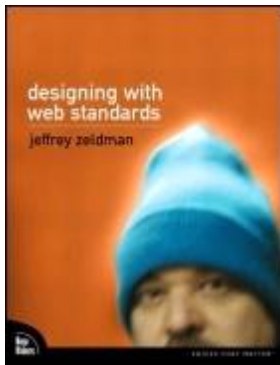


[\[Team LiB \]](#)

NEXT ►



[Table of Contents](#)

[Index](#)

Designing With Web Standards

By [Jeffrey Zeldman](#)

Start Reading ►

Publisher: New Riders Publishing

Pub Date: June 05, 2003

ISBN: 0-7357-1201-8

Pages: 350

You code. And code. And code. You build only to rebuild. You focus on making your site compatible with almost every browser or wireless device ever put out there. Then along comes a new device or a new browser, and you start all over again.

You can get off the merry-go-round.

It's time to stop living in the past and get away from the days of spaghetti code, insanely nested table layouts, tags, and other redundancies that double and triple the bandwidth of even the simplest sites. Instead, it's time for forward compatibility.

Isn't it high time you started designing with web standards?

Standards aren't about leaving users behind or adhering to inflexible rules. Standards are about building sophisticated, beautiful sites that will work as well tomorrow as they do today. You can't afford to design tomorrow's sites with yesterday's piecemeal methods.

Jeffrey teaches you to:

- - Slash design, development, and quality assurance costs (or do great work in spite of constrained budgets)
- - Deliver superb design and sophisticated functionality without worrying about browser incompatibilities
- - Set up your site to work as well five years from now as it does today
-

- Redesign in hours instead of days or weeks

- Welcome new visitors and make your content more visible to search engines

- Stay on the right side of accessibility laws and guidelines

- Support wireless and PDA users without the hassle and expense of multiple versions

- Improve user experience with faster load times and fewer compatibility headaches

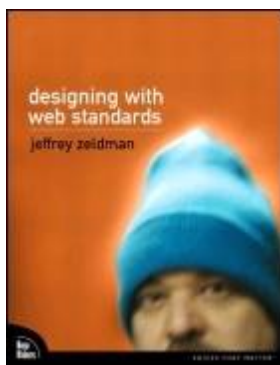
- Separate presentation from structure and behavior, facilitating advanced publishing workflows

[\[Team LiB \]](#)

NEXT ►

[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶



[Table of Contents](#)

[Index](#)

Designing With Web Standards

By [Jeffrey Zeldman](#)

Start Reading ▶

Publisher: New Riders Publishing

Pub Date: June 05, 2003

ISBN: 0-7357-1201-8

Pages: 350

[Copyright](#)

[About the Author](#)

[About the Technical Reviewers](#)

[Thanks and Praise](#)

[Tell Us What You Think](#)

[Introduction](#)

[One Size Does Not Fit All](#)

[A Continuum, Not a Set of Inflexible Rules](#)

[The Smell of Change](#)

[Part I. Houston, We Have a Problem](#)

[Before You Begin](#)

[Spiraling Costs, Diminishing Returns](#)

[Ending the Cycle of Obsolescence](#)

[What Is Forward Compatibility?](#)

[No Rules, No Dogma](#)

[Practice, Not Theory](#)

[Is This Trip Really Necessary?](#)

[Chapter One. 99.9% of Websites Are Obsolete](#)

[Modern Browsers and Web Standards](#)

[The "Version" Problem](#)

[Backward Thinking](#)

[When Good Things Happen to Bad Markup](#)

[The Cure](#)

[Chapter Two. Designing and Building with Standards](#)

[Jumping Through Hoops](#)

[The Cost of Design Before Standards](#)

[Modern Site, Ancient Ways](#)

[The Trinity of Web Standards](#)

[Into Action](#)

[Benefits of Transitional Methods](#)

[The Web Standards Project: Portability in Action](#)

[A List Apart: One Page, Many Views](#)

[Where We Go from Here](#)

[Chapter Three. The Trouble with Standards](#)

[Lovely to Look At, Repulsive to Code](#)

[2000: The Year That Browsers Came of Age](#)

[Too Little, Too Late?](#)

[Bad Browsers Lead to Bad Practices](#)

[Confusing Sites, Bewildering Labels](#)

[The F Word](#)

[Compliance Is a Dirty Word](#)

[Chapter Four. XML Conquers the World \(And Other Web Standards Success Stories\)](#)

[The Universal Language \(XML\)](#)

[XML Applications and Your Site](#)

[Compatible by Nature](#)

[A New Era of Cooperation](#)

[Web Standards and Authoring Tools](#)

[The Emergence of CSS Layout](#)

[The Mainstreaming of Web Standards](#)

[Executive Summary](#)

[Part II. Designing and Building](#)

[Chapter Five. Modern Markup](#)

[The Secret Shame of Rotten Markup](#)

[A Reformulation of Say What?](#)

[Executive Summary](#)

[Which XHTML Is Right for You?](#)

[Chapter Six. XHTML: Restructuring the Web](#)

[Converting to XHTML: Simple Rules, Easy Guidelines](#)

[Character Encoding: The Dull, the Duller, and the Truly Dull](#)

[Structural Healing—It's Good for Me](#)

[Visual Elements and Structure](#)

[Chapter Seven. Tighter, Firmer Pages Guaranteed: Structure and Meta-Structure in Strict and Hybrid Markup](#)

[Must Every Element Be Structural?](#)

[Hybrid Layouts and Compact Markup: Dos and Don'ts](#)

[Outdated Methods on Parade](#)

[Chapter Eight. XHTML by Example: A Hybrid Layout \(Part I\)](#)

[Benefits of Transitional Methods Used in These Chapters](#)

[Basic Approach \(Overview\)](#)

[First Pass Markup: Same as Last Pass Markup](#)

[Chapter Nine. CSS Basics](#)

[CSS Overview](#)

[Anatomy of Styles](#)

[External, Embedded, and Inline Styles](#)

[The "Best-Case Scenario" Design Method](#)

[Chapter Ten. CSS in Action: A Hybrid Layout \(Part II\)](#)

[Preparing Images](#)

[Establishing Basic Parameters](#)

[Navigation Elements: First Pass](#)

[Navigation Bar CSS: First Try at Second Pass](#)

[Navigation Bar CSS: Final Pass](#)

[Final Steps: External Styles and the "You Are Here" Effect](#)

[Chapter Eleven. Working with Browsers Part I: DOCTYPE Switching and Standards Mode](#)

[The Saga of DOCTYPE Switching](#)

[Controlling Browser Performance: The DOCTYPE Switch](#)

[Celebrate Browser Diversity! \(Or at Least Learn to Live with It\)](#)

[Chapter Twelve. Working with Browsers Part II: Box Models, Bugs, and Workarounds](#)

[The Box Model and Its Discontents](#)

[The Whitespace Bug in IE/Windows](#)

[The "Float" Bug in IE6/Windows](#)

[Flash and QuickTime: Objects of Desire?](#)

[A Workaday, Workaround World](#)

[Chapter Thirteen. Working with Browsers Part III: Typography](#)

[Size Matters](#)

[User Control](#)

[Old-School Horrors](#)

[A Standard Size at Last—But for How Long?](#)

[The Heartbreak of Ems](#)

[Pixels Prove Pixels Work](#)

[The Font Size Keyword Method](#)

[Chapter Fourteen. Accessibility Basics](#)

[Access by the Books](#)

[Widespread Confusion](#)

[The Law and the Layout](#)

[Accessibility Myths Debunked](#)

[Accessibility Tips, Element by Element](#)

[Tools of the Trade](#)

[Working with Bobby](#)

[Planning for Access: How You Benefit](#)

[Chapter Fifteen. Working with DOM-Based Scripts](#)

[Meet the DOM](#)

[Please, DOM, Don't Hurt 'Em](#)

[Showing and Hiding](#)

[Dynamic Menus \(Drop-Down and Expandable\)](#)

[Style Switchers: Aiding Access, Offering Choice](#)

[Chapter Sixteen. A CSS Redesign](#)

[Defining Goals](#)

[Establishing Basic Parameters](#)

[Rules-Based Design](#)

[A Home Button with CSS Rollover Effects](#)

[A CSS/XHTML Navigation Bar](#)

[Finishing Up](#)

[Part III. Back End](#)

[Backend A. Modern Browsers: The Good, the Bad, and the Ugly](#)

[Compliant Browsers: The First Wave](#)

[Index](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Copyright

Copyright 2003 by New Riders Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means—electronic, mechanical, photocopying, recording, or otherwise—without written permission from the publisher, except for the inclusion of brief quotations in a review.

Library of Congress Catalog Card Number: 2001094556

Printed in the United States of America

First edition: May 2003

07 06 05 04 03 7 6 5 4 3 2 1

Interpretation of the printing code: The rightmost double-digit number is the year of the book's printing; the rightmost single-digit number is the number of the book's printing. For example, the printing code 03-1 shows that the first printing of the book occurred in 2003.

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. New Riders Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty of fitness is implied. The information is provided on an as-is basis. The authors and New Riders Publishing shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Credits

Associate Publisher

Stephanie Wall

Production Manager

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

About the Author



Jeffrey Zeldman is among the best-known web designers in the world. His personal site (www.zeldman.com) has welcomed more than sixteen million visitors and is read daily by thousands in the web design and development industry.

He is the publisher/creative director of A List Apart (www.alistapart.com), an online magazine "For People Who Make Websites," and the founder of Happy Cog (www.happycog.com), a design and consulting agency whose clients include Clear Channel Entertainment, Warner Bros. Pictures, Fox Searchlight Pictures, and The New York Public Library. In 1998, he co-founded The Web Standards Project (www.webstandards.org), a grassroots coalition of web designers and developers that helped end the Browser Wars by persuading Microsoft and Netscape to support the same technologies in their browsers.

Jeffrey is the author of *Taking Your Talent to the Web* (New Riders: 2001) and of numerous articles for A List Apart, Adobe, Creativity, Digital Web, Macworld, PDN-Pix, and other sites and print publications. He has been a juror for the Communication Arts Interactive Festival, the Art Directors Club of New York, the 5K, the Addy Awards, and the Radio Mercury Awards, and is an Advisory Board member of the Meet the Makers and i3Forum conferences.

He has lectured to groups including the American Institute of Graphic Arts (AIGA), The Columbia University Libraries, Los Alamos National Laboratories, The New York Public Library, The Public Library Association, and The New York State Forum for Information Resource Management, and at conferences including Builder, CMP, Seybold, SXSW Interactive, Web Design World, and Webvisions, among others. But what he really wants to do is direct.

About the Technical Reviewers

These reviewers contributed their considerable hands-on expertise to the entire development process for Designing with Web Standards. As the book was being written, these dedicated professionals reviewed all the material for technical content, organization, and flow. Their feedback was critical to ensuring that Designing with Web Standards fits our readers' need for the highest-quality technical information.



Eric A. Meyer has been working with the web since 1993. He is currently employed as a Standards Evangelist with Netscape Communications while living in Cleveland, Ohio—which is a much nicer city than you've been led to believe. A recognized name in the industry, Eric is often asked to speak at conferences on the subjects of web standards, cross-browser compatibility, CSS, and web design. A graduate of and former Webmaster for Case Western Reserve University, Eric coordinated the authoring and creation of the W3C's CSS1 Test Suite and has recently been pushing the limits of CSS-based design as far as he can. Eric is also the author of *Eric Meyer on CSS: Mastering the Language of Web Design* (New Riders), *Cascading Style Sheets: The Definitive Guide* (O'Reilly & Associates), *CSS2.0 Programmer's Reference* (Osborne' McGraw-Hill), and the well-known CSS Browser Compatibility Charts.



J. David Eisenberg lives in San Jose, California with his kittens Marco and Zoë. He teaches HTML, XML, Perl, and JavaScript, and enjoys writing educational software and online tutorials.

David attended the University of Illinois, where he worked for the PLATO computer-assisted instruction project. He has also worked at Burroughs and Apple.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Thanks and Praise

I wrote this book, but many hands guided mine.

I am indebted to Jennifer Eberhardt at New Riders. Without her patience and persistence, this book would have stalled many times. Michael Nolan brought me to New Riders in 2000, for which I am grateful, and played a supporting role in the creation of this book. Chris Nelson encouraged me, and David Dwyer cleverly suggested I write this book when I had proposed instead a treatise to be called The Little Orange Book of Web Design.

In revising the text, I relied on the insights of two gifted technical editors. J. David Eisenberg is a teacher, a member of The Web Standards Project, and the author of SVG Essentials (<http://www.oreilly.com/catalog/svguess/>). Eric Meyer is a force behind Netscape's standards evangelism and the author of Eric Meyer on CSS (<http://www.ericmeyeroncss.com/>), whose title still embarrasses him. Both men are also my friends. You should be so lucky.

My thanks to the partners and colleagues with whom I was able to try out many of the techniques described in this book: Brian Alvey, Leigh Baker-Foley, Hillman Curtis, Nick Finck, Dennis James, Jamal Kales, Erin Kissane, Bruce Livingstone, Tanya Rabourn, Brad Ralph, Ian Russell, and Waferbaby. My gratitude also to Happy Cog's clients during the year I wrote this book, especially Steve Broback, Don Buckley, Eric Etheridge, Andrew Lin, Alec Pollak, and Randy Walker. Thanks for seeking the kind of work we do, indulging the occasional failed experiments, and paying those invoices.

Anything I've gotten right is largely due to Tantek Çelik, Joe Clark, Todd Fahrner, and (again) Eric Meyer. They know more than I ever will but think no less of me.

Without George Olsen and Glenn Davis there would not have been a Web Standards Project, and without the Project by now we would be coding every site fifteen ways. Thanks, gents, for helping to change the world.

I am grateful to every member of the Project's steering committee but wish to especially thank Tim Bray for the wit, Steven Champeon for keeping it on a higher plane, and Dori Smith for keeping it real. Thanks, too, to Rachel and Andrew for working so well with Macromedia and with its user community.

Jeffrey Veen may not realize it, but he helped me learn to relax as a public speaker and that enabled me to more effectively spread the message of web standards.

I learned from every supporter of The Web Standards Project and gained even more from its detractors, for it was their concerns that demanded an answer and their objections that helped shape the group's strategy between 1998 and 2002.

Thanks to the browser engineers who have worked so hard over the past three years, overcoming great odds and small budgets to truly deliver on the promise of web standards.

Thanks also to Janet Daly, Karl Dubost, Håkon Lie, Molly Holzschlag, Meryl Evans, and Michael Schmidt. And to Douglas Bowman, Owen Briggs, Chris Casciano, Eric Costello, Todd Dominey, Craig Salla, Christopher Schmitt, Mark Newhouse, and Waferbaby for pioneering CSS layouts. And to hundreds of web designers whose work invigorates and inspires us all. This book would double in size if I tried to list and thank you all, but you know who you are.

Special thanks to those designers whose work orbits a different planet from the one described in this book—people like Warren Corbitt, Joshua Davis, Matt Owens, and Lee Misenheimer. Your work challenges me to think twice about everything I do and believe as a web designer. There is room for both our visions.

I want to thank my father Maurice for co-authoring me and to wish him and his bride Katherine continued health,

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶

Tell Us What You Think

As the reader of this book, you are the most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As the Senior Development Editor for New Riders Publishing, I welcome your comments. You can fax, email, or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger. When you write, please be sure to include this book's title, ISBN, and author, as well as your name and phone or fax number. I will carefully review your comments and share them with the author and editors who worked on the book.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of email I receive, I might not be able to reply to every message.

Fax: 317-581-4663

Email: jennifer.eberhardt@newriders.com

Mail: Jennifer Eberhardt
Senior Development Editor
New Riders Publishing
201 West 103rd Street
Indianapolis, IN 46290 USA

[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Introduction

There was a time not that long ago when many drivers thought nothing of tossing empty bottles out the windows of their cars. Years later, these same citizens came to realize that littering was not an acceptable way to dispose of their trash. The web design community is now undergoing a similar shift in attitude, and web standards are key to this transformation.

The history of our medium has been to solve today's problems at tomorrow's expense. This book will show that the build-now, pay-later approach is no longer productive or necessary, and that today's problems can be solved without generating worse dilemmas downstream. It will also lay to rest the notion that designing for standards means leaving some users behind. In fact, it most often means just the opposite.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

One Size Does Not Fit All

In this book, we'll examine some of the ways that standards can solve common problems of design and production. No book can cover every problem and solution, and another author's book might take an entirely different approach to any issue discussed in these pages. This book is biased toward meeting practical and immediate needs in a way that anticipates future requirements. The techniques and ideas advanced in this book have been tested and found useful in my agency's design and consulting practice, and these ideas, or variations on them, have been used on thousands of forward-looking sites.

Not every reader will immediately use every idea discussed in this book. If your style emphasizes tight grids, you might not cotton to rules-based design as described in [Chapter 16](#), "A CSS Redesign." If your site must look great in 4.0 browsers, you might be interested in the hybrid techniques described in [Chapters 8](#), "XHTML by Example: A Hybrid Layout (Part I)," through [10](#), "CSS in Action: A Hybrid Layout (Part II)," and indifferent to the pure CSS layout techniques discussed in [Chapter 16](#).

Any thinking designer, developer, or site owner will endorse the general notions advanced in this book. Standards are vital to any medium. Because the software through which the web is viewed finally supports standards, it makes sense to learn about and correctly use them. Doing so saves time and money, reduces overhead, extends the usable life of our sites, and provides greater access to our content.

The latter point is important to anyone who wants to reach a wider, rather than a narrower, audience—particularly as nontraditional Internet access increases. It also has legal implications as more nations and more U.S. states create and begin to enforce accessibility regulations. Web standards and accessibility can help your site stay on the right side of these laws.

Theory Versus Practice

But some specific ideas and techniques advanced in this book are open to debate. If you are a hardcore standards geek (and I mean that in the nicest way possible), you might be unwilling to use XHTML until all browsers properly support sending XHTML documents as application/xhtml+xml instead of text/html. For details, see Ian Hickson's "Sending XHTML as Text/HTML" (<http://www.hixie.ch/advocacy/xhtml>).

If you agree with Ian's view, you might choose to use HTML 4.01 for now, or you might want to configure your web server to send application/xhtml+xml to browsers that understand it and text/html to those that do not (<http://lists.w3.org/Archives/Public/www-archive/2002Dec/0005.html>). In this book, I have avoided such issues because of my bias toward getting work done under present conditions—a bias I believe most of this book's readers will share.

Hybrid Layouts: On Their Way Out?

Likewise, some hardcore CSS fans might despise the idea of hybrid CSS/table layouts. Hybrid layout methods ([Chapters 8](#) through [10](#)) are offered for those who need them. Specifically, they are offered for designers whose work must look almost as good (and almost the same) in old, noncompliant browsers as it does in newer, more compliant ones.

We might not need these methods for long. As I write this page, [ESPN.com](#) has relaunched using CSS layout [[L1](#)]. Ten million readers visit this sports site each day. When a site that big and that commercial adopts CSS layout

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

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 ([Chapter 2](#), "[Designing & Building with Standards](#)") in your first site or even your fifth. Your early efforts at accessibility ([Chapter 14](#), "Accessibility Basics") might deliver only the minimum required by WAI Priority 1, and you might not get all of it exactly right.

