001, as the last of the standards-compliant browsers came to market, my cohorts and I converted the magazine to pure CSS layout and encouraged other designers to do likewise on sites they were designing. Hundreds of thousands have since done so.

A List Apart's form and structure demonstrate advantages of the powerful combination of semantic markup and CSS layout. The combination lets us support old browsers and other non-CSS-compliant devices without the need to create separate versions: devices that understand CSS see the layout [**2.17**]; those that don't understand CSS see the content, formatted by their browser in accordance with document structure [**2.18**].

Indeed, some readers prefer it this way; immediately following *A List Apart*'s initial CSS redesign, Netscape 4 usage by *ALA* visitors temporarily increased. It seems these visitors preferred a plain page their browser could handle to the previous layout whose gymnastics only underscored Netscape 4's weaknesses. Those who hold that standards hurt users of old browsers might consider this story an indicator of just the opposite: when properly used, standards help everyone.

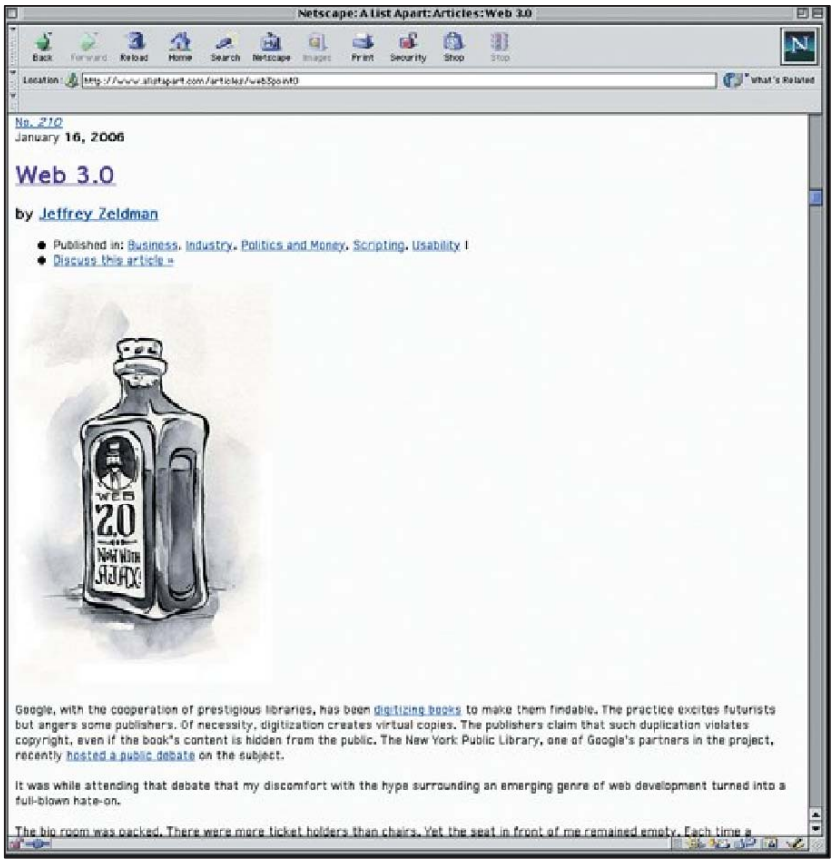
Further, in most cases this combination of semantic markup and CSS layout frees you from the need to create separate, "printer-friendly" page versions [**2.19**].