The point is to begin. Fear of imperfection can immobilize the unwary in 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 nonstructural markup with meaningful `<h>` 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 and XHTML.

Show, Don't Sell

Designers sometimes bog down on the selling part of standards. Over the years, I have received hundreds of letters from designers who want to use standards, "but my client won't let me." If standards are a continuum, how can any client oppose at least some effort in their direction? For instance, even the most table-driven site can validate against HTML 4.01 or XHTML 1.0 Transitional and can be made to conform to U.S. Section 508 or WAI Priority 1 accessibility guidelines. No client would object to an error-free, accessible site.

What most designers concerned with selling standards are really saying is that they can't go as far as they would like to go on a particular project. For instance, they can't use pure CSS layout for a particular project because the client (or their boss) uses Netscape 4, whose CSS support is patchy at best. That might be true, but it is no reason not to write valid markup and correct CSS and use the Two-Sheet method described in [Chapter 9](#) to deliver an acceptable and branded look and feel across multiple browser generations.

My agency is religious about web standards and accessibility but not about which methods we use or where they fall on an imaginary Standards Purity continuum. We use the method that best suits the project. To sell it, we do two things:

-

In our proposals, we explicitly state which technologies will be used, keeping the description simple and straightforward. For instance, "XHTML 1.0 Transitional, a current standard, will be used for markup." After the client has agreed to the proposal and signed the contract, "permission" to use the indicated standard has been granted, and further hand wringing is not needed. Where a choice will impact the visual result in older browsers, this is also explicitly stated in the proposal.

-

As work begins, in showing various stages to the client, we keep technological discussion to a minimum—even when dealing with a technologically savvy client. When delivering a redesign that is one-third the bandwidth of its predecessor and that retains formatting (even advanced formatting), no matter how many times it is changed or updated, we don't say, "CSS makes this possible." We say, "We've set up a system that protects formatting and is low bandwidth." If the client thinks we're smart and chooses to grant us more business, we can live with that.

Let Your Work Do the Selling for You

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Smell of Change

Such changes are taking place everywhere, sometimes quickly and other times slowly. These changes will materialize anywhere thinking people face the task of creating or updating web content. Almost unnoticed amid economic and political worries and the grind of daily deadlines, our shared understanding of how the web works and how it should be built is undergoing a profound and continual metamorphosis. Web standards will soon be as widely discussed and studied as web usability and information architecture, and they will be considered every bit as essential as those disciplines—because they are every bit as essential to the health of our sites and of the medium in which we toil.

This book is large and has been crafted with care, 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 than what this book or any single reference could convey. And as we've already mentioned, there are more ways to look at the issues we've covered than the way this author looks at them.

Put two designers in a room and you will hear three opinions. No two designers are likely to agree on every aspect of typography, branding, navigation, or color. The same is true of standards. There are as many disagreements in this realm as there are practitioners and theorists.

No book can deliver all things to all people, and this book is merely a pointer in the general direction of a journey whose path you must find for yourself. 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.

I stumbled onto web standards after three years of designing sites the old-fashioned way, and it took me another five years to reach the understanding on which this book is based. You might disagree with any part of what I've said in these pages. I might disagree with some parts myself six months or two years from now. 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 projects reach the most people for the longest time, and often at the lowest cost.

If not now, when? If not you, who?

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Part I: Houston, We Have a Problem

[Before You Begin](#)

[1 99.9% of Websites Are Obsolete](#)

[2 Designing & Building with Standards](#)

[3 The Trouble with Standards](#)

[4 XML Conquers the World \(And Other Web Standards Success Stories\)](#)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

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. It's a cost most of us can no longer afford (if we ever could).

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Spiraling Costs, Diminishing Returns

Spaghetti code, deeply nested table layouts, tags, and other redundancies double and triple the bandwidth required for our simplest sites. Our visitors pay the price by waiting endlessly for our pages to load. (Or they tire of waiting and flee our sites. Some wait patiently only to discover that when our site finally loads, it is inaccessible to them.)

Our servers pay the price by chugging out 60K per page view when 20 might suffice—and we pay our hosting companies (or increase our IT budgets) to keep up with the bandwidth our pages squander. The more visitors we attract, the higher the cost. To cope with our ad hoc front-end designs, our databases make more queries than they have to, further escalating our expense. Eventually we're forced to buy or lease additional servers to keep up with the demand—not of increased visitors but of excess markup and code.

Meanwhile, the hourly rates of the programmers we pay to code our sites six different ways (and to then write more code to serve the appropriate version to each visitor) can drive development costs so high we simply run out of money. A new browser or wireless device comes along, and with no cash left to cover the cost of coding for that browser or device, the cycle of obsolescence resumes.

Many of us have had the startling experience of visiting a site with a new browser, only to be told the site requires a "modern" browser that's much older than the one we're using. The site's owners and developers are not necessarily stupid or inconsiderate; they simply might have burned through their "perpetual upgrade" budget and have nothing left to give.

In other cases, the problem is not lack of funds but lack of knowledge or a misguided sense of what constitutes the best return on investment. Connected Earth, whose slogan is "How communication shapes the world," was recently redesigned at a reported cost of 1,000,000 (approximately \$1.6 million U.S. at the time of this writing). Despite the funds lavished on the site's development, it is incompatible with nearly every modern browser on earth. It denies access to users of Mozilla [1], Netscape 6/7, and Opera [2]. Because the site is also incompatible with non-Windows operating systems, users of Internet Explorer for Macintosh are equally out of luck.

1. Despite a huge development budget, Connected Earth is incompatible with nearly all modern browsers. It excludes users of Mozilla (shown here), Netscape 6/7, and Opera on any platform, and users of Internet Explorer in Mac OS (www.connected-earth.com).



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Ending the Cycle of Obsolescence

Technologies created by the World Wide Web Consortium (see the sidebar, "[What Is the W3C?](#)") and other standards bodies and supported by most current browsers and devices now make it possible to design and build sites that will continue to work, even as those standards and browsers evolve. Battle-scarred industry veterans might be skeptical of this claim, but this book will show how it works.

This book will teach you how to escape the "build, break, rebuild" cycle, designing for the present and future web on and beyond the desktop, without excluding potential visitors and without wasting increasingly scarce time and money on short-sighted proprietary "solutions" that contain the seeds of their own obsolescence.

This is not a book for theorists or purists, but for practical people who need to get work done. It's for creative and business professionals who seek sensible advice and a range of proven techniques so that they can modify their skills and thinking and get on with the job of creating great websites that work for more visitors and customers.

Armed with this book, designers and developers will be able to quickly modify their existing practices to create websites that work in multiple browsers and devices instead of a handful, while avoiding perpetual obsolescence born of proprietary markup and coding techniques.

Site owners and managers who read this book will be able to stop wasting money on specs that only perpetuate the wasteful cycle. They will know how to write requirement documents that lead to forward-compatible sites.

What Is the W3C?

Created in 1994, the World Wide Web Consortium (W3C) (<http://www.w3.org/>) hammers out 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 (<http://www.w3.org/People/Berners-Lee/>), invented the web in 1989. Specifications developed by the W3C include HTML, CSS, XML, XHTML, and the standard Document Object Model (DOM), among many others.

For years, the W3C referred to such 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 (www.webstandards.org) 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. (See the related sidebar, "[What Is The Web Standards Project?](#)")

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." See [Chapter 3](#), "The Trouble with Standards," for details.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

What Is Forward Compatibility?

What do we mean by forward compatibility? We mean that, designed and built the right way, any document that is 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. Open standards make this possible.

As an added attraction, designing and building with standards lowers production and maintenance costs while making sites more accessible to those who have special needs. (Translation: more customers for less cost, improved public relations, decreased likelihood of accessibility-related litigation.)

What do we mean by web standards? We mean structural languages like XHTML and XML, presentation languages like CSS, object models like the W3C DOM, and scripting languages like ECMAScript, all of which (and more) will be explained in this book.

Hammered out by committees of experts, these technologies have been carefully designed to deliver the greatest benefits to the largest number of web users. Taken together, these technologies form a roadmap for rational, accessible, sophisticated and cost-effective web development. (Site owners and managers: Don't worry yourselves over the technical chapters in this book. Just make sure your employees or vendors understand them.)

What do we mean by standards-compliant browsers? We mean products like Mozilla, Netscape 6+, MSIE5+/Mac, MSIE6+/Win, and Opera 7+ that understand and support XHTML, CSS, ECMAScript, and the DOM. 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.

Nevertheless, modern browsers are solid enough that we can discard outdated methods, work smarter, and satisfy more users. And because standards are inclusive by nature, we can even accommodate folks who use old browsers and devices—but in a forward-compatible way.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

No Rules, No Dogma

This is not a religious or 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. It will not oversell web standards by pretending that every W3C recommendation is flawless.

This book will tell you what you need to know to work around the occasional compliance hiccup in Internet Explorer, Netscape Navigator, 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.

This book will not lie to you. Some proprietary methods and shortcuts are easier to use than some W3C specs. For instance, proprietary, IE-only scripting using Microsoft shortcuts such as innerhtml might be faster and easier than scripting for all browsers via the W3C standard DOM. Even so, from a business point of view, it makes more sense to code for all browsers rather than for one, and the DOM is the way to do that.

Likewise, even though we'll explore the benefits of structural markup and XHTML, we won't pretend that every tag on every page of every site always needs to be structural. And we won't tell you that every site must immediately move from HTML to XHTML—although we hope the advantages of XHTML will compel you to consider making that transition as soon as you can.

This book's author is known for advocating that web pages be laid out with Cascading Style Sheets (CSS) instead of traditional HTML tables whenever possible (whenever audience appropriate). The CSS standard solves numerous problems for developers and readers alike. CSS layout is the future and is already being used on many sites, from heavily trafficked corporate entities like Wired (www.wired.com) [3] to major search engines (www.alltheweb.com) to public sector and personal sites.

3. It's big. It's corporate. And it's built with web standards XHTML and CSS (www.wired.com).



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Practice, Not Theory

We have nothing against the purists whose passion drives the creation of web standards. Quite the contrary: We admire such people immensely and are 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. Our exploration of web standards will be rooted in the context of design, content, and marketing issues. Our goal is to guide you through the world of web standards and to help you reach your own decisions—the only decisions with which any designer or client can ever be truly happy.

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.

The old-school techniques had their place when some standards had yet to be written, whereas others were poorly supported in mainstream browsers. But that day is gone. Likewise, 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, too, is gone.

XHTML, XML, CSS, ECMAScript, and the DOM are here to stay. They are not ends in themselves, but components of a rational solution to problems that have plagued site owners and builders since the <blink> tag.

What Is The Web Standards Project?

Founded in 1998, The Web Standards Project helped end the browser wars by persuading Netscape, Microsoft, and other manufacturers to accurately and completely support specifications ("standards") that reduce the cost and complexity of development and ensure simple, affordable access for all. In addition to browser makers, the group now works with development tool manufacturers like Macromedia and with site owners and developers. The Web Standards Project is a grassroots coalition. Its activities are entirely voluntary and not for profit.

Is This Trip Really Necessary?

For websites to shed their traditional woes and move to the next level, designers and developers must learn to use web standards, and site owners and managers must be told how standards can help their business.

The revelation of web standards will not manifest itself on its own. Business owners are unlikely to scour the W3C website, decipher its pocket-protector-style documents, and intuitively grasp what cryptic acronyms like XHTML or CSS might mean to their profitability. Likewise, overworked designers and developers, struggling to make deadlines, have little time to trek through mailing lists and online tutorials in search of comprehensible explanations of changing web technologies.

To make the case for standards, this book had to be written. As cofounder of The Web Standards Project, the job fell to me. My name is Jeffrey. I carry a mouse. I also write in the editorial "we" except when absolutely necessary. It's a conceit I adopted when I began publishing websites in 1995. I (we) trust the reader will not be disturbed by this habit.

You'd probably rather read about graphics and motion design, new thinking in site architecture, and usability than about the changing technological underpinnings of the web. I would rather write about those things. But our best efforts in design, architecture, and usability will be wasted if our sites stop working in Browser X or Device Y. There could be no filmmaking without industry-wide agreement on frame rates, lenses, and audio recording techniques. Likewise, the continued health of web design and development depends on the adoption of web standards. Without these standards, there can be no true usability and no coherent approach to design.

In terms of acceptance, the web got off to a faster start than any other medium ever introduced. But its commercial success preceded the development of industry standards, throwing all of us into the perilous position of creating products (websites) that were continually made obsolete by one proprietary browser or device innovation after another. We were all so busy producing that we had no time to question the way we worked. Today, if we intend to keep working and producing, we must question and modify our methods.

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.

—Jeffrey Zeldman
New York City
2003

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 multimillion-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 might not know it yet, 99.9% of all websites are obsolete.

These sites might look and work all right in mainstream, desktop browsers whose names end in the numbers 4 or 5. But outside these fault-tolerant environments, the symptoms of disease and decay have already started to appear.

In modern versions of Microsoft Internet Explorer, Opera Software's Opera browser, Netscape Navigator, and Mozilla (the Open Source, Gecko-based browser whose code drives Navigator, CompuServe, AOL for OS X, AOL China, and other browsing environments), carefully constructed layouts have begun falling 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 increasingly popular nontraditional devices from Palm Pilots to web-enabled cell phones, many of these sites have never worked and still don't, whereas others function marginally at best. Depending on 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 continually increases the cost and complexity of web development while never truly achieving its intended goal, we must examine modern browsers and see how they differ from the incompatible browsers of the past.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Modern Browsers and Web Standards

Throughout this book, when we refer to "modern" or "standards-compliant" browsers, we mean browsers that understand and support HTML and XHTML, Cascading Style Sheets (CSS), ECMAScript, 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.

As of this writing, such browsers include, among others, Mozilla 1.0 and higher; Netscape Navigator 6 and higher; Microsoft Internet Explorer 6 and higher for Windows; Microsoft Internet Explorer 5 and higher for Macintosh; and Opera Software's Opera 7 browser. For a chart that lists and compares the first wave of compliant browsers, please see, "[Modern Browsers: The Good, the Bad, and the Ugly](#)," in [Part III](#) of this book. Note that it is not an exhaustive list. Any attempt to list every standards-compliant browser would date this book faster than the Macarena. Although we will use the term "standards-compliant," please remember what we said before this chapter in this book's "[Before You Begin](#)" section: No browser is perfectly standards compliant or can be.

Lack of browser perfection is no reason to avoid standards. Millions of people currently use Internet Explorer 5 or 5.5 for Windows. From a standards point of view, those browsers are inferior to IE6/Windows, Netscape 6+, and so on. Does that mean if your audience includes IE5/Windows users you should forget about web standards? Does it mean you should tell IE5/Windows users to upgrade or get lost? We think 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 Netscape 4 users to go take a hike. A site properly designed and built with standards is unlikely to look pixel-for-pixel the same in Netscape 4 as it does in more compliant browsers. In fact, depending on your design method, it might look entirely different. And that's probably okay. We'll explain the why and how in [Part II](#) of this book, "[Designing and Building](#)."

New Code for a New Job

Modern browsers are not merely newer versions of the same old thing. They differ fundamentally from their predecessors. In many cases, they've been rebuilt from the ground up. Mozilla, Netscape 6/7, and related Gecko-based browsers are not new versions of Netscape Navigator 4. IE5+/Mac is not an updated version of IE4/Mac. Opera 7 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 nearly perfectly as possible with the web standards discussed in this book.

By 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 that did not pose much of a development headache. If browsers didn't support the Portable Network Graphic (PNG) standard, for example, then developers didn't use PNG images. No problem. The trouble was, these old browsers paid lip service to some standards by supporting them partially and incorrectly. Slipshod support for standards as basic as HTML created an untenable web-publishing environment that led in turn to unsustainable production methods.

When a patient's appendix bursts, a qualified surgeon performs a complete appendectomy. Now imagine if, instead, a trainee were to remove half the appendix, randomly stab a few other organs, and then forget to sew the patient back up. We apologize for the unsavory imagery, but it's what standards support was like in old browsers: dangerously incomplete, incompetent, and hazardous to the health of the web.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The "Version" Problem

The creation of multiple versions of nonstandard markup and nonstandard code, each tuned to the nonstandard quirks of one browser or another, is the source of the perpetual obsolescence that plagues most sites and their owners. The goal posts are always receding, and the rules of the game are forever changing.

Although it's costly, futile, and unsustainable, the practice persists even when it's unnecessary. 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 scripts even though IE6 can handle standard ECMAScript and the DOM. They then feel compelled to write separate detection scripts (and separate code) for modern Netscape browsers that can also handle standard ECMAScript and the DOM.

As the example suggests, much browser and device sniffing and much individual version creation is unnecessary in today's standards-friendly climate. In fact, it's worse than unnecessary. Even with constant updating—which not every site owner can afford—detection scripts often fail.

For instance, in Windows, the Opera browser identifies itself as Explorer. It does this mainly to avoid getting blocked by sites (such as many banking sites) that sniff for IE. But scripts written exclusively for IE browsers are likely to fail in the Opera browser. When Opera identifies itself as IE (its default setting upon installation) and when developers write IE-only scripts, site failure and user frustration loom large. Users have the option to change their User Agent (UA) string and force Opera to identify itself honestly instead of trying to pass for IE. But few users know about this option, and they shouldn't need to.

In addition to proprietary scripts, developers write presentational markup that doubles the bandwidth needed to view or serve a page while making that page less accessible to search engines and nontraditional browsers and devices. Such strategies often cause the very problem they were intended to solve: inconsistent rendering between one browser and another [[1.1](#)].

1.1. The MSN Game Zone (zone.msn.com/blog.asp) sports seven external style sheets and still doesn't render properly in most modern browsers. It also brags 14 scripts (most of them inline), including heavy-duty browser detection. And it still doesn't work. Throwing more versions of code at a problem rarely solves it.



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Backward Thinking

Peel the skin of any major 2003-era site, from Amazon to Microsoft.com, from Sony to ZDNet. Examine their tortuous nonstandard markup, their proprietary ActiveX and JavaScript (often including broken detection scripts), and their ill-conceived use of CSS—when they use CSS at all. It's a wonder such sites work in any browser.

These sites work in yesterday's mainstream browsers because the first four to five 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.

Often, nonstandards-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 practice taxes the dial-up 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.

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 ECMAScript and the DOM (see "[The 'Version' Problem](#)").

At the same time, these multiple versions squander 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 that 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 would save \$875. One that spends \$160,000 a year would save \$56,000.

Yahoo's front page [[1.3](#)] is served millions of times per day. Each byte that is wasted on outdated HTML design hacks is multiplied by an astronomical number of page views, resulting in gigabytes of traffic that tax Yahoo's servers and add Pentagon-like costs to its overhead. If Yahoo would simply replace its deprecated, bandwidth gobbling `` tags [[1.4](#)] with bandwidth-friendly CSS, the cost of serving each page would greatly diminish, and the company's profits would consequently rise. So why hasn't Yahoo made the switch?

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

When Good Things Happen to Bad Markup

Early in a computer programmer's education, he or she learns the phrase "Garbage In, Garbage Out." Languages like C and Java don't merely encourage proper coding practice; they demand it.

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. You can turn a high-quality EPS into a suitably optimized web page logo, but you can't convert a low-resolution GIF into a high-quality web, print, or TV logo. Garbage in, garbage out.

But traditional mainstream browsers don't work the same 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 of which they are largely unaware. At the same time, it has 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:

[\[View full width\]](#)

```
<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 `` 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, more structural 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 XML-based markup. 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 and the browsers that love them. Little wonder that skilled coders often view front-end development as brain-dead voodoo unworthy of respect or care.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Cure

After a long struggle pitting designers and developers against the makers of leading browsers, we can finally employ techniques that guarantee the appearance and behavior of our sites, not simply in one manufacturer's browser, but in all of them.

Hammered out by the members of the World Wide Web Consortium (W3C) and other standards bodies and supported in current browsers developed by Netscape, Microsoft, Opera, and other companies, technologies like CSS, XHTML, ECMAScript (the standard version of 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 devices, from wireless gadgets and web-enabled cell phones fancied by teens and executives to Braille readers 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.
- - Separate style from structure and behavior, delivering creative layouts backed by rigorous document structure and facilitating the repurposing of web documents in advanced publishing workflows.
- - Transition from HTML, the language of the web's past, to the more powerful XML-based markup of its 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

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

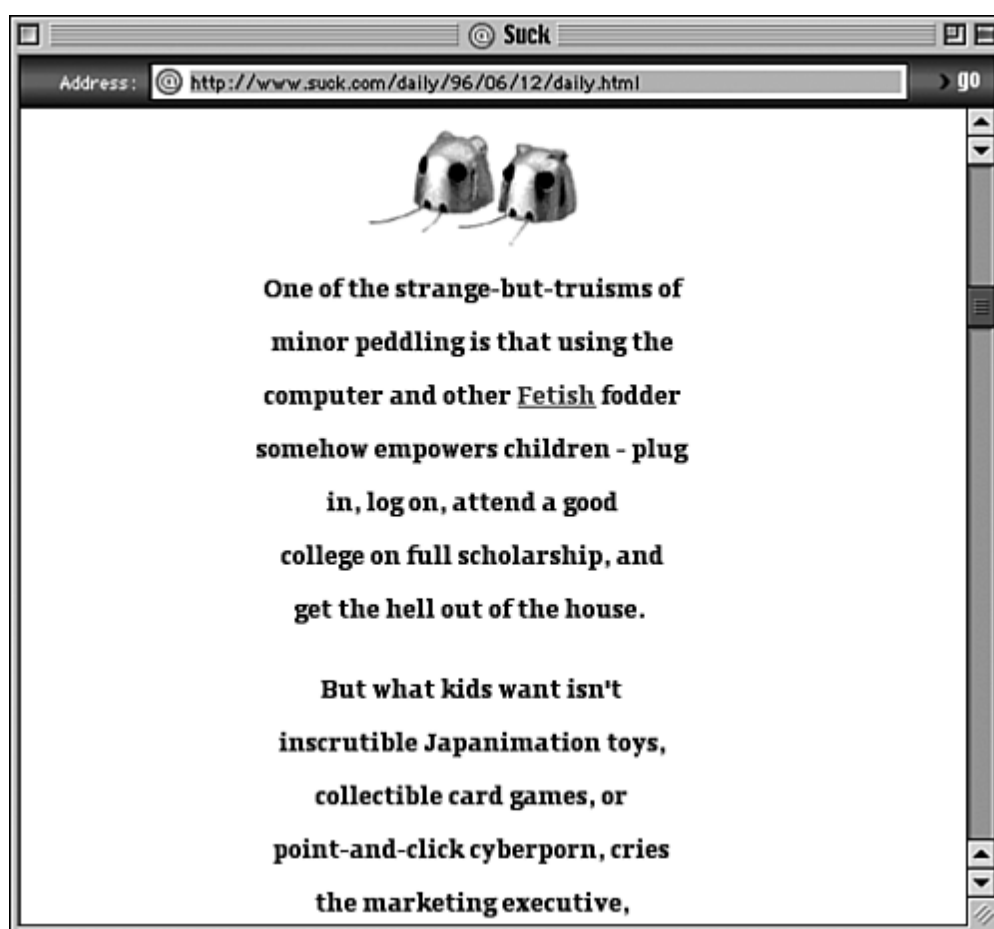
◀ PREVIOUS

NEXT ▶

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 [Suck.com](http://www.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, but in the mid-1990s, when Suck debuted, most sites buried their content behind splash pages, welcome pages, mission statements, and confusing "Table of Contents" pages.

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



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 exercises in metallic bevels and high-tech, Gothic glows, or non-designed messes flung together by systems administrators and self-taught HTML auteurs. Many sites 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 sites still revel in these techniques [2.2]. In a web where more was more, Suck stood out by daring to do less.

2.2. Moon's Design's Ulead Glow Frame tutorial (moondesigns.com) harkens back to typical mid-1990s web design. 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.



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

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 no longer available.)

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. 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 in their daily work, and many books still teach these outdated—and in today's web, counterproductive—methods. One otherwise excellent web design book of 2002 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 scriptable, but bad or nonsensical advice of this kind continues to appear in widely distributed web design tomes, thus perpetuating folly and ignorance.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

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 browsers 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 the design 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.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Modern Site, Ancient Ways

Leap from 1995 to 2001 and consider a contemporary website promoting The Gilmore Keyboard Festival (www.thegilmore.com) [2.3]. It's a pretty site, put together using labor-intensive table layout techniques that have long been industry norms.

2.3. The Gilmore Keyboard Festival website (www.thegilmore.com). Lovely to look at, but painful to update or maintain.



Aside from the fact that the Gilmore site makes assumptions about the size of the visitor's monitor and browser window, what's wrong with this picture? From the owner's point of view, the site hits a couple of sour notes:

-

Financial penalty of change— If anything about the festival changes—for instance, if an additional Recital Series is added to this year's lineup—the site can't be updated via a simple text link. Nor can the web designers easily add additional links to the text GIF image map that serves as the site's navigational menu. The HTML table that combines the various image slices into an apparent whole [2.4] would burst apart if the size of any component image were to be altered.

2.4. The same site, with CSS margins and borders added to reveal construction methods (and potential maintenance headaches).



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

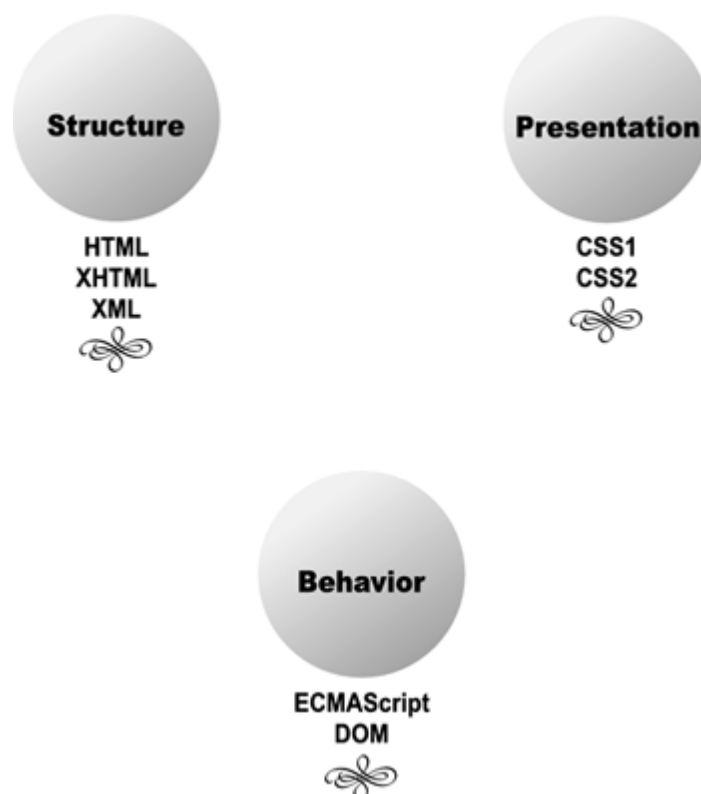
◀ PREVIOUS

NEXT ▶

The Trinity of Web Standards

[Figure 2.6](#) indicates how web standards solve the problems we've been discussing by breaking any web page into three separate components: structure, presentation, and behavior.

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



Structure

A markup language (XHTML: <http://www.w3.org/TR/xhtml1>) contains text data formatted according to its structural meaning: headline, secondary headline, paragraph, numbered list, definition list, and so on.

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 paragraph 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/
    xhtml1>XHTML</a>) contains text data formatted according
    to its structural 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. The subhead, "Structure," would be marked up as a
    definition title. The paragraph you're now reading
    would be wrapped in definition data tags.
  </dd>
</dl>
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Into Action

If Suck were alive and well today, web standards like XHTML and CSS would allow the staff to concentrate on writing. A basic XHTML template would deliver the document structure. CSS would control the look and feel without requiring additional design work per-issue, aside from the preparation of article-specific images. Paragraph tags would be properly used to denote the beginnings and ends of paragraphs, not to force vertical gaps between each line of text. (CSS would do that job.)

In graphical browsers like IE, Mozilla/Netscape, and Opera, style sheets would ensure that Suck looked as its designer intended. Structured XHTML would deliver Suck's content not only to these browsers but also to personal digital assistants (PDAs), screen readers, and text browsers without the accompanying nonstructural hiccups of fake paragraphs and similar hostages to markup-as-a-design-tool.

As a content-focused site, [Suck.com](#) would make 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. But Suck could also benefit from a transitional approach: simple XHTML tables for the positioning of primary content areas and CSS for the rest.

The creators of The Gilmore site could not benefit from templating—at least, not on the site's front page as it is currently designed. But they could deliver their existing layout with CSS, conserving bandwidth while enabling the design team to change one section of the page without reworking the entire layout.

The Gilmore could use the CSS background property to position its primary image as a single JPEG 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, including some that work in 4.0 browsers whose support for CSS is incomplete.

The creators of The Gilmore could also use valid XHTML and accessibility attributes including alt, title, and longdesc to ensure that the site's content would be accessible to all instead of meaningless to many [\[2.5\]](#). Less graphic-intensive inner site pages could be delivered via transitional (CSS plus tables) or stricter (pure CSS) methods.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Benefits of Transitional Methods

Transitional methods that comply with XHTML and CSS would be a vast improvement over what we have today and would solve the problems discussed so far. With transitional techniques—CSS for typography, color, margins, and so on and XHTML tables for basic layout—you increase usability, accessibility, interoperability, and long-term viability, albeit at the cost of more work and expense (and in most cases, more bandwidth) than a nontable, pure-CSS approach.

Happy Cog (www.happycog.com), my web agency's business site, is a transitional one [2.7, 2.8, 2.9]. It combines XHTML table layout techniques with CSS1 and CSS2 and simple DOM-based scripting. The site complies with the XHTML 1.0 Transitional and CSS standards, and it's compatible with the accessibility requirements of Section 508 and WAI Priority 1 Guidelines (more about those guidelines in [Part II](#)).

2.7. Happy Cog, this author's business site (www.happycog.com), is an exercise in transitional forward compatibility, combining streamlined XHTML table layouts with CSS and the DOM. When viewed in a modern browser, its proper presentation is fully revealed.



2.8. Opened in an old browser (Netscape 4) whose support for CSS is iffy at best, most of Happy Cog's presentation comes through intact. Several subtleties of the design are lost, but we don't mind and neither will most Netscape 4 users, who are accustomed to a certain lack of polish on most sites they visit.



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Web Standards Project: Portability in Action

The Web Standards Project (WaSP) [2.10] 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 (a.k.a. yelling, whining, and pleading), but eventually browser makers bought into The WaSP's view that interoperability via common standards was an absolute necessity if the web was to move forward.

2.10. The Web Standards Project's home page as seen in Chimera, a Gecko-based browser for Mac OS X (www.webstandards.org). Its CSS layout looks the same in all modern, standards-compliant browsers. But wait, there's more!



Lip Service

One of the ironies of the struggle for standards compliance in browsers was that Netscape and Microsoft are W3C members who have contributed significantly to the creation of web standards, yet had to be bullied into fully supporting the very technologies they helped to create. Go figure.

After browsers finally began meaningfully supporting standards (see [Chapter 3](#), "The Trouble with 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 looks nice in standards-compliant browsers [2.10]. It also looks acceptable in older, less-compliant browsers [2.11]. But the site transcends the PC-based browsing space without requiring additional or alternative markup, code, or device detection. (Look, Ma, no versions!)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

A List Apart: One Page, Many Views

A List Apart (www.alistapart.com), this author's online magazine "for people who make websites," converted to CSS-only layout in February 2001. The design [2.15] was adapted from the site's branded "HTML Minimalist" look and feel, which had previously been accomplished via HTML tables.

2.15. A List Apart, the author's online magazine for web designers, as seen in a CSS-compliant browser (www.alistapart.com). ALA switched to CSS-only layout in February, 2001. Hundreds of other sites soon followed suit.



A Source of Inspiration

All style sheets used on A List Apart (and at zeldman.com, for that matter) are open source, free for your use and adaptation when creating your own sites. To find a site's style sheets, select View Source in your browser, make a note of the CSS file locations referenced in the <head> of the document, and then cut and paste that file location into your browser's address bar. For instance, if the XHTML in the <head> of the document reads like this:

```
<style type="text/css" media="all">
@import "/styles/basic.css ";
</style>
```

or