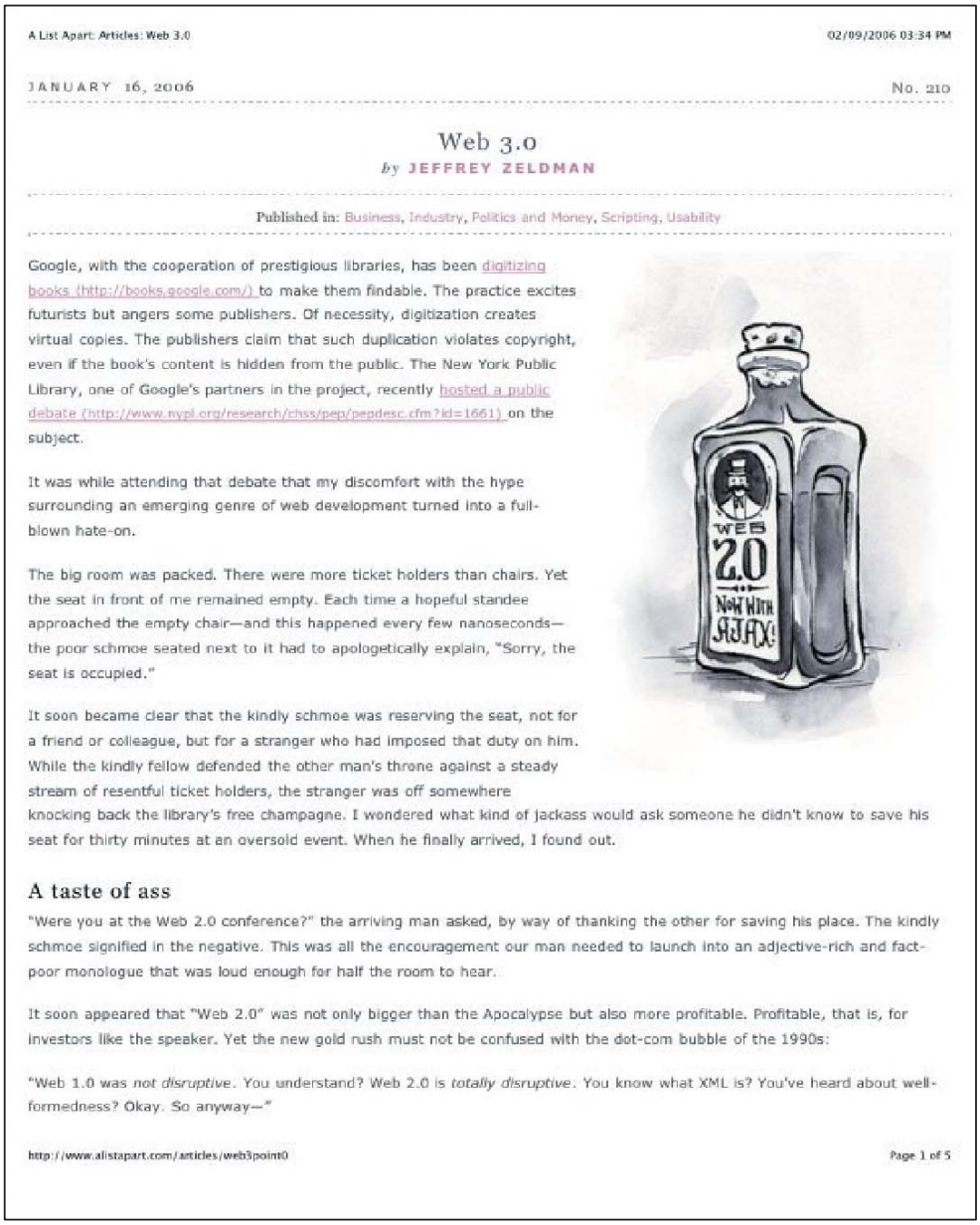
2.17

A List Apart (**www.alistapart.com**), "for people who make websites." Jason Santa Maria's beautiful layout shines when the site is viewed in a standards-compliant browser.

2.18

The same site in a 4.0 browser. No CSS layout, no problem. The site is still readable and usable; markup, filtered through browser defaults, provides the basic layout.





A List Apart: Articles: Web 3.0 02/09/2006 03:34 PM

JANUARY 16, 2006 No. 210

Web 3.0

by JEFFREY ZELDMAN

Published in: Business, Industry, Politics and Money, Scripting, Usability

Google, with the cooperation of prestigious libraries, has been digitizing books (http://books.google.com/) to make them findable. The practice excites futurists but angers some publishers. Of necessity, digitization creates virtual copies. The publishers claim that such duplication violates copyright, even if the book's content is hidden from the public. The New York Public Library, one of Google's partners in the project, recently hosted a public debate (http://www.nypl.org/research/chss/pep/pepdesc.cfm?id=1661) on the subject.