```
<link rel="StyleSheet"
href="/styles/basic.css"
type="text/css" media="screen" />
```

you'll type `http://www.domain.com/styles/basic.css` into your browser's address bar. Most browsers will

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Where We Go from Here

Markup languages based on XML (see [Chapter 4, "XML Conquers the World \[And Other Web Standards Success Stories\]"](#)) will make today's web look like kindergarten. But we can't get to tomorrow's web by following yesterday's design and development norms.

There are two ways forward: transitional forward compatibility (a savory blend of traditional and standards-based techniques) and strict forward compatibility based on complete (or near-complete) separation of structure, presentation, and behavior.

Transitional forward compatibility accepts the reality of today's mixed browsing environment. It's well suited to projects where branding is a priority and noncompliant browsers make up a significant portion of your audience. Strict forward compatibility, as its name implies, hews closer to the spirit of standards, is the most forward-compatible approach, and provides the greatest benefits when used in appropriate contexts. Let's look more closely at what these approaches entail.

Transitional Forward Compatibility

Ingredients

- Valid XHTML for markup. (HTML 4.01 can also be used.)
- Valid CSS for control of typography, color, margins, and so on.
- Light use of XHTML tables for layouts, avoiding deep nesting by letting CSS do some of the work.
- Optionally: Structural labels applied to significant table cells (facilitates CSS and scripting and helps with next year's transition to table-less CSS layout).
- DOM-based JavaScript/ECMAScript, possibly with code forking to accommodate 4.0 versions of IE and Navigator.
- Accessibility attributes and testing.

Recommended for

Transitional forward compatibility is recommended for sites visited by a high percentage of 4.0 and older browsers that simply aren't up to the job of adequately supporting CSS, let alone the DOM. Also recommended for those occasions in which tables do a better job of layout delivery than CSS. The transitional approach is used by the branch libraries of The New York Public Library [\[2.18\]](#) to accommodate a huge installed base of Netscape 4 users while still complying with the XHTML and CSS standards, with an eye toward accessibility and long-term viability.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Three. The Trouble with Standards

Web standards hold the key to accessible, cost-effective web design and development, but you wouldn't know it from surveying significant commercial and creative sites of the past few years. In this chapter, we'll explore some of the reasons why web standards have not yet been incorporated into the normative practice of all design shops and in-house web divisions and are not yet obligatory components of every site plan or request for proposal.

If you would prefer to read web standards success stories, turn to [Chapter 4](#), "[XML Conquers the World \(And Other Web Standards Success Stories\)](#)." If you're sold on standards and are ready to roll up your sleeves, skip ahead to [Chapter 5](#), "Modern Markup." But if you need help selling standards to your colleagues—or if you simply want to understand how an industry can attain standards without using them—this chapter is for you.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

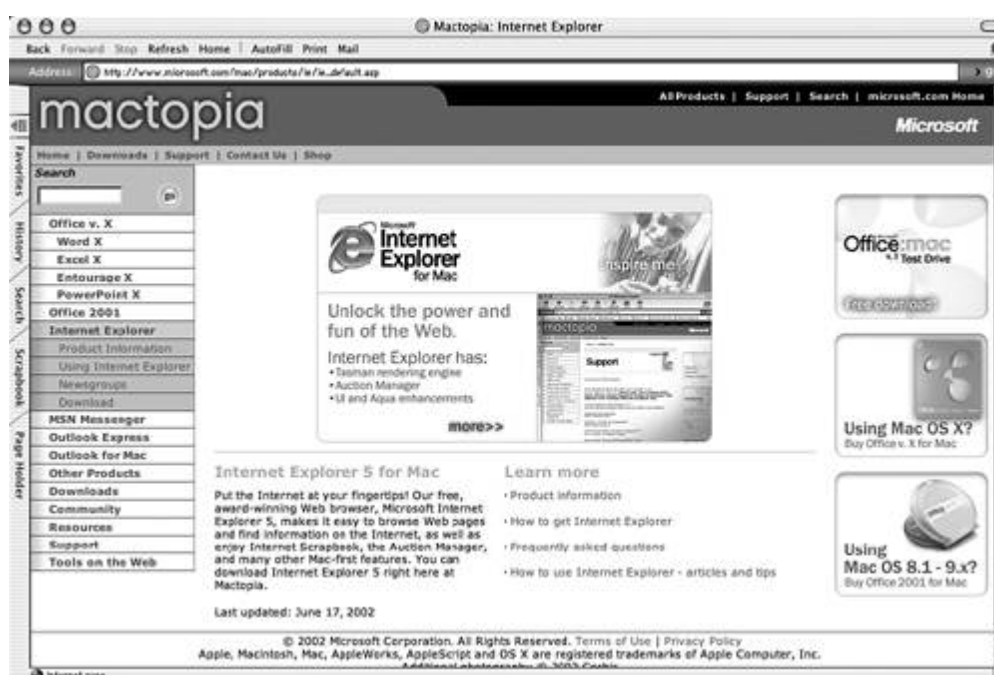
2000: The Year That Browsers Came of Age

With the release in March 2000 of IE5 Macintosh Edition, the world (or at least that portion of the world that uses Macs) finally got more than a teasing taste of web standards. IE5/Mac supported XHTML, ECMAScript, nearly all of the CSS1 specification, much of CSS2, and most of the DOM. IE5/Mac was also able to display raw XML, although it's not clear why anyone would want to do so. (See [Chapter 5](#) for more about this, or visit Bugzilla at http://bugzilla.mozilla.org/show_bug.cgi?id=64945 to see some of the approaches that uber-geeks have taken to the problem of displaying raw XML in browsers.)

IE5/Mac: Switching and Zooming

IE5/Mac was so attuned to standards that it varied its display and performance according to the `<!DOCTYPE>` listed at the top of a web page's markup—a technique called DOCTYPE switching, to be discussed in greater detail in [Chapter 11](#), "Working with Browsers Part I: DOCTYPE Switching and Standards Mode." Put simply, with the right DOCTYPE, a page would display and function as web standards said it should. With an old or partial DOCTYPE (or none), the page would render in backward-compatible "Quirks" mode, to avoid breaking nonstandards-compliant sites—that is, to be kind to 99.9% of commercial sites on the web, at least for now [\[3.5\]](#).

3.5. Hello, world, it's IE5 Macintosh Edition, the first browser to get most web standards mostly right, and one whose innovations found their way into competitive products (www.microsoft.com). Some of those innovations eventually even made their way into IE for Windows. But not all of them, unfortunately.



IE5/Mac also included a feature called Text Zoom [\[3.6\]](#) that enabled users to magnify or shrink any web text, including text set in pixels via CSS, thus solving a long-standing accessibility problem. Prior to IE5/Mac, only Opera Software's Opera browser allowed users to shrink or magnify all web text, including text set in pixels. Opera did this by "zooming" the entire page, graphics and all—an innovative approach to the occasionally epic conflict between what a designer desires and what a user might need [\[3.7, 3.8, 3.9\]](#).

3.6. IE5/Mac's Text Zoom at work. At the touch of a command key or the click of a drop-down menu, users can enlarge (or reduce) the text on any web page, whether that text is set in pixels, points, centimeters, or any other relative or absolute unit. Images on the page are unaffected—only the text size is changed. Text Zoom soon found its way into Netscape, Mozilla, Chimera, and other leading

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Too Little, Too Late?

The release of solidly compliant mainstream browsers was great news for web users and builders. But by the time the glad tidings arrived, many designers and developers were convinced that web standards were a pipe dream, and many had ceased even trying to implement them correctly. It's not hard to understand why. The perception was years in the making.

CSS: The First Bag Is Free

The CSS1 spec had been issued around Christmas of 1996. A few months later, IE3 debuted, including rudimentary CSS support among its features. CSS support (entirely missing in Netscape 3) gave Microsoft's browser its first whiff of credibility at a time when Netscape Navigator dominated the web. IE3 supported just enough CSS to let you dump your nonstandard `` tags and begin experimenting with margins, leading, and other rudiments of CSS layout. Excited by what they saw on Microsoft demo pages [3.10] touting its new browser's capabilities, many designers took their first plunge into CSS design—and soon came up gasping for air.

3.10. A page from Microsoft's 1998 CSS gallery (<http://www.microsoft.com/typography/css/gallery/>).

Overlapping type and all other design effects were created entirely in CSS—no GIF images, no JPEGs.

IE3 could display these effects; Netscape 3 (then the market leader) could not. The gallery's CSS used incorrect values required by IE3's imperfect CSS engine, and its overall standards compliance was nil, but the genie was out of the bottle. Having glimpsed what CSS might do, many of us never looked back.



IE3's CSS support was a bold first step, but like all first steps, it was buggy and incomplete. Those authoring CSS for the first time exulted in the creative freedom it offered but quickly bogged down in early IE bugs that could make a page unusable. For instance, under certain circumstances, images on a CSS-driven page would sit on top of text instead of alongside it. To get an idea of what this was like, place your hand on this paragraph and try to read it through your flesh. Unless you're Claude Rains, you'll have a tough time.

The workaround to this early CSS rendering bug in IE 3 was to place every image and paragraph in its own table cell, thus doubling the weight of your page while defeating the purpose of CSS (to control layout without tables and without excess bandwidth). Designers soon concluded that CSS was not ready for prime time—a determination that seemed reasonable given the absence of any CSS support in market-leading Netscape 3.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Bad Browsers Lead to Bad Practices

Then came the 4.0 browsers. Although still buggy and incomplete, IE4 greatly improved on IE3's CSS support. Netscape 4 offered CSS for the first time in a last-minute implementation so broken and foul it set adoption of CSS back by at least two years.

To be fair, Netscape 4's CSS support was far better than IE3's had been (<http://www.webreview.com/style/css1/leaderboard.shtml>). But while almost nobody today uses IE3, tens of millions still use Netscape 4. Thus, many site owners feel compelled to support Netscape 4—and mistake "support," which is a good thing, with "pixel-perfect sameness and identical behavior," which is a bad thing because it ties developers' hands and forces them to write bad code and dumb markup.

The Curse of Legacy Rendering

Among Netscape 4's chief CSS failings were legacy renderings and lack of inheritance.

Designed to abstract presentation from structure, CSS makes no assumptions about how elements are supposed to be displayed or even what markup language you're going to use, although browsers and other user agents typically do make those assumptions. (Some modern browsers use CSS to enforce their assumptions and allow the designer's style sheets to override them.) By default in most browsers, without CSS, the `<h1>` heading would be big and bold, with vertical margins (whitespace) above and below.

CSS lets you change that. With CSS, `<h1>` can be small, italic, and margin-free if it suits the designer to make it so. Alas, not in Netscape 4, which adds its default legacy renderings to any CSS rule the designer specifies. If the CSS says there shall be no whitespace below the headline, Netscape 4 will go ahead and stick whitespace down there anyway.

When designers applied CSS to standard HTML markup, they quickly discovered that IE4 mainly did what they asked it to do, whereas Netscape 4 made a mess of their layouts.

Some designers abandoned CSS. Others (sadly including me) initially worked around the problem by eschewing structural markup, using constructions like `<div class="headline1">` instead of `<h1>`. This solved the display problem at the expense of document structure and semantics, thereby placing short-term gain ahead of long-term viability and leading to numerous problems down the road. Said problems have now come home to roost.

This author has long since abandoned the practice of wholesale document structural deformation, but a huge proportion of designers and developers still write junk markup in the name of backward compatibility with Netscape 4. This normative practice is fatally flawed, creating usability problems while stymieing efforts to normalize and rationalize data-driven workflows.

Content management systems, publishing tools, and visual web editors (a.k.a. WYSIWYG editors) developed during the 4.0 browser era are littered with meaningless markup that vastly increases the difficulty and expense of bringing sites into conformance with current standards or preparing legacy content for XML-driven databases. On large sites created by multiple designers and developers, each designer might use different nonstandard tags, making it impossible to gather all the data and reformat it according to a more useful scheme. (Imagine a public library where books were indexed, not by the Dewey Decimal System, but according to the whims of Joe, Mary, and various other

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

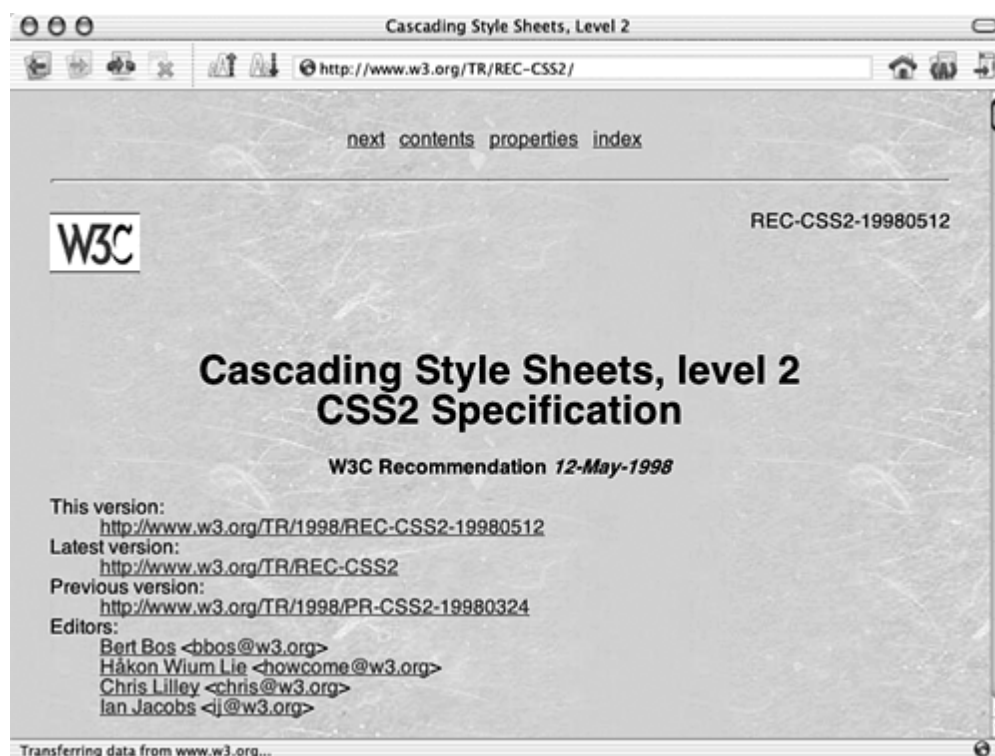
◀ PREVIOUS

NEXT ▶

Confusing Sites, Bewildering Labels

Behold the CSS2 specification as presented by the W3C [3.11]. CSS2 is a powerful standard presentation language created to facilitate the needs of designers, but you wouldn't know it from gazing at this page. It's about as uninspiring a presentation as you've seen since your Uncle Carl showed you the personal site he threw together one afternoon using Microsoft FrontPage and a \$50 image editor.

3.11. The CSS 2 specification per W3C (www.w3.org/TR/REC-CSS2/): An inspiring presentation language whose presentation here is anything but.



Ignoring a gnawing feeling of dread, you attempt to read and comprehend the spec in spite of its unappealing appearance. After all, the W3C is made up of scientists, not graphic designers. All that matters are the words, right? Twenty minutes into your reading experience, cross-eyed and weeping, you surf to an online computer store and buy Macromedia Flash.

To be fair, not only is the W3C not in the business of graphic design, usability consulting, or information architecture, but it's also not in the business of writing designer-friendly tutorials. The W3C site is a series of accurate technical documents created by leading thinkers and expert technologists—and that's all it was ever supposed to be.

In "How to Read W3C Specs" (www.alistapart.com/stories/readspec/), O'Reilly author and WaSP steering committee member (and one of this book's technical editors) J. David Eisenberg explains it this way: "When you seek answers, you're looking for a user manual or user reference guide; you want to use the technology. That's not the purpose of a W3C specification. The purpose of a 'spec' is to tell programmers who will implement the technology, what features it must have, and how they are to be implemented. It's the difference between the owner's manual for your car and the repair manuals."

By definition, the W3C speaks to engineers, not the public. It's not trying to explain or sell the standards it creates. As noted earlier, it doesn't even call them "standards," although that's what they are. (Actually, in recent press materials and on parts of the W3C site, the Consortium has begun using the "s" word instead of the more passive

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

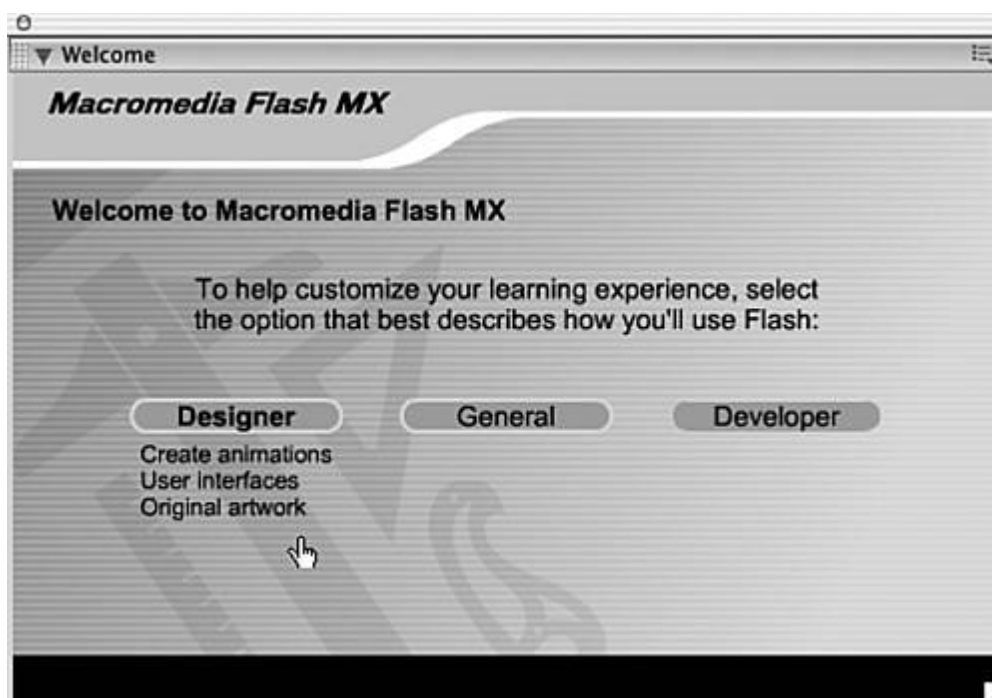
◀ PREVIOUS

NEXT ▶

The F Word

Of all the competing proprietary solutions that corporations have tried to sell, none has succeeded as brilliantly as Macromedia Flash [3.14]. The product began as a humble plug-in called FutureSplash that allowed designers to embed vector-based graphics and animations on their pages.

3.14. Welcome to Macromedia Flash! The Flash authoring environment might be rich, deep, and complex, but Macromedia does all it can to guide designers and developers by the hand as they begin to climb the product's steep learning curves.



Designers paid little attention to FutureSplash, but Macromedia, being the smart company that it is, immediately recognized its potential. Macromedia bought the plug-in and its associated authoring tool, renamed it Flash, and rebuilt it into a richly flexible authoring environment driven by a powerful JavaScript-like programming language called ActionScript.

Macromedia also managed to foster a cult of Flash development.

The Value of Flash

While the incompatible scripting languages and Object Models of the 4.0 browsers wreaked havoc and drove up costs, Flash 4 and its powerful scripting language worked equally well in Navigator, IE, and Opera and nearly as well in Mac OS, Linux, and UNIX as it did in Windows. For many designers, it was adios to HTML, botched 4.0 browser CSS, and rat's nests of incompatible code, and hello baby to Flash.

Spinning logos, tedious "loading" screens, and endless, unwanted "intros" initially gave Flash a bad name among users. But juvenile abuse of the new tool's power eventually gave way to sophisticated user experiences created by the likes of One9ine [3.15], Juxt Interactive [3.16], and other high-end shops. Less talented and less innovative agencies hastily hopped on the Flash bandwagon, often producing far less engaging sites, but you can't blame bad carpentry on the hammer and nails. Flash was eating the rich application space the way Microsoft's browser was eating Netscape's lunch.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Compliance Is a Dirty Word

The other obstacle to widespread acceptance of web standards is the mistaken belief that standards will somehow diminish creativity, tie the developer's hands, or result in lessened user experiences vis-à-vis old-school, proprietary methods. Where does this mistaken notion come from?

It might come from developers who tried using web standards in 4.0 and older browsers and were rightfully displeased with the results. But the days of poor standards compliance are gone.

The Power of Language to Shape Perceptions

The phrase "web standards" might be at fault. The Web Standards Project coined the phrase as an act of propaganda. We sought a set of words that would convey to browser makers exactly what was at stake—a set of words whose underlying ethical imperative would remind browser makers of their commitment to technologies they had helped to create and pledged to support. We needed a phrase that would convey to developers, clients, and tech journalists the urgent importance of reliable, consistently implemented, industry-wide technologies. "Recommendations" didn't cut it. "Standards," we felt, did.

We had no budget and few hopes, yet somehow we succeeded. Today, companies like Netscape, Microsoft, Adobe, and Macromedia strive for standards compliance and brag of it as an expected and desired feature—like four-wheel drive. But although those companies "get it," many in the design community do not. Some mistake "web standards" for an imposed and arbitrary set of design rules (just as some think of usability that way—as do some usability consultants, unfortunately). It should be explained to these designers that web standards have nothing to do with external aesthetic guidelines or commandments.

If not the phrase, "web standards," perhaps the word "compliance" might be at fault. Designers want to comply with their creative visions, not with a complex set of technological rules. They should be told that those rules have nothing to do with the look and feel of any site; they merely enable browsers to deliver what the designer has created. Likewise, clients want to comply with corporate or institution-driven site goals based on marketing requirements and user analysis. Again, web standards can only help by ensuring that sites work for more people on more platforms.

The Inspiration Problem

Designers and clients might be turned off by the lack of visual inspiration (sometimes bordering on hostility to design and branding) found on some sites that discuss web standards or brag about their compliance with one or more W3C specifications. We'll encounter the same problem when we discuss accessibility. (Some accessibility sites are downright ugly, but the problem lies with those sites' designers, not with accessibility, which carries no visual penalty. The same is true for web standards, even if the look and feel of the W3C website or of ECMA is unlikely to motivate designers to get busy learning about XML and CSS2.)

The Wthremix contest [3.17], launched in December of 2002, sought to generate some of that missing aesthetic interest. The founders explained their goals this way:

3.17. The Wthremix contest launched in December 2002 challenged designers and coders to redesign the W3C's website (<http://w3mix.web-graphics.com/>).

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Four. XML Conquers the World (And Other Web Standards Success Stories)

Before we go any further, it's worth countering the previous chapter's bad vibes by discussing the ways web standards have established firm footholds in and beyond the Internet. Despite the misunderstandings that stymie their adoption in some quarters, web standards are winning the day on many fronts and are rapidly changing technology, business, and publishing on and off the web.

In this chapter, we'll take a first look at the most successful web standard since HTML: Extensible Markup Language (XML). XML is an all-embracing data format that's been almost universally adopted and adapted to meet complex needs. We'll discover how XML helps software products remain viable in a rapidly changing market, solves the problems of today's data-driven businesses, and has given rise to a new generation of interoperable web applications and services.

We'll also see how web standards have facilitated détente and encouraged cooperation between arch competitors in the browser business. We'll learn how professional web authoring tools that once ignored standards have come to embrace them. And we'll see how the personal sites of forward-thinking designers and developers have fostered wider acceptance of CSS layout, XHTML markup, and conformance with Web Accessibility Initiative (WAI) and Section 508 accessibility guidelines.

Each of these success stories shares a common core: Standards are gaining acceptance because they work. The more these standards are accepted, the harder and better they will work and the smoother the road will be before us all.

[\[Team LiB \]](#)

◀ PREVIOUS

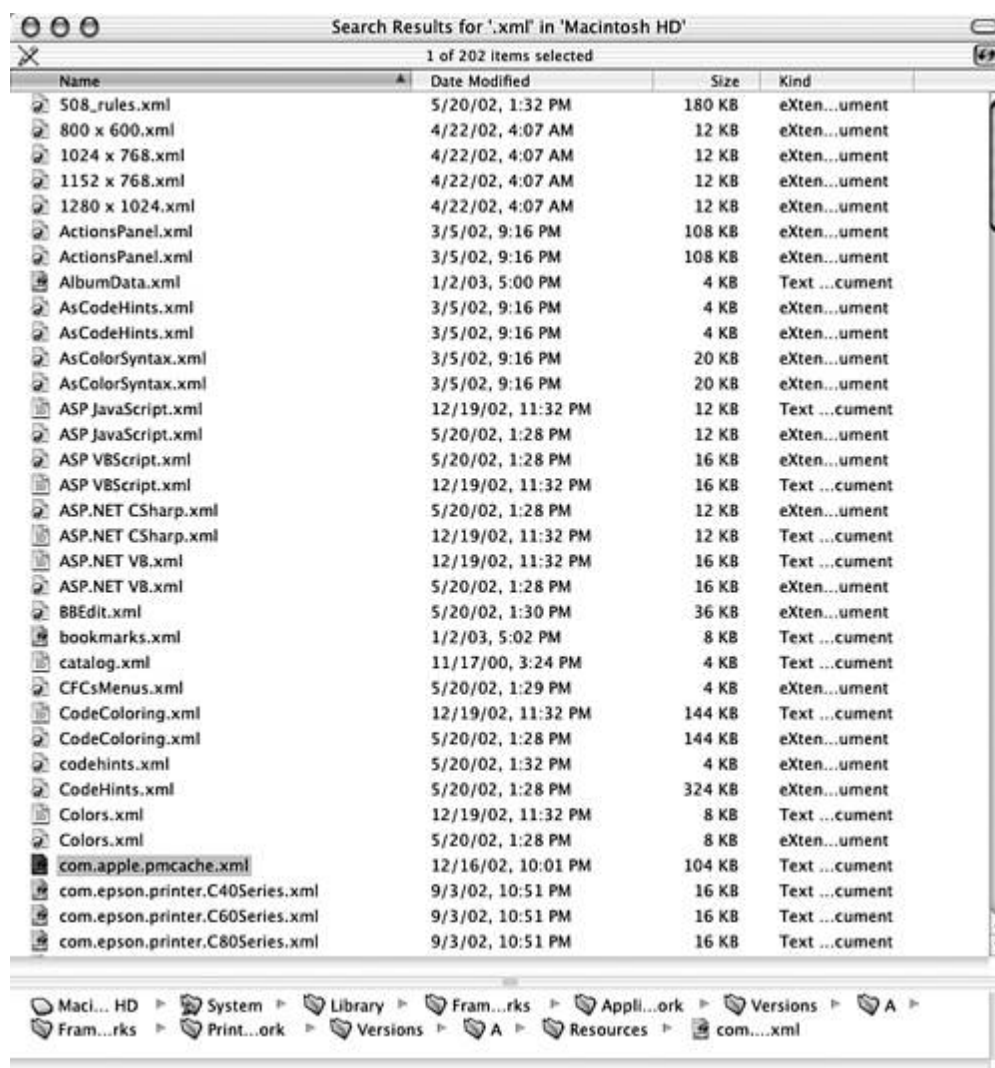
NEXT ▶

The Universal Language (XML)

[Chapter 3](#), "[The Trouble with Standards](#)," described how poor compliance in early browsers persuaded many designers and developers that standards were a pipe dream, leading some to cling doggedly to counterproductive, obsolete methods, and others to embrace Flash to the exclusion of HTML, CSS, and JavaScript. Reading that chapter might have convinced you that web standards face an uphill battle in terms of acceptance and correct usage. Then what are we to make of XML?

The Extensible Markup Language standard (www.w3.org/TR/2000/REC-xml-20001006), introduced in February 1998, took the software industry by storm [[4.1](#)]. For the first time, the world was offered a universal, adaptable format for structuring documents and data, not only on the web, but everywhere. The world took to it as a lad in his Sunday best takes to mud puddles.

4.1. "Mommy, there's XML on my computer!" Run a quick search on an average Macintosh, and you'll find hundreds of XML files. Some store operating system preferences, whereas others drive printers. Still others are essential components of applications including Acrobat, iPhoto, iTunes, Eudora, Internet Explorer, Mozilla, Chimera, Flash MX, Dreamweaver MX, and more. XML is a web standard that goes way beyond the web.



XML and HTML Compared

Although it's based on the same parent technology that gave rise to good old HTML (and just like HTML, it uses

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

XML Applications and Your Site

XML is the language on which Scalable Vector Graphics (www.w3.org/TR/SVG/) and Extensible Hypertext Markup Language (www.w3.org/TR/2002/REC-xhtml1-20020801/) are based. Illustrators who export their client's logo in the SVG format and web authors who compose their pages in XHTML are using XML, whether they know it or not.

The rules that are common to all forms of XML help these formats work together and with other kinds of XML—for instance, with XML stored in a database. An SVG graphic might be automatically altered in response to a visitor-generated search or continuously updated according to data delivered by an XML news feed.

The site of a local TV news channel could use this capability to display live Metro Traffic in all its congested glory. As one traffic jam cleared and another began, the news feed would relay this information to the server, where it would be formatted as user-readable text content in XHTML and as an updated traffic map in SVG. At the same time, the data might be syndicated in RDF or RSS for sharing with other news organizations or used by SOAP to help city officials pinpoint and respond to the problem.

Although based on XML, SVG graphics are easy to create in products like Adobe Illustrator 10 (www.adobe.com/products/illustrator/main.html). Like Flash vector graphics, images created in SVG can fill even the largest monitors while using little bandwidth. And SVG graphics, like other standard web page components, can be manipulated via ECMAScript and the DOM. Not to mention that SVG textual content is accessible by default, and can even be selected with the cursor no matter how it's been stretched or deformed.

Still in Its Infancy

Presently, the power of SVG is somewhat limited by the need to use a plug-in (www.adobe.com/svg/), as with Flash. Nor does the plug-in currently work equally well across platforms and browsers. When browsers natively support SVG, its ability to add standards-based visual interactivity to all websites will be that much more enhanced.

Presently too, browser support for XML is in its infancy. Although XML powers software, databases, and web services, few browsers can usefully display raw XML files, and the creation of XML applications exceeds the capabilities of most designers and site owners.

The development community has solved the latter problem by creating XML-based languages, protocols, and products the rest of us can use "behind the screens." The W3C has solved the problem of browser support for XML by combining the familiar simplicity of HTML with the extensible power of XML in the XHTML markup standard we'll explore in [Chapter 5](#), "Modern Markup."

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Compatible by Nature

Because they share a common parent and abide by the same house rules, all XML applications are compatible with each other, making it easier for developers to manipulate one set of XML data via another and to develop new XML applications as the need arises, without fear of incompatibility.

Ubiquitous in today's professional and consumer software, widely used in web middleware and back-end development, essential to the emerging web services market, and forward compatible by design, XML solves the obsolescence problem described in [Chapter 1](#), "[99.9% of Websites Are Obsolete](#)." XML has succeeded beyond anyone's wildest dreams because it solves everyone's worst nightmares of incompatibility and technological dead ends.

Software makers, disinclined to risk customer loss by being the "odd man out," recognize that supporting XML enables their products to work with others and remain viable in a changing market. Executives and IT professionals, unwilling to let proprietary legacy systems continue to hold their organization's precious data hostage, can solve their problem lickety-split by converting to XML. Small independent developers can compete against the largest companies by harnessing the power of XML, which rewards brains, not budgets.

In today's data-driven world, proprietary formats no longer cut it—if they ever did. XML levels the playing field and invites everyone to play. XML is a web standard, and it works.

And that is the hallmark of a good standard: that it works, gets a job done, and plays well with other standards. Call it interoperability (the W3C's word for it), or call it simple cooperation between components. Whatever you call it, XML is a vast improvement over the bad old days of proprietary web technologies. Under the spell of web standards, competitors, too, have learned to cooperate.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

A New Era of Cooperation

As we might have mentioned a few hundred times, Microsoft, Netscape, and Opera finally support the same standards in their browsers. As an unexpected consequence of their technological cooperation, these once bitter competitors have learned to play nicely together in other, often surprising ways.

In July 2002, Microsoft submitted to the W3C's HTML Working Group "a set of HTML tests and testable assertions in support of the W3C HTML 4.01 Test Suite Development" (<http://lists.w3.org/Archives/Public/www-qa-wg/2002Jul/0103.html>). The contribution was made on behalf of Microsoft, Openwave Systems, Inc., and America Online, Inc., owners of Netscape and Mozilla. Opera Software Corporation (makers of the Opera browser) and The Web Standards Project also reviewed it.

Test Suites and Specifications

W3C test suites enable browser makers to determine if their software complies with a standard or requires more work. No test suite existed for HTML 4.01 (the markup language that is also the basis of XHTML 1.0). In the absence of such a test suite, browser makers wanting to comply with those standards had to cross their fingers and hope for the best.

Moreover, in the absence of a test suite, the makers of standards found themselves in an odd position. How can you be certain that a technology you're inventing adequately addresses the problems it's supposed to solve when you lack a practical proving ground? It's like designing a car on paper without having a machine shop to build what you've envisioned.

In the interest of standards makers as well as browser builders, a test suite was long overdue.

How Suite It Is

When Microsoft took the initiative to correct the problem created by the absence of a test suite, it chose not to act alone, instead inviting its competitors and an outside group (WaSP) to participate in the standards-based effort. Just as significantly, those competitors and that outside group jumped at the chance. The work was submitted free of patent or royalty encumbrance, with resulting or derivative works to be wholly owned by the W3C. Neither Microsoft nor its competitors attempted to make a dime for their trouble.

In the ordinary scheme of things, Microsoft is not known for considering what's best for AOL/Netscape, nor is the latter company overly concerned with helping Microsoft—and neither wastes many brain cells figuring out what's good for Opera. And these companies didn't go into business to lose money on selfless ventures. Yet here they all were, acting in concert for the good of the web, and focusing, not on some fancy new proprietary technology, but on humble HTML 4.

Ignored by the trade press, the quiet event signifies a sea change. We've come a long way from the days when browser makers focused on "our browser only" technologies at the expense of web standards and web users and in hopes of spoiling the market for their competitors. Browser makers still innovate, of course, and they still hope you'll choose their product over a competitor's. But their newfound willingness to work together shows how deeply they are committed to interoperability via web standards and how greatly the industry has changed from the days of the browser wars (1996–1999).

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Web Standards and Authoring Tools

Developed at the height of the browser wars, market-leading, professional visual editors like Macromedia Dreamweaver and Adobe GoLive initially addressed the problem of browser incompatibility by generating markup and code optimized for 3.0 and 4.0 browsers.

When browsers ran on nonstandard, invalid HTML tags, Dreamweaver and GoLive fed them what they needed. If each browser had its own incompatible Document Object Model, driven by its own proprietary scripting language, Dreamweaver and GoLive would code accordingly.

In so doing, Dreamweaver and GoLive were no more at fault than developers who authored the same kind of markup and code by hand. As explained in [Chapter 2](#), "[Designing and Building with Standards](#)," site builders did what they had to do when standards were still being developed and browsers had yet to support them. "Feed 'em what they ask for" made sense in those days but is no longer an appropriate strategy. As browsers shored up their standards support, tools like Dreamweaver and GoLive needed to do likewise. Today they have.

The Dreamweaver Task Force

The Web Standards Project's Dreamweaver Task Force was created in 2001 to work with Macromedia's engineers in an effort to improve the standards compliance and accessibility of sites produced with Dreamweaver, the market leader among professional visual web editors. The Task Force's history can be found at www.webstandards.org/act/campaign/dwtf/.

Among the group's primary objectives as crafted by Rachel Andrew and Drew McLellan were these:

- - Dreamweaver should produce valid markup "out of the box." [Valid markup uses only standard tags and attributes and contains no errors. We'll explain this further in [Chapter 5](#).]
- - Dreamweaver should allow the choice between XHTML and HTML versions, inserting a valid DTD for each choice. [A DTD, or Document Type Definition, tells the browser what kind of markup has been used to author a web page. See [Chapter 5](#).]
- - Dreamweaver should respect a document's DTD and produce markup and code in accordance with it.
- - Dreamweaver should enable users to easily create web documents accessible to all.
- - Dreamweaver should render CSS2 to a good level of accuracy so that pages formatted with CSS can be worked on within the Dreamweaver visual environment.
- - Dreamweaver should not corrupt valid CSS layouts by inserting inline styling without the user's consent.
- - Dreamweaver users should feel confident that their Dreamweaver-created pages will validate and have a high

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Emergence of CSS Layout

The CSS1 specification was created in 1996 to end presentational HTML, separating the way a site looked from the data it contained. By 2000, all popular browsers were finally capable of properly displaying pure CSS layouts. But designers and developers still hesitated to embrace CSS so long as a significant portion of their visitors used noncompliant 4.0 and older browsers. Something had to be done to make web standards "safe" for mainstream designers and developers. The Web Standards Project decided that guerrilla tactics were needed.

The Browser Upgrade Campaign

In February 2001, The Web Standards Project launched its Browser Upgrade campaign (www.webstandards.org/act/campaign/buc/). As its name suggests, the campaign was intended to foster consumer trial of newer, more standards-compliant browsers, thereby also encouraging designers to use standards instead of HTML hacks and proprietary code.

In some cases, the campaign did this by encouraging developers to act as if their entire audience were already using standards-compliant browsers and nothing but. A JavaScript fragment in the <head> of a document could test the DOM awareness of any browser hitting that page. If the browser "got" the DOM, it also got the page. If it didn't, the visitor was bounced to a page his browser could understand.

That page advised the visitor to upgrade his browser to a more recent version of IE, Netscape, Opera, or other compliant browsers, and it also told him why such an upgrade would improve his web experience.

Unlike the "Best Viewed With " marketing campaigns of old, the WaSP's Browser Upgrade Campaign was manufacturer agnostic. We didn't care whose browser you downloaded, as long as that browser complied with standards.

This brute force technique was recommended only for sites that correctly used CSS layout and the DOM, and only for noncommercial or nonessential sites that could afford to risk bouncing visitors in this way. Participating designers and developers were encouraged to build their own "Browser Upgrades" pages, customized to their visitors' needs; to use such techniques only on valid, standards-compliant sites; and to carefully weigh the advantages and disadvantages of taking such a step.

Browser Upgrades Used and Abused

Alas, all too often, lazy developers used the technique to bounce visitors away from sites that were not even close to complying with standards. Compounding the damage, instead of creating their own upgrade pages, these developers invariably sent them to WaSP's. As you might expect, such efforts more frequently resulted in consumer frustration than in the desired trial of compliant browsers.

Among the offenders was a site featuring pictorials of cheerleaders. Raiderettes.com [4.9], a top referrer to The Web Standards Project's Browser Upgrade Campaign page, is a nice-enough looking site, but an attempt to validate its markup (<http://validator.w3.org/>) results in the following report:

```
Fatal Error: no document type declaration; will parse without
validation.
I could not parse this document, because it uses a public
identifier that is not in my catalog
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Mainstreaming of Web Standards

In 2001, the U.S. and Canada published guidelines demanding that government-related sites be developed with web standards and made accessible. The governments of Britain and New Zealand soon followed suit, as did numerous American states.

In 2002, mainstream government-related sites including Texas Parks & Wildlife (www.tpwd.state.tx.us) and the home page of Juneau, Alaska [4.18] converted to web standards including CSS layout. Texas Parks & Wildlife explained its conversion this way (www.tpwd.state.tx.us/standards/tpwbui.htm):

Texas Parks & Wildlife, as a state agency, is required by Texas Administrative Code §201.12 to code web pages using the web standards set forth by the W3C. The driving force behind these requirements is the issue of accessibility. Web pages must be accessible to all users, regardless of disability or technology used to access web pages.

But if the pages do not look the same for everyone, how are they accessible?

Being accessible does not mean that everyone sees the same thing. Accessibility is about content and information. It is about making all content (text, images, and multimedia) available to the user. In order to do this according to the standards, TPW makes use of Cascading Style Sheets to separate content from presentation. Separating presentation from content will enable TPW to offer higher quality pages and dynamic, timely content.

The web coding standards set forth by the W3C make creating accessible pages possible. By using such W3C standards as Cascading Style Sheets and XHTML, Texas Parks & Wildlife has begun to create compliant web pages.

4.18. Welcome to beautiful Juneau, Alaska, home of tourism, mining, fishing, and style sheets (www.juneau.org). Juneau.org was among the first public sites to scrap old-school frames and such for CSS layout and structured XHTML markup.



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Executive Summary

Standards like XML have changed the face of business. Support for standards has ended the browser wars and the toll they took on usability, accessibility, and long-term viability, ushering in a new era of cooperation among even the staunchest competitors. Standards like CSS and XHTML, supported in mainstream browsers and professional authoring tools, are changing site design methods for the better—not only on bleeding-edge independent and personal sites, but also on government, institutional, and commercial sites with no axe to grind beyond improved efficiency and increased outreach. Web standards are winning, and you are there.

Now let's stop exulting and get down to work. [Chapter 5](#) gets the ball rolling and tosses it in your court. (We told you we kind of stink at sports metaphors, didn't we?)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Part II: Designing and Building

[5 Modern Markup](#)

[6 XHTML: Restructuring the Web](#)

[7 Tighter, Firmer Pages Guaranteed: Structure and Meta-Structure in Strict and Hybrid Markup](#)

[8 XHTML by Example: A Hybrid Layout \(Part I\)](#)

[9 CSS Basics](#)

[10 CSS in Action: A Hybrid Layout \(Part II\)](#)

[11 Working with Browsers Part I: DOCTYPE Switching and Standards Mode](#)

[12 Working with Browsers Part II: Box Models, Bugs, and Workarounds](#)

[13 Working with Browsers Part III: Typography](#)

[14 Accessibility Basics](#)

[15 Working with DOM-Based Scripts](#)

[16 A CSS Redesign](#)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Five. Modern Markup

In [Part I](#), "Houston, We Have a Problem," we described the creative and business problems engendered by old-school web design methods, outlined the benefits of designing and building with standards, and sketched a general picture of advances in the medium. The rest of this book will move from the general to the particular, and the best place to start is by taking a second look at the fundamentals of web markup.

Many designers and developers will balk at the thought. Surely those of us who've spent more than a few weeks designing professional sites know all there is to know about HTML. Don't designers and developers have newer, more powerful languages to learn in their limited free time? For instance, isn't it more important to study server-side technologies like PHP, ASP, or ColdFusion (see the sidebar, "[What Is PHP](#)") than piddle precious hours away rethinking rudiments like the HTML table or paragraph tag?

The answer is yes and no. Server-side technologies are vitally important to the creation of dynamic sites that respond to user queries.

Even traditional informational sites can benefit by storing their content in databases and fetching it as needed via PHP or similar technologies. Indeed, like the web standards this book discusses, server-side scripting languages perform a similar function of abstracting data from the interface. As standards like CSS free the designer from the need to imprison each snippet of content in semantically meaningless, bandwidth-bursting table cells, so do languages like PHP and ASP free site creators from the dunderheaded drudgery of creating each page by hand.

But those dynamically generated web pages won't be much use if they are inaccessible, incompatible with a broad range of browsers and devices, or cluttered with junk markup. If those dynamic pages don't render in some browsers and devices or take 60 seconds to load over dial-up when 10 seconds might suffice, the server-side technologies won't be doing all they can for you or your users. In short, it's not an either/or but a both. Server-side technologies and databases facilitate smarter, more powerful sites, but what those sites deliver is content that works best when it is semantically and cleanly structured. And that's where many of us (and many of the content management systems we rely on) fall short.

What Is PHP?

PHP (www.php.net) is an open source, general-purpose scripting language that is ideally suited to web development and can be embedded in XHTML. Its syntax draws upon C, Java, and Perl and is comparatively easy to learn. PHP (which stands, rather confusingly, for PHP: Hypertext Preprocessor) has many capabilities but has become vastly popular for one: When used in combination with a MySQL (www.mysql.com) database, PHP lets designers and developers easily author dynamic sites and build web applications.