It was while attending that debate that my discomfort with the hype surrounding an emerging genre of web development turned into a full-blown hate-on.

The big room was packed. There were more ticket holders than chairs. Yet the seat in front of me remained empty. Each time a hopeful standee approached the empty chair—and this happened every few nanoseconds—the poor schmoe seated next to it had to apologetically explain, "Sorry, the seat is occupied."

It soon became clear that the kindly schmoe was reserving the seat, not for a friend or colleague, but for a stranger who had imposed that duty on him. While the kindly fellow defended the other man's throne against a steady stream of resentful ticket holders, the stranger was off somewhere knocking back the library's free champagne. I wondered what kind of jackass would ask someone he didn't know to save his seat for thirty minutes at an oversold event. When he finally arrived, I found out.

A taste of ass

"Were you at the Web 2.0 conference?" the arriving man asked, by way of thanking the other for saving his place. The kindly schmoe signified in the negative. This was all the encouragement our man needed to launch into an adjective-rich and fact-poor monologue that was loud enough for half the room to hear.

It soon appeared that "Web 2.0" was not only bigger than the Apocalypse but also more profitable. Profitable, that is, for investors like the speaker. Yet the new gold rush must not be confused with the dot-com bubble of the 1990s:

"Web 1.0 was *not disruptive*. You understand? Web 2.0 is *totally disruptive*. You know what XML is? You've heard about well-formedness? Okay. So anyway—"

http://www.alistapart.com/articles/web3point0 Page 1 of 5

2.19
From web to print: *ALA* articles become printer friendly on the fly, thanks to print style sheets.

Design Beyond the Screen

Figure 2.19 shows what an *A List Apart* article looks like when printed. As you can see, the sidebar has been removed, as readers of the printed piece don't need the navigational links the sidebar contains. The site's strong issue number bullet has been replaced by a simple link. Fonts have been optimized for printing, and the URL of every link is usefully displayed, whether said URL appears on the screen version or not. This magic is accomplished via a print style sheet designed by Eric Meyer, author of *Eric Meyer on CSS* (New Riders, 2002)

and a half dozen other CSS masterworks. In an article (`www.alistapart.com/articles/goingtoprint`), Eric explains the rationale and techniques for creating a print style sheet.

The important concept to grasp for now is that with a single, lightweight document—a print style sheet—*A List Apart* no longer needs to produce separate, "printer-friendly" versions. In all probability, neither will your sites. A possible exception: sites that use multipage article formats, like `www.nytimes.com` or the O'Reilly Network (`www.oreilly.com`), will still require a printer-friendly page simply to stitch the whole story together into a single document. But they, too, can still benefit from using a print style sheet, because the output of that stitched-together page can use a print style sheet.

Let's review the benefits reaped by the two sites we've just looked at.

Time and Cost Savings, Increased Reach

If designing with standards means you no longer need to create multiple versions of every site, it's easy to see that time and cost savings can be enormous:

- No more browser-specific versions
- No more "basic" versions for old browsers
- In many cases, no more separate mobile-specific versions
- In many cases, no more printer-friendly versions
- No more browser and platform sniffing, and no more straining of the server to fetch various browser or device-optimized components

Much as they might want to accommodate users of mobile devices, many organizations simply cannot afford to build separate mobile versions. Thanks to the XHTML and CSS standards, they don't have to. Without lifting a finger, these organizations will still reach new readers and customers, whose numbers are legion.

Strict standards compliance also provides a huge head start on solving the accessibility problem. If your site works in an old Palm Pilot, it most likely works in a screen reader like JAWS, although, of course, you need to test to be sure, and you might need to do a bit more work to be truly accessible. We'll look deeper into accessibility in Chapter 14 and in other sections of Part II.

Where We Go from Here

We can't get to tomorrow's web by following yesterday's design and development norms. Our way ahead is one of forward compatibility, of hewing closely to the spirit of web standards. What does this entail? Let's take a look.