PHP is a project of the Apache Software Foundation (www.apache.org) and is free, which is one source of its popularity. (Another is the language's wide array of profiling and debugging tools.) Open source developers and independent web designers are enamored of the language and continually create PHP-based applications they make freely available to their peers. For instance, Dean Allen's Refer ([http://www.refer.com](#)) and its successor, Refer 2.0 ([http://www.refer2.com](#)), are both built with PHP. (Refer 2.0 is a more powerful and flexible version of Refer 1.0.)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Secret Shame of Rotten Markup

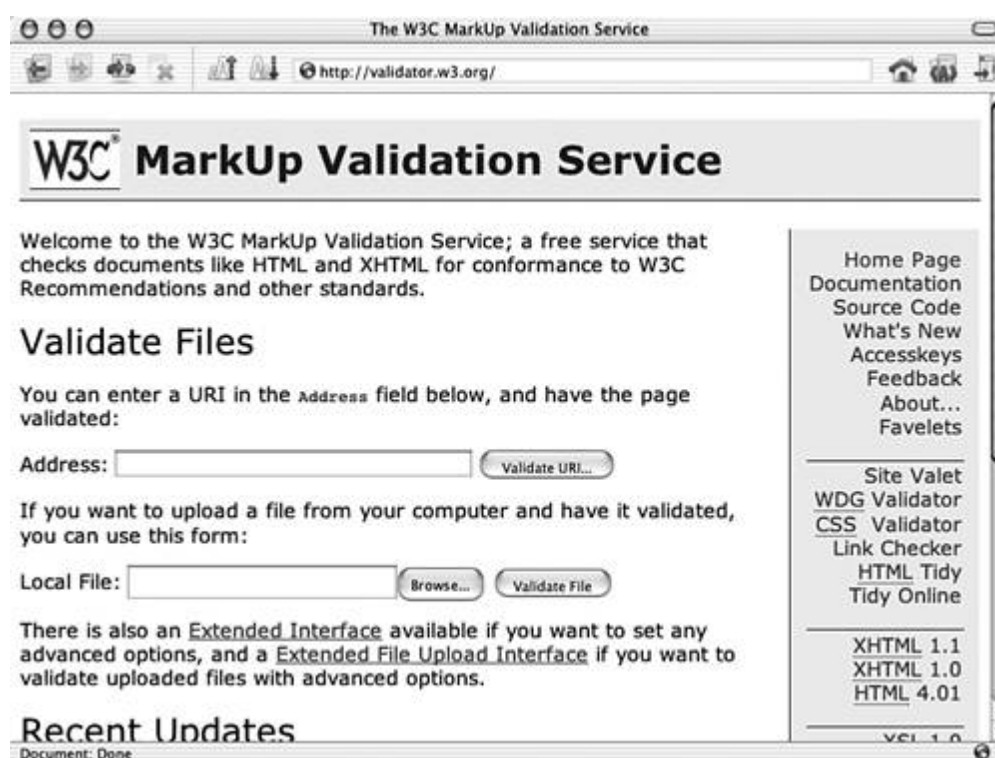
The more successful we've been in the field of web design and the longer we've been at it, the less likely we are to even be aware of the hidden cost of rotten markup. During our industry's first decade, designing for the web was like feeding a roomful of finicky toddlers. To build sites that worked, we dutifully learned to accommodate each browser's unique dietary requirements. Today's browsers all eat the same nutritious stuff, but many professionals haven't grasped this and are still crumbling M&Ms into the soufflé.

Bad food hardens arteries, rots teeth, and diminishes the energy of those who consume it. Bad markup is equally harmful to the short-term needs of your users and the long-term health of your content. But until recently, this fact was hidden from many of us by the high junk code tolerance of most popular browsers, as discussed in [Chapter 1](#), "99.9% of Websites Are Obsolete."

In this chapter and those that follow, we'll investigate the forgotten nature of clean, semantic markup and learn to think structurally instead of viewing web markup as a second-rate design tool. At the same time, we'll examine XHTML, the standard language for marking up web pages, discussing its goals and benefits and developing a strategy for converting from HTML to XHTML.

By a strange coincidence, proper XHTML authoring encourages structural markup and discourages presentational hacks. In XHTML 1.0 Transitional, such hacks are "deprecated," which means you can use them if you must, but you are encouraged to achieve the same design effects in other ways (for instance, by using CSS). In XHTML 1.0 and 1.1, Strict, presentational hacks are actually forbidden: Use them and your page will no longer pass muster when you run it through the W3C's Markup Validation Service, familiarly known as "the Validator" [5.3] (If it's not familiar to you now, it will become so as you learn to design and build with standards. See the "[Validate This!](#)" sidebar.)

5.3. The W3C's free online Validation Service (<http://validator.w3.org/>) is used by thousands of designers and developers to ensure their sites comply with web standards. Did we mention the service is free?



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

A Reformulation of Say What?

According to the W3C, "XHTML (<http://www.w3.org/TR/xhtml1/>) is a reformulation of HTML in XML." In plainer if slightly less precise English, XHTML is an XML-based markup language that works and looks like HTML with a few small but significant differences. To web browsers and other user agents, XHTML works exactly the same way as HTML, although some sophisticated modern browsers might treat it a bit differently, as we'll discuss in [Chapter 6](#), "XHTML: Restructuring the Web." To designers and developers, writing XHTML is almost the same as writing HTML but with slightly tighter house rules and one or two new elements, to be covered next.

In [Chapter 4](#), "[XML Conquers the World \(And Other Web Standards Success Stories\)](#)," we described XML, a.k.a. Extensible Markup Language (<http://www.w3.org/XML/>), as a "super" markup language from which programmers can develop other custom markup languages. XHTML (Extensible Hypertext Markup Language) is one such markup language. XHTML 1.0 is the first and most backward-compatible version of XHTML, hence the most comfortable version to learn and the least troublesome to browsers and other user agents.

Additional applications and protocols based on XML are legion, and their popularity is partially due to their ability to exchange and transform data with relatively little effort and few (if any) compatibility hassles—a virtue they share with XHTML. Among these protocols are Scalable Vector Graphics (<http://www.w3.org/TR/SVG/>), Synchronized Multimedia Integration Language (<http://www.w3.org/TR/REC-smil/>), Simple Object Access Protocol (<http://www.w3.org/TR/SOAP/>), Resource Description Framework (<http://www.w3.org/RDF/>), and Platform for Privacy Preferences (<http://www.w3.org/TR/P3P/>).

Each of these protocols (and countless others) plays an important role in the emerging web, but none is as vital to designers and developers as XHTML—and none is as easy to learn.

Why "reformulate" HTML in XML or anything else? For one thing, XML is consistent where HTML is not. In XML, if you open a tag, you must close it again. In HTML, some tags never close, others always do, and still others can close or not at the developer's discretion. This inconsistency can create practical problems. For instance, some browsers might refuse to display an HTML page that leaves table cells unclosed even though HTML says it is okay not to close them. XHTML compels you to close all elements, thereby helping you avoid browser problems, save hours of testing and debugging, and stop wasting valuable neurons trying to remember which tags close and which don't.

More importantly, XML-based languages and tools are the golden coin of the web's present and the key to its future. If you author your markup in an XML-based language, your site will work better with other XML-based languages, applications, and protocols.

If XML is that important, why create an XML-based markup language that works like HTML? XML is powerful and pervasive, but you can't serve raw XML data to most web browsers and expect them to do anything intelligent with it, such as displaying a nicely formatted web page [\[5.4\]](#). In essence, then, XHTML is a bridge technology, combining the power of XML (kind of) with the simplicity of HTML (mostly).

5.4. A sample XML document (http://p.moreover.com/cgi-local/page?index_xml+xml) as displayed by a modern browser (in this case, Chimera). Some browsers display XML as text, others as text plus tags. Neither approach is particularly useful. By contrast, virtually all browsers and devices know what to do with XHTML, and that makes it both useful and powerful.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Executive Summary

Loosely speaking, XHTML is XML that acts like HTML in old and new web browsers and also works as expected in most Internet devices, from Palm Pilots to web phones to screen readers, making it portable, practical, and cost efficient.

XHTML is as easy to learn and use as HTML—a little easier for newcomers who have no bad habits to unlearn, and perhaps a little harder for old hands who embraced web design and development in the wild and wooly 1990s.

XHTML is the current markup standard (replacing HTML 4), and it's designed to return rigorous, logical document structure to web content, to work well with other web standards, such as CSS and the DOM, and to play well with other existing and future XML-based languages, applications, and protocols. We list all the benefits of XHTML after the short, but important, detour that follows.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Which XHTML Is Right for You?

In this chapter and throughout this book, we will focus on XHTML 1.0 and emphasize XHTML 1.0 Transitional, which is the most forgiving flavor of XHTML, the most compatible with existing development methods, and the easiest to learn and transition to.

Many standards geeks prefer XHTML 1.1 Strict, and there's nothing wrong with it, but it is less compatible with the past and it is properly served with a MIME type that causes some popular current browsers to misbehave. Additionally, converting old-school web pages to XHTML 1.1 Strict takes more work and requires more rethinking than converting to XHTML 1.0 Transitional. For many readers of this book, XHTML 1.0 Transitional is likely to be the best choice for the next few years.

As of this writing, a draft XHTML 2.0 specification has been presented to the development community for comment. As it is currently configured, this new spec moves closer to a semantic ideal and further from existing web development methods. By design, the present XHTML 2.0 draft is not backward compatible with HTML or XHTML 1.0. It abandons familiar conventions including the IMG element (OBJECT is to be used instead), the
 tag (replaced by a new LINE element), and the time-honored anchor link (in whose stead we are presently offered a technology called HLINK). These curiosities might change by the time this book hits the shelves, or they might congeal into the hard shell of reality.

Some developers have greeted the proposed XHTML 2 specification with little yelps of joy, whereas others have greeted it with incensed snorts of derision. Still others have cautiously adopted a "wait and see" policy. And many web professionals grinding away in the trenches know nothing about any of this and are still wondering what the accessibility options in Dreamweaver are for.

It remains to be seen how much of the proposed spec will make it into a final recommendation. It also remains to be seen whether developers and browser makers will rally 'round the flagpole of XHTML 2 or ignore it. Because the proposed spec is far from final and is not yet supported by any browser, its nascent existence is interesting but not yet relevant to this book or our jobs, and we once more recommend XHTML 1.0.

Finally, if the thought that the proposed XHTML 2.0 specification is not backward compatible makes you question whether XHTML is forward compatible, you should know that no browser or device maker has any intention of abandoning support for XHTML 1. For that matter, no browser maker intends to stop supporting HTML 4, despite our dour mutterings about badly authored HTML in [Chapter 1](#) of this book. (In [Chapter 1](#), we were grouching about bad markup and ill-advised scripting methods, not about any web standard—and HTML 4 is a web standard, if an outdated one.) Sites correctly authored to the HTML 4.01 specification will keep working for years to come. The same is true for sites properly authored in XHTML 1. Choosing between these two specs (HTML and XHTML) comes down to 10 key points summarized in the following list.

Top 10 Reasons to Convert to XHTML

1.

XHTML is the current markup standard, replacing HTML 4.

2.

XHTML is designed to work well with other XML-based markup languages, applications, and protocols—an advantage that HTML lacks.

3.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Six. XHTML: Restructuring the Web

We might have titled this chapter, "[XHTML: Simple Rules, Easy Guidelines](#)." For one thing, the rules and guidelines discussed in this chapter are simple and easy. For another, "simple" and "easy" are to web design books what "new!" and "free!" are to supermarket packaging—namely, hackneyed but effective attention-getting devices that stimulate interest and encourage trial.

And we certainly want to stimulate interest in and encourage trial of this chapter. Why? Because after you've grasped the simple, easy ideas in this chapter, you'll rethink the way web pages work—and begin changing the way you build web pages. And we don't mean you'll write this year's tags instead of last year's tags. We mean you'll genuinely think (and work) differently. "[Simple Rules, Easy Guidelines](#)" doesn't begin to cover all that.

On the other hand, "Attain Oneness with the One True Way in a Blinding Flash of Enlightenment," another possible chapter title we kicked around, seemed a bit too thick for its pants. And restructuring is really what XHTML (and this chapter) is about. So, "[XHTML: Restructuring the Web](#)" it is.

In this chapter, we'll learn the ABCs of XHTML and explore the mechanics and implications of structural versus presentational markup. If you've been incorporating web standards into your design/development practice, some of this material will be familiar to you. But even old hands might feel a frisson of surprise as they unlock the hidden treasures of [Chapter 6](#). Okay, frisson might be overstating things a bit. You might just say, "Hey, this is news to me." We'll settle for that.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Converting to XHTML: Simple Rules, Easy Guidelines

Converting from traditional HTML to XHTML 1.0 Transitional is quick and painless, as long as you observe a few simple rules and easy guidelines. (We really can't get enough of that phrase.) If you've written HTML, you can write XHTML. If you've never written HTML, you can still write XHTML. Let's zip through the (simple and easy) basics, shall we? Here are the rules of XHTML.

Open with the Proper DOCTYPE and Namespace

XHTML documents begin with elements that tell browsers how to interpret them and validation services how to test them for conformance. The first of these is the DOCTYPE (short for "document type") declaration. This handy little element informs the validation service which version of XHTML or HTML you're using. For reasons known only to a W3C committee member, the word DOCTYPE is always written in all caps.

Why a DOCTYPE?

XHTML allows designers/developers to author several different types of documents, each bound by different rules. The rules of each type are spelled out within the XHTML specifications in a long piece of text called a document type definition (DTD). Your DOCTYPE declaration informs validation services and modern browsers which DTD you followed in crafting your markup. In turn, this information tells those validation services and browsers how to handle your page.

DOCTYPE declarations are a key component of compliant web pages; your markup and CSS won't validate unless your XHTML source begins with a proper DOCTYPE. In addition, your choice of DOCTYPE affects the way most modern browsers display your site. The results might surprise you if you're not expecting them. In [Chapter 11](#), "Working with Browsers Part I: DOCTYPE Switching and Standards Mode," we'll explain the impact of DOCTYPE in Internet Explorer and in Gecko-based browsers like Netscape, Mozilla, and Camino.

XHTML 1 offers three yummy choices of DTD and three possible DOCTYPE declarations:

- Transitional— The comfortable, slightly frowsy DTD whose motto is "live and let live" (see the next section, "[Which DOCTYPE Is Your Type?](#)")
- Strict— The whip-wielding, mysteriously aloof DTD that flogs you for using presentational markup elements or attributes
- Frameset— The one with the '90s haircut; also the one that lets you use frames in your design—which comes to the same thing

Which DOCTYPE Is Your Type?

Of the three flavors listed in the previous section, XHTML 1.0 Transitional is the one that's closest to the HTML we all know and love. That is to say, it's the only one that forgives presentational markup structures and deprecated elements and attributes.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Character Encoding: The Dull, the Duller, and the Truly Dull

In reading the second rule of XHTML in the preceding section ("[Declare your content type](#)"), you might have asked yourself, "Why should I declare my content type?" You might even have asked yourself, "What is a content type?" The answers to these dull questions appear next. Perhaps you're asking yourself, "Must I really read this tedious stuff?" The answer, of course, is yes. If we had to write it, you have to read it. That's only fair. Plus, you might learn something.

Unicode and Other Character Sets

The default character set for XML, XHTML, and HTML 4.0 documents is Unicode (<http://www.w3.org/International/O-unicode.html>), a standard defined, oddly enough, by the Unicode Consortium (www.unicode.org). Unicode is a comprehensive character set that provides a unique number for every character, "no matter what the platform, no matter what the program, no matter what the language." Unicode is thus the closest thing we have to a universal alphabet, although it is not an alphabet but a numeric mapping scheme.

Even though Unicode is the default character set for web documents, developers are free to choose other character sets that might be better suited to their needs. For instance, American and Western European websites often use ISO-8859-1 (Latin-1) encoding. You might be asking yourself what Latin-1 encoding means, or where it comes from. Okay, to be honest, you're not asking yourself any such thing, but we needed a transition, and that was the best we could do on short notice.

What Is ISO 8859?

ISO 8859 is a series of standardized multilingual single-byte coded (8 bit) graphic character sets for writing in alphabetic languages, and the first of these character sets, ISO-8859-1 (also called Latin-1), is used to map Western European characters to Unicode. ISO 8859 character sets include Latin-2 (East European), Turkish, Greek, Hebrew, and Nordic, among others.

The ISO 8859 standard was created in the mid-1980s by the European Computer Manufacturer's Association (ECMA) and endorsed by the International Standards Organization (ISO). Now you know.

Mapping Your Character Set to Unicode

Regardless of which character set you've chosen, to map it to the Unicode standard, you must declare your character encoding, as discussed in the second rule of XHTML presented earlier. (You see, there was a point to all this.) Sites can declare their character encoding in any of three ways:

- A server administrator ("systems guy") might set the encoding via the HTTP headers returned by the web server. The W3C recommends this approach, but it is rarely followed—maybe because systems guys would rather play networked games than muck around with HTTP headers. Come to think of it, who wouldn't?
- For XML documents (including XHTML), a designer/developer might use the optional XML prolog to specify the encoding. This, too, is a W3C recommendation, but until more browsers learn to handle it properly, we can't recommend it.
-

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Structural Healing—It's Good for Me

Developing in XHTML goes beyond converting uppercase to lowercase and adding slashes to the end of `
` tags. If changing "tag fashions" were all there was to it, nobody would bother, and instead of web standards, this book would be filled with delicious tofu recipes. Like tofu honey pie with blueberries. Yum! It's even better if you use cream cheese instead of tofu. And sugar—lots of sugar.

And butter and eggs—don't forget the eggs. But we digress. To benefit from XHTML, you need to think about your markup in structural rather than visual terms.

Marking Up Your Document for Sense Instead of Style

Remember: To the greatest extent possible, you want to use CSS for layout. In the world of web standards, XHTML markup is not about presentation; it's about core document structure. Well-structured documents make as much sense to a Palm Pilot or screen reader user as they do to someone who's viewing your page in a fancy-pants graphical desktop browser. Well-structured documents also make visual sense in old desktop browsers that don't support CSS or in modern browsers whose users have turned off CSS for one reason or another.

Not every site can currently abandon HTML table layouts. The W3C, inventors of CSS, did not convert to CSS layout until December 2002. Moreover, even die-hard standards purists might not always be able to entirely separate structure from presentation—at least not in XHTML 1. But that separation of structure from presentation (of data from design) is an ideal toward which we can make great strides and from which even hybrid, transitional layouts can benefit. Here are some tips to help you start to think more structurally.

Color Within the Outlines

In grammar school, most of us were forced to write essays in a standard outline format. Then we became designers, and, oh, how free we felt as we cast off the dead weight of restrictive outlines and plunged boldly into unique realms of personal expression. (Okay, so maybe our brochure and commerce sites weren't that unique or personal. But at least we didn't have to worry about outlines any more. Or did we?)

Actually, according to HTML, we should always have structured our textual content in organized hierarchies (outlines). We couldn't do that and deliver marketable layouts in the days before browsers supported CSS, but today we can deliver good underlying document structure without paying a design penalty.

When marking up text for the web or when converting existing text documents into web pages, think in terms of traditional outlines:

```
<h1>My Topic</h1>
<p>Introductory text</p>
<h2>Subsidiary Point</h2>
<p>Relevant text</p>
```

Avoid using deprecated HTML elements such as `` tags or meaningless elements like `
` to simulate a logical structure where none exists. For instance, don't do this:

```
<font size="7">My Topic</font><br />
Introductory text <br /><br />
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Visual Elements and Structure

"Web standards" abide not only in the technologies we use, but also in the way we use them. Writing markup in XHTML and using CSS to handle some or all layout chores doesn't necessarily make a site any more accessible or portable or any less of a bandwidth burster. XHTML and CSS can be misused or abused as easily as earlier web technologies were. Verbose XHTML markup wastes every bit as much of the user's time as verbose HTML ever did. Long-winded, overwrought CSS is not an adequate replacement for presentational HTML; it's simply one bad thing taking the place of another.

The guidelines in the earlier "[Structural Healing](#)" section can help avoid overly complex, semantically meaningless interior structures (body copy and so on). But what do we do about branded visual elements, such as site-wide navigation bars, which typically include a logo? Can these elements be structural? And must they be?

The answer to the first question is yes—visual elements including navigation bars can indeed be structural. For example, as mentioned in the previous section, CSS can make an ordered or unordered list display as a full-fledged navigation bar, complete with push buttons and rollover effects.

The answer to the second question is that visual elements like navigation bars need not be made structural in hybrid, transitional layouts. If those layouts avoid verbosity and use good structure in their primary content areas, if their XHTML and CSS validate, and if every effort has been made to provide accessibility, then you will have achieved transitional standards compliance and will have nothing to be ashamed of. (Well, you might have something to be ashamed of, but it won't be the way your website was built.)

In the next chapter, we examine the difference between good and bad authoring (whether that bad authoring happens to use XHTML and CSS or not) and talk about how to create good, clean, compact markup in nonstructural components of hybrid, transitional layouts.

Chapter Seven. Tighter, Firmer Pages Guaranteed: Structure and Meta-Structure in Strict and Hybrid Markup

Whatever you do, don't skip this chapter. Reading it will improve your skills, trim unwanted fat from your web pages, and sharpen your understanding of the difference between markup and design. The ideas in this chapter are easy to follow but can make a profound difference in the performance of your sites and the facility with which you design, produce, and update them.

In this chapter, we'll show how to write logical, compact markup that can lower your bandwidth by 50% or more, restoring pep and vigor to your site's loading times while reducing server aches and stress. We'll achieve these savings by banishing presentational elements from our XHTML and learning to avoid many common practices that are frankly no good at all.

These bad practices afflict many sites on the web, especially those that incorporate some CSS into primarily table-based layouts. This is almost always done wastefully and ineptly, even when the designers are quite skilled in everything else they do. It is an equal opportunity problem, as likely to appear in hand-coded sites as in those created with the assistance of visual editing tools like Dreamweaver and GoLive.

In this chapter, we will name these common mistakes so that you can recognize and guard against them, and we will learn what to do instead. We will also make friends with the unique identifier (id) attribute, the "sticky note" of standards-based design, and show how it allows you to write ultra-compact XHTML, whether you're creating hybrid or pure CSS layouts.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Must Every Element Be Structural?

At the conclusion of [Chapter 6](#), "[XHTML: Restructuring the Web](#)," we sassily asserted that it was often okay if navigational elements in transitional sites were not always structural. We'll go further out on a limb and add that even in CSS layouts, some components need not be structural—at least, not in the "headline, paragraph, list" sense discussed in [Chapter 6](#).

These elements can, however, be meta-structural. That is, they can be marked up via generic structural elements and specific structural attributes that indicate the larger structural role they play within the site's design. This is not done for purposes of theory, but to gain real-world benefits including reduced bandwidth and easier site maintenance.

The Daily Report at [zeldman.com](http://www.zeldman.com/daily/) [7.1] uses CSS for layout and XHTML for structure. And nearly all elements that go into its markup are structural, from lists to paragraphs to the address tag that's used to denote—you guessed it—the address.

7.1. The Daily Report at [zeldman.com](http://www.zeldman.com/daily/) uses CSS for layout and XHTML for structure. The navigational graphics at the top are not structural. Or are they?



The navigational graphics at the top of the page are not structural in the sense described by [Chapter 6](#). They are merely linked images, held in place by an XHTML block level element (a `div`) to which the site's clever designer has assigned a name by means of the unique identifier (`id`) attribute. We'll define our terms next and then go on to see how these components work together to create a meta-structure and to reduce page weight and control layout without resorting to presentational markup.

div, id, and Other Assistants

This chapter and those that follow make much reference to the `div` element and the `id` attribute. Used correctly, `div` is the Hamburger Helper of structured markup, whereas `id` is an amazing little tool that permits you to write highly compact XHTML, apply CSS wisely, and add sophisticated behavior to your site via the standard Document Object

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Hybrid Layouts and Compact Markup: Dos and Don'ts

Fear of learning CSS layout or the inability to use it on a particular project is no reason to avoid web standards. Combine streamlined table layouts with basic CSS and structural, accessible XHTML, and you have the makings of a hybrid, transitional approach that works.

CSS is still evolving, browsers are still learning to support CSS1 and CSS2, and some layout ideas work better when their basic elements are held in place by simple XHTML tables. But table markup is not the same as stupid markup, and hybrid layouts that use a few nonstructural components are quite different from the needlessly ornate junk HTML with which most sites are still being built in 2003.

A Note of Caution

We have said that some interface components of hybrid sites might not be structural. But that does not mean that structural markup has no place on the rest of such sites; paragraphs should still be marked up as paragraphs, lists as lists, and so on. Nor is it an invitation to produce navigational menus or other components using every sloppy old trick in the book. Structural or not, every component should be produced with clear, compact markup and clean CSS. Clean markup is what this chapter is all about.

Giving Names to Bad Things

In the section that follows, we'll look at the dumb ways too many sites are built and explain why these methods are counterproductive. We'll also concoct labels for several especially unsavory techniques that are used too often. Reading this section will not only banish these bad habits from your work, but it will also help you spot these errors in the work of your colleagues and vendors.

After you've read this section of the chapter and engraved its simple lessons on your heart, when colleagues or vendors try to get away with certain kinds of foolish markup, you will call them on it with the chillingly apt descriptive labels invented in this chapter. Your colleagues and vendors will develop a newfound respect for markup, a newfound respect for you, and above all, a profound discomfort whenever you're around them.

Actually, we've named the bad authoring practices described next not to make fun of anyone, but simply to identify and banish these practices from our own work. We hope they help you, too.

Common Errors in Hybrid Markup

Consider a typical, table-based site design [\[7.2\]](#), complete with a menu bar [\[7.3\]](#) that changes state [\[7.4\]](#) to indicate the visitor's location as she moves from page to page. The site's logo will obviously be a graphic element, most likely a GIF image. The menu labels (EVENTS, ABOUT, CONTACT, and so on) can also be images, or they can be simple hypertext links. If hypertext is chosen, old-school web designers might litter their markup with font tags to control textual appearance (size, face, color) and outdated attributes to control the alignment, border properties, and background color of the table cells containing these menu labels. Modern designers would use CSS instead, but the results might be no better if approached the wrong way.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Outdated Methods on Parade

For reference purposes (and in case you still use some of these methods), we'll wrap up this chapter by examining outdated production methods in the order in which they first came to the web.

The Year of the Map

In the earliest days of commercial web design, image maps were used to deliver a branded navigation bar such as the one in [Figure 7.3](#) or at the top and bottom of [Figure 7.5](#). Early image maps consisted of five parts:

- A Photoshop (or Canvas, or other image editor generated) file that was quite literally a picture of a user interface, saved in GIF or (later) JPEG format
- A map file containing the coordinates of each "active" region of the image, along with the URL to which each active region linked
- A CGI program, typically written in PERL and installed in a special directory of the server, whose job was to translate the user's mouse clicks into the URLs specified in the map file
- An "ISMAP" attribute that was added to the image element in the page's source code
- A hypertext anchor around the image element that pointed to the CGI program's location

The HTML typically looked something like this:

```
<A HREF="/cgi-bin/imagemap/image.map"><IMG SRC="/images/image.gif" ISMAP></A>
```

This was the famous server-side image map of the early mid-1990s, so named because the CGI script on the site owner's server did the work of translating user mouse clicks to mapped URLs. Of course, the designer did a lot of work as well. Image map design was no picnic, but it was the only way a designer could create the visual effect desired.

Server-side image maps were soon replaced by client-side image maps, so named because the translation of clicks to coordinates took place "on the client"—that is, in the visitor's browser—instead of on the server. Client-side image maps did not require a separate map file; instead, the coordinates were encrusted right in the page's HTML, where they might look something like this:

```
<map name="navigation">
<area shape="rect" coords="0,400,75,475" href="index.html">
<area shape="rect" coords="401,500, 425, 525" href="events.html">
...
</map>
```

Client side imagemaps were easier, and so, of course, server-side image maps went by the wayside. Ain't progress

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Eight. XHTML by Example: A Hybrid Layout (Part I)

This chapter and the two that follow it form a tight little unit. In this chapter, we'll roll up our sleeves and apply what we've learned about XHTML thus far to mark up a real-world design project. The markup we create will be partly structural, partly transitional, and fully standards-compliant. In [Chapter 9](#), "CSS Basics," we'll cover Cascading Style Sheets (CSS) basics for beginning and intermediate users. Finally, in [Chapter 10](#), "CSS in Action: A Hybrid Layout (Part II)," we'll learn still more about CSS while using it to complete our project. This "teaching by doing" business is not unlike learning to swim by being tossed into deep, cold water, although we prefer to think of it as picking up French by visiting Paris. For good measure, as we build this project, we'll start learning how to incorporate accessibility into our markup (and hence, into our sites).

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Benefits of Transitional Methods Used in These Chapters

In this chapter, we'll begin crafting a hybrid, transitional layout combining traditional (but here, streamlined) table layout techniques with structured textual markup and accessibility enhancements. The techniques used in this project and explained in these three chapters are ideal for libraries and other public institutions, along with small companies and any other organization that seeks to do the following:

- Manage large amounts of content on a limited budget
- Support a wide range of browsers and devices
- Conserve visitors' bandwidth (and their own)
- Begin the transition to web standards with publishing methods that are reliable, cost-effective, and easy to implement

Style Sheets Instead of JavaScript

By the end of these three chapters, we will have produced a standards-compliant template for the i3Forum site [8.1]. The final templates created in these chapters can be viewed on the Happy Cog staging server at <http://i3.happycog.com/>. The finished site, produced by means of these templates, is located at <http://i3forum.com/>.

8.1. The finished template, as it will appear when the work explained in Chapters 8 through 10 has been completed.



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

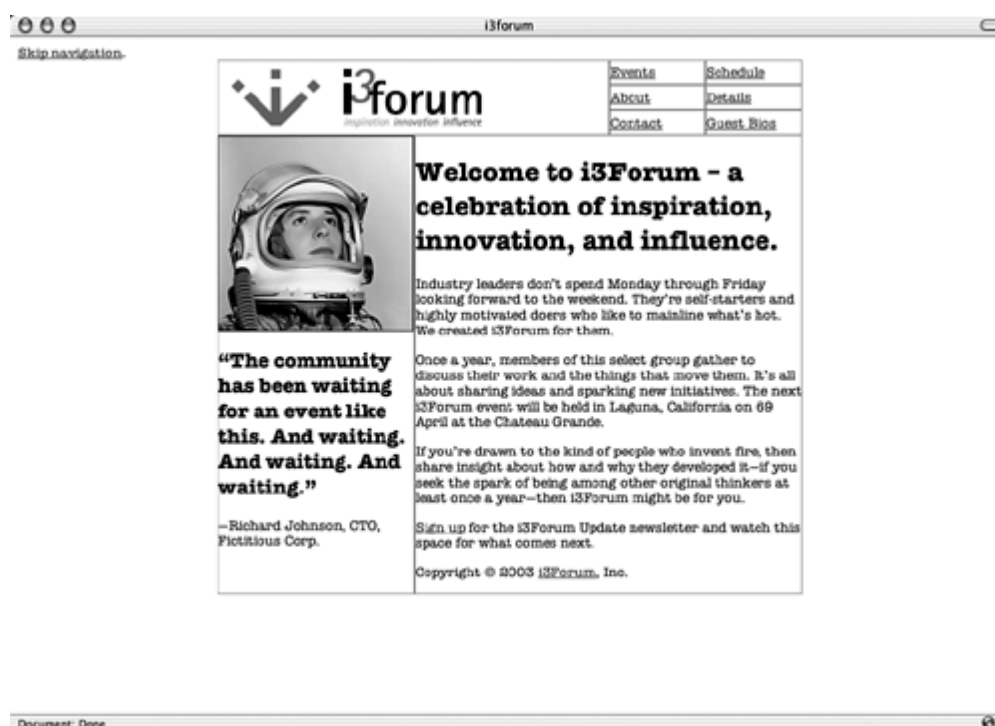
◀ PREVIOUS

NEXT ▶

Basic Approach (Overview)

The i3Forum layout is designed to deliver a crisp, punchy brand identity with a minimum of fuss, and its XHTML is equally straightforward. It is composed of two XHTML tables, both centered, and both enhanced and controlled via CSS. The first table delivers the navigational menu; the second provides the content [8.2].

8.2. The template we'll build in this chapter, with CSS turned off and borders turned on. Note the slightly thicker line between menu and content areas, where two separate tables meet.



The XHTML for the tables will be shown in the pages that follow. But a preliminary question might already have occurred to you. Traditionally, such layouts would use a single table, with rowspans and colspans juggling the various rows and columns. If we used Adobe ImageReady to automatically slice and dice the Photoshop comp used to design the site (and to sell the design to the client), ImageReady would render the entire page in a single table. So why have we used two tables?

Separate Tables: CSS and Accessibility Advantages

If you skipped [Chapter 7](#)'s ("Tighter, Firmer Pages Guaranteed: Structure and Meta-Structure in Strict and Hybrid Markup") discussion of "div, id, and Other Assistants," you might want to glance at it before going any further. Breaking our layout into two tables allows us to harness the power of the id attribute to do the following:

- Streamline the CSS we'll create in [Chapter 10](#)
- Provide certain accessibility enhancements

Structurally label each table according to the job it does, making it easier to some day revisit the layout and replace presentational XHTML tables with divs styled via CSS

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

First Pass Markup: Same as Last Pass Markup

On this and the next few pages, you'll find our first pass at the site's markup from `<body>` to `</body>`. It is also our final pass at the site's markup. Any subsequent design adjustments were handled entirely in CSS. That is one benefit of offloading as much presentational stuff as possible to your style sheets, even in a hybrid layout like this one. To save space in this book, we've replaced the lovely body copy used in the template with generic placeholder text.

Note, too, that in this markup we've used relative image file reference links (`img src="images/logo.gif"`) instead of absolute ones (`img src="/images/logo.gif"`) because we're working off our desktop instead of on the staging server. The final markup will use absolute file reference links. (Absolute URLs are more reliable than relative URLs because they don't break if file locations change; for instance, if `/events.html` moves to `/events/index.html`, the absolute reference to `/images/logo.gif` will still work. Also, absolute URLs help avoid a CSS bug in some old browsers that misunderstand relative file references in style sheets.)

Technically speaking, the "final" markup differs slightly from the first pass markup by replacing relative file references with absolute file references. Not that most of you care, but there is always one reader who views page source to verify claims made in a book.

You might find it easier to view source at the source. As mentioned earlier in this chapter, the project is archived at <http://i3.happycog.com/>.

Navigational Markup: The First Table

What follows is the navigation section, from the body element on. To keep things interesting, we'll tell you in advance that this portion of the markup, although it validates, commits a "sin" of nonpresentational markup purity. See if you can spot the sin.

[\[View full width\]](#)

```
<body bgcolor="#ffffff">
<div class="hide"><a href="#content" title="Skip navigation." accesskey="2">Skip
➡navigation</a>.</div>
<table id="nav" summary="Navigation elements" width="600" border="0" align="center"
➡cellpadding="0" cellspacing="0">
<tr>
<td rowspan="3" id="home" width="400"><a href="/" title="i3Forum home page."></a></td>
<td width="100" height="25" id="events"><a href="events.html">Events</a></td>
<td width="100" height="25" id="schedule"><a href="schedule.html">Schedule</a></td>
</tr>
<tr id="nav2">
<td width="100" height="25" id="about"><a href="about.html">About</a></td>
<td width="100" height="25" id="details"><a href="details.html">Details</a></td>
</tr>
<tr id="nav3">
<td width="100" height="25" id="contact"><a href="contact.html">Contact</a></td>
<td width="100" height="25" id="guestbios"><a href="guestbios.html">Guest Bios</a></td>
</tr>
</table>
```

Presentation, Semantics, Purity, and Sin

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Nine. CSS Basics

In [Chapter 8](#), "XHTML by Example: A Hybrid Layout (Part I)," we began creating a hybrid, transitional, standards-compliant site. In [Chapter 10](#), "CSS in Action: A Hybrid Layout (Part II)," we'll finish the job with Cascading Style Sheets (CSS). But a few basics must be covered first. That's this chapter's job. In this chapter, we'll move swiftly through the rudiments of CSS grammar, cover a few not-so-basic ideas, and end by describing a CSS design method that is different from the one used by most designers and taught in most books. Even if you're familiar with CSS, you might want to stick around.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

CSS Overview

The W3C rather crisply defines CSS as "a simple mechanism for adding style (for example, fonts, colors, spacing) to web documents" (www.w3.org/Style/CSS/). A few details omitted from that summary are worth noting:

-
- CSS is a standard layout language for the web—one that controls colors, typography, and the size and placement of elements and images.
-
- Although precise and powerful, CSS is easy to author by hand, as this chapter will show.
-
- CSS is bandwidth friendly: a single CSS file can control the appearance of an entire site, comprising thousands of pages and hundreds of megabytes.
-
- CSS has long been intended by its creators (W3C) to replace HTML table-based layouts, frames, and other presentational hacks, but as we'll see in the next chapter, it can also be highly effective in hybrid, transitional layouts.
-
- Pure CSS layout, combined with structural XHTML, can help designers separate presentation from structure, making sites more accessible and easier to maintain, as described in the next section, "[CSS Benefits](#)".

CSS Benefits

A Russian proverb states, "Repetition is the mother of learning." So forgive us if we wax a tad repetitive in reminding you that CSS, like other web standards, was not created for abstract purposes and was not intended for some distant future. Used well, right now, CSS provides practical benefits including (but not limited to) these:

-
- Conserves user bandwidth, speeding page load times, especially over dial-up.
-
- Reduces owner server and bandwidth overhead, thus saving money. (See the section "[Outdated Markup: The Cost to Site Owners](#)" in [Chapter 1](#), "99.9% of Websites Are Obsolete.")
-
- Reduces design and development time. Producing the site shown in [Chapters 8](#) and [10](#) took only a couple of hours, and part of that time we were working on [Chapters 8](#) and [10](#), not the site. (These savings pertain only to time spent on development, of course: Creating content and artwork still takes as long as it takes.)
-
- Reduces updating and maintenance time:
 -
 - Content folks no longer need to worry about complex tables, font tags, and other old-school layout components that can break when text is updated. Because there are no (or few) such elements, there is little or nothing to break.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Anatomy of Styles

In this section, we'll introduce you to the thighs, wings, and drumsticks of CSS. Well, anyway, we'll introduce you to the thighs and drumsticks. This book is not a full-blown CSS reference manual. A CSS reference manual could exceed the length of this book, although our favorite CSS reference is small enough to fit in your pocket—see the previous sidebar, "[CSS by the Book\(s\)](#)." On the other hand, how many full-blown CSS reference manuals use the word "thighs" three times in one paragraph? You're right, none of them do. Your money was well spent on this book.

Our anatomy of styles begins next. We'll learn still more about CSS by applying it in [Chapter 10](#), and we'll keep talking (and learning) about CSS as the rest of this book unfolds.

Selectors, Declarations, Properties, and Values

A style sheet consists of one or more rules that control how selected elements should be displayed. A CSS rule consists of two parts: a selector and a declaration:

```
p      { color: red; }
```

In the preceding rule, `p` is the selector, whereas `{ color: red; }`, contained within curly braces, is the declaration.

Declarations, in turn, also consist of two parts: a property and a value. In the earlier declaration, `color` is the property, `red` the value.

Choices and Options

Instead of the English word `red`, we might have written the hexadecimal (web color) value of `#ff0000`:

```
p      { color: #ff0000; }
```

We might save a few bytes by using CSS shorthand that means exactly the same thing:

```
p      { color: #f00; }
```

We could also have used RGB in either of two ways:

```
p      { color: rgb(255,0,0); }  
p      { color: rgb(100%,0%,0%); }
```

Zero Is Optional, Except When It's Not

Notice that when you're using RGB percentages, the percent sign appears even when the value is zero. This is untrue in other CSS situations. For instance, when specifying a size of 0 pixels, the 0 need not be followed by `px`.

Many of us consider it bad form to write `0px` or `0in` or `0pt` or `0cm`. Zero is zero. Who cares what the unit of measurement is when its value is zero? But when specifying RGB percentages, the value of zero requires a percentage sign. Don't ask us; we just work here. Like the rule forbidding underscores in class and id names (see

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

External, Embedded, and Inline Styles

Style sheets can be applied to a web page in any of three ways: external, embedded, or inline. We'll start with the best first.

External Style Sheets

An external style sheet (CSS file) is a text document that lives separately from the XHTML pages it controls. An XHTML page uses that CSS file by referring to it via a link in the head of the document or by importing it into the style element (also in the head of the document). A style sheet link looks like this:

```
<link rel="StyleSheet" href="/styles/mystylesheet.css" type="text/css" media="all" />
```

The @import directive, used to import a style sheet, looks like this:

```
<style type="text/css" media="all">
@import "/styles/mystylesheet.css";
</style>
```

or this:

```
<style type="text/css" media="all">
@import url("/styles/mystylesheet.css");
</style>
```

Greatest Power, Lowest Cost

Whether linked or imported, external style sheets provide the greatest power at the lowest cost. After an external style sheet has been downloaded to the user's cache, it remains active and can control the design of one, dozens, hundreds, or even hundreds of thousands of pages on the site without requiring an additional download. That's mighty handy.

Linking and Importing as Browser Selectors

Virtually all CSS-capable browsers support the link method, including old browsers whose CSS support is less than stellar. By contrast, the @import method works only in 5.0 and later browsers. Thus, designers can use @import to hide style sheets from 4.0 browsers.

This does not necessarily mean hiding all CSS from old browsers, as was discussed in [Chapters 2](#), "Designing and Building with Standards," and [4](#), "XML Conquers the World (And Other Web Standards Success Stories)." Indeed, just the opposite is true: The fact that some browsers "get" @import and others don't enables us to serve them all. How does that work?

Because more than one style sheet can be used on a site, designers can place basic styles in a linked style sheet and sophisticated styles in an imported sheet. The linked, basic style sheet, visible to all browsers that even pretend to understand CSS, provides basic design elements such as fonts used, and text, background, and link colors. The imported sheet, whose styles are visible only to more compliant browsers, contains additional CSS rules that new browsers understand and old browsers would bungle.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The "Best-Case Scenario" Design Method

In the old days, when we created layouts almost entirely with presentational markup, we would test our work in the oldest, crummiest browser on our hard drive. To make it look right in that old browser, we'd build deeply nested tables; use nonstructural divs in place of structural elements like h1, h2, li, and p; and do all the other things we don't want to do any more. When the site looked right in the bad old browser, we would test it in a new browser, where it also quite likely looked good—but at a terrible cost to bandwidth and semantics. Many web designers still follow this practice of designing for the worst browser they can get their hands on. But the cost is too high; the method is no longer productive.

Instead, write your CSS in an embedded style sheet and preview your work in a browser you trust, such as Mozilla, Chimera, Netscape 7, IE6/Windows, IE5/Mac, or Opera 7 (to name a few good ones). In this way, you'll create accessible, low-bandwidth, compliant pages that let markup be markup and that use CSS correctly.

When you're satisfied with what you've designed, test the page in other good, compliant browsers. It should look the same in all of them. If it doesn't, you have more work to do. One of your CSS rules might be incorrectly written, yet your favorite browser understood what you intended, as a friend sometimes understands you when you talk with your mouth full.

From Embedded to External Styles: The Two-Sheet Method

When your site looks and works right in all the compliant browsers at your disposal, it's still not time to open your worst old browser. Instead, copy your CSS rules to a new file called `basic.css`, upload the file to the `/css/` directory on your server, delete the embedded style sheet from your page, and link to it from the head of your document:

```
<link rel="StyleSheet" href="/css/basic.css" type="text/css" media="all" />
```

Empty your caches, quit and restart your browsers, and test again to make certain you haven't accidentally deleted rules when moving from embedded to external CSS.

Testing In and Supporting Noncompliant Browsers

If you're still happy, now is the time to open any noncompliant browsers you want to support. There's a slight chance the site might look okay in your worst browser. If it doesn't, create a new, empty document, call it `sophisto.css`, and begin transferring those CSS rules whose sophistication you suspect of tripping up the bad old browser. As you copy suspiciously sophisticated rules to `sophisto.css`, delete them from `basic.css`. (The names, of course, can be anything. We like `basic` and `sophisto`, but any clear names will do.)

Upload `sophisto.css` to your `/css/` subdirectory and link to this second external style sheet by means of `@import`:

```
<style type="text/css" media="all">@import "/css/sophisto.css";</style>
```

Empty your bad browser's cache, reload the page, and see if the display is now acceptable. It might be. If it's not, you need to move a few more rules from `basic.css` to `sophisto.css`. Eventually your site will look the way it should in compliant browsers and will still be reasonably attractive in the old, prestandards relics.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Ten. CSS in Action: A Hybrid Layout (Part II)

In [Chapter 8](#), "XHTML by Example: A Hybrid Layout (Part I)," we created hybrid markup for the i3Forum site, combining structural elements like `<h1>`, `<h2>`, and `<p>` with nonstructural components (XHTML tables used to lay out the basic grid), and we used table summaries, accesskey, and Skip Navigation to make the site more accessible in nontraditional browsing environments.

In this chapter, we'll complete our production task by using CSS to achieve design effects that support the brand and make the site more attractive without relying on GIF text, JavaScript rollovers, spacer pixel GIF images, deeply nested table cell constructions, or other staples of old-school web design.

[Figure 10.1](#) shows the home page template as it appears after our first pass at writing a style sheet. As with all design, using CSS is an iterative process. In this chapter, we'll complete our CSS in two passes. The first pass handles 90% of what's needed; the second fixes errors and adds finishing touches.

10.1. The template as it appears after our first pass at CSS. Elements, sizes, fonts, and colors are in place, but backgrounds don't quite fill the right-side menu "buttons." A bit more work is needed.



[\[Team LiB \]](#)

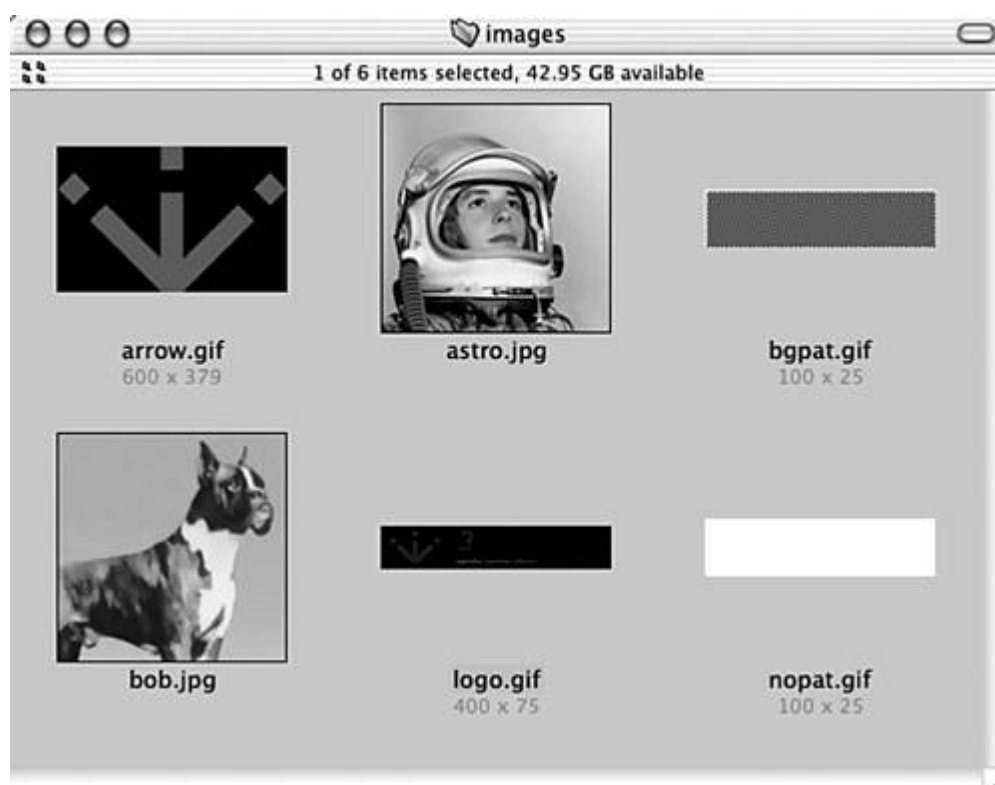
◀ PREVIOUS

NEXT ▶

Preparing Images

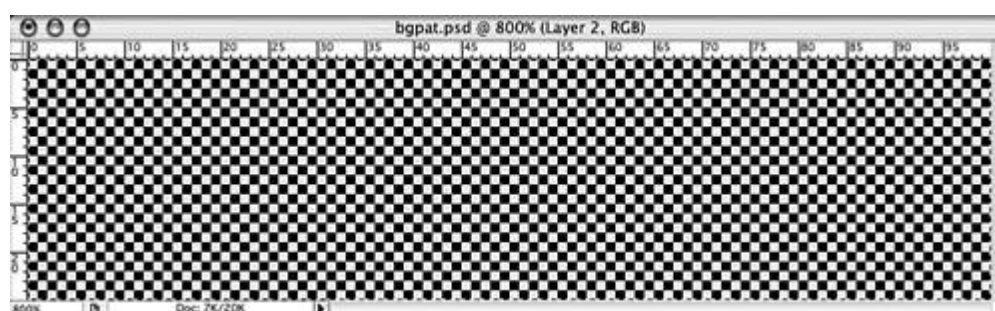
Although the site was designed in Photoshop, it's not your typical slice and dice job. [Figure 10.2](#) shows the six images used to create the entire site. Three are for foreground use: the astronaut photo, the "best of breed" dog photo, and the transparent logo GIF used at the top-left corner of the menu bar.

10.2. Just six images were used in this site, three of them as backgrounds. Photoshop's/ImageReady's "slice and dice" features are not needed.



The remaining three images are backgrounds. Arrow.gif is a screened image derived from the logo that will be placed in the background as a watermark. Bgpat.gif [[10.3](#)], consisting of single-colored pixels alternating with single transparent pixels, will be used to create background color effects in the menu bar. Nopat.gif, a plain-white background, will replace bgpat.gif in CSS rollover effects and might also be used to indicate the visitor's position within the site's hierarchy via an embedded style added to each page at the end of the project (see the following sidebar, "[The Needless Image](#)").

10.3. The alternating pixel background GIF image enlarged 800% and with a black background, inserted for this book's purposes, standing in for the transparent pixels.



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Establishing Basic Parameters

Our images are in place, and with our original comp to guide us, we can begin using CSS to establish basic design parameters. We know that the site will be white with a black background, that its text will be set in several sizes of Georgia (or an alternative serif), that a screened watermark based on the logo will hug the bottom of the content area, and that certain whitespace values must be enforced without using spacer GIF images or additional table junk. Let's make it happen.

Overall Styles, More About Shorthand and Margins

In our first rule, we establish basic page colors and upper and lower margins:

```
body {
  color: #000;
  background: #fff;
  margin: 25px 0;
  padding: 0;
}
```

Per this rule, all text will be black (#000) on a white (#fff) background. The colors are described in [CSS](#) shorthand, as explained in [Chapter 9](#), "CSS Basics." (#000 is shorthand for #000000; #fff is the same as #ffffff.) Shorthand can only be used to replace paired characters; #fc0 is the same as #ffcc00. Shorthand cannot be used in the absence of paired characters. There is no shorthand for a nonwebsafe color like #f93C7a, for example.

Shorthand and Clock Faces

The first rule also establishes a top and bottom margin of 25px and left and right margins of 0. It is a shorthand version of this:

```
margin: 25px 0 25px 0;
```

The preceding, in turn, is a shorthand version of this:

```
margin-top: 25px;
margin-right: 0;
margin-bottom: 25px;
margin-left: 0;
```

In CSS, values are assigned in the order of the main numbers on a clock: 12 o'clock (the top margin), 3 o'clock (the right margin), 6 o'clock (the bottom margin), and 9 o'clock (the left margin). If we wanted our page to have a top margin of 25px, a right margin of 5px, a bottom margin of 10px, and a left margin of 30% of the entire width, our rule would read like this:

```
margin: 25px 5px 10px 30%;
```

When the vertical margins are the same at the top and bottom (as they are in this site—namely, 25px) and when the horizontal margins are the same at left and right (as they are in this site—namely 0), we can save a few bytes of user bandwidth by typing this:

```
margin: 25px 0;
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Navigation Elements: First Pass

Up to now, we've been doing all right. Every rule we've written displays as expected in the standards-friendly, best-case-scenario browser we use to test our work. Getting the navigation bar just right will be trickier. In our first pass, shown next, we nail certain desired features and just miss on others:

```
/* Navigation bar components */
table#nav {
    border-bottom: 1px solid #000;
    border-left: 1px solid #000;
}
```

This rule tells the table whose id is nav to create a 1px solid black border effect at its bottom and left (but not at the top or right).

```
table#nav td {
    font: 11px verdana, arial, sans-serif;
    text-align: center;
    vertical-align: middle;
    border-right: 1px solid #000;
    border-top: 1px solid #000;
}
```

You don't need us to explain that this rule specifies 11px Verdana as the preferred menu text font and fills out the border elements that the previous rule neglected (namely, at the right and top). If we had told the table to create a border effect around all four sides, then the table cells would have added an extra pixel of border at the top and right, bringing shame to our family and sadness to all viewers.

The preceding rule also tells text to be horizontally centered in each table cell and vertically aligned in the middle of the table cell, much like the old-school `td valign="middle"` presentational hack we all know and love. (This seemed like the right thing to do, but in our second pass, we had to remove it.)

```
table#nav td a {
    font-weight: normal;
    text-decoration: none;
    display: block;
    margin: 0;
    padding: 0;
}
```

In the preceding rule, you recognize that we're telling links how to behave, and you also recognize that we're doing so with a sophisticated chain of contextual selectors. The multipart selector means "apply the following rule only to links within table cells, and only if they are found in the table whose unique identifier is nav."

As we hinted in the "[The Image Rule](#)" discussion, we're also using the CSS `display: block` declaration to turn the humble XHTML links into block-level elements that completely fill their table cells. (At least, we hope they will completely fill their table cells.)

```
#nav td a:link, #nav td a:visited {
    background: transparent url(images/bgpat.gif) repeat;
    display: block;
    margin: 0;
}
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Navigation Bar CSS: First Try at Second Pass

In a first try at a second pass, we specify sizes on our link effects:

```
#nav td a:link, #nav td a:visited {
  background: transparent url(images/bgpap.gif) repeat;
  display: block;
  margin: 0;
  width: 100px;
  height: 25px;
}
```

As expected, this causes the right-side buttons to be filled in completely, but it louses up our logo, whose background is also now a mere 100 x 25 pixels [10.13]. 100 x 25 is the right value for the little buttons, but it's wrong for the logo (which is 400 x 75).

10.13. One step forward, two steps back: Specifying sizes on nav pseudo-classes fills in their backgrounds completely but louses up the logo's background. It also obliterates the vertical alignment we established earlier. Text now hugs the top of each cell.



Somehow, the changes we've made also kill the vertical alignment we established earlier. Elements are vertically aligned within their cells, but their content is not. As Fig. 10.13 makes plain, "button" text now hugs the top of each table cell instead of being vertically aligned in the middle. This top-hugging presentation is the same in all CSS-compliant browsers tested. It is not a bug, but unexpected behavior that falls out of the CSS layout model.

Fortunately, when creating the markup way back in Chapter 8, we gave each cell of the table a unique identifier. Home is the id for the cell that contains our logo. Can we use home to create an additional set of rules that override the rules used to fill in the 100 x 25 buttons? You bet we can:

```
td#home a:link, td#home a:visited {
  background: transparent url(images/bgpap.gif) repeat;
  width: 400px;
  height: 75px;
}
td#home a:hover {
  background: white url(images/nopat.gif) repeat;
  width: 400px;
  height: 75px;
}
```

These new rules fill in the logo just right, and the site is nearly perfect. But the loss of the behavior we expected with vertical-align: middle is still unacceptable. We'll fix it in the final pass.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Navigation Bar CSS: Final Pass

In the final pass, we get everything we wanted:

```
/* Navigation bar components */
table#nav {
  border-bottom: 1px solid #000;
  border-left: 1px solid #000;
}
table#nav td {
  font: 11px verdana, arial, sans-serif;
  text-align: center;
  border-right: 1px solid #000;
  border-top: 1px solid #000;
}
table#nav td a {
  font-weight: normal;
  text-decoration: none;
  display: block;
  margin: 0;
  padding: 0;
}
#nav td a:link, #nav td a:visited {
  background: transparent url(/images/bgpat.gif) repeat;
  display: block;
  margin: 0;
  width: 100px;
  line-height: 25px;
}
#nav td a:hover {
  color: #f60;
  background: white url(/images/nopat.gif) repeat;
}
td#home a:link img, td#home a:visited img {
  color: #c30;
  background: transparent url(/images/bgpat.gif) repeat;
  width: 400px;
  height: 75px;
}
td#home a:hover img {
  color: #f60;
  background: white url(/images/nopat.gif) repeat;
  width: 400px;
  height: 75px;
}
```

What changed? We removed the vertical-align: middle instruction altogether. Then we deleted the line that said buttons were 25px tall and replaced it with this:

```
line-height: 25px;
```

Line-height filled in the 25px just as height had done, but it also correctly positioned the text in the vertical middle of each button. It would take a CSS genius to explain why this method worked better than the other. The main thing is, it worked.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Final Steps: External Styles and the "You Are Here" Effect

To wrap the site and ship it to the client, two more steps are needed. First, we must move our embedded styles to an external CSS file and delete the embedded style sheet, as explained in [Chapter 9](#). Then we must create a "you are here" effect [[10.14](#), [10.15](#)] to help the visitor maintain awareness of which page she's on. Remember: We're not changing the markup