Forward Compatibility Ingredients

- Full separation of structure from presentation and behavior
- Valid CSS used for layout. Tables used only for their true and original purpose: the presentation of tabular data such as that found in spreadsheets, address books, stock quotes, event listings, and so on
- Valid XHTML 1.0 Strict or Transitional (or HTML 4.01 or 5) used for markup
- Emphasis on structure. No presentational hacks in markup (Strict) or as few as possible presentational hacks in markup (Transitional)
- Structural labeling/abstraction of design elements—"Menu" rather than "Green Box"
- DOM-based scripting for behavior: if you need to fork your code, then sniff for object support, not browser versions
- Accessibility attributes and testing

Why You Should Care

Strict forward compatibility—CSS layout plus valid, semantic markup—is recommended for today's web, and the wide array of standards-compliant browsers that populate it. By adopting the techniques outlined above, you'll allow noncompliant browsers to access your content, but perhaps not every element of your site's branding and behavior. The US Navy site designed by Campbell-Ewald [**2.20**] combines pure CSS layout and XHTML 1.0 Strict markup with bits of Flash content to keep things interesting.

Benefits

- Forward compatibility: increased interoperability in existing and future browsers and devices (including mobile devices).
- Reaches more users with less work.
- No versioning.

2.20

The U.S. Navy (**www.navy.com**) is strict. XHTML 1.0 Strict, that is. Its site, designed by Campbell-Ewald, uses CSS, not tables, for layout (and mixes in a little Flash).

2.21

And just because we're apparently on a patriotic kick, the 2009 incarnation of the White House website (**www.whitehouse.gov**) is code we can believe in. XHTML 1.0 Transitional structures the content, CSS drives the presentation, and some DOM-based JavaScript adds a touch of behavior.

- Fewer accessibility problems. The content of sites so designed is generally accessible to all.
- Restores elegance, simplicity, and logic to markup.
- Restores document structure to documents.
- Faster, easier, less expensive production and maintenance—because sites cost less to produce and maintain, low budgets can avoid strain, while higher budgets (if available) can be put into writing, design, programming, art, photography, editing, and usability testing.
- Easier to incorporate into dynamic publishing and template-driven content-management systems.
- CSS layout makes possible some designs that cannot be achieved with HTML tables.
- Sites will continue to work in future browsers and devices.

Things to Consider

- Sites are likely to look quite plain in old browsers.
- Even in 2009, browser support for CSS is imperfect: some workarounds ("CSS hacks") may be required, mostly for older versions of Internet Explorer.
- DOM-based behaviors will not work in 4.0 and earlier mainstream browsers or in screen readers, text browsers, and most wireless devices; but if you're building with an eye toward progressive enhancement, then users without JavaScript will still be able to benefit from your content (more on this later).

Part II explains how standards work (individually and collectively) and offers tips and strategies to solve design and business problems related to various types of web development. But before we delve in, let's pause to consider some questions that might already have occurred to you.

If standards increase interoperability, enhance accessibility, streamline production and maintenance, reduce wasted bandwidth, and lower costs, why aren't all designers and developers using web standards correctly and consistently on every site they create?

Why aren't all clients clamoring for standards compliance the way inmates in old prison movies rattle tin cans against the bars of their cells when they want better chow? Why was it necessary for me to write this book or for you to read it, let alone beg your clients, colleagues, bosses, and vendors to read it? Why, six years later, was it necessary to write a third edition, and why were there still plenty of non-standards-compliant sites being created? Why aren't web standards more widely understood and used?

One reason is lack of knowledge. You can counter this ignorance and become a superb ambassador for web standards simply by building lean, accessible, fast-loading sites whose content can be easily found by search engines. Showing by doing—advocating by delivering results—will win you respect from teammates, managers, and clients. Creating sites that work harder for more users, more easily accomplish business goals, and are simpler and less expensive to maintain and update will not only help you evangelize web standards, it may also lead to a promotion and new clients.

The other reason many sites on the web still don't seem to have grokked standards is that some site owners, managers, and IT directors object to the use of web standards, typically on dubious grounds that don't stand up to the light of investigation and reason. If your boss or client tells you that you can't use web standards for one ridiculous reason or another, it's your job to persuade them otherwise. Fortunately, the next chapter tells you how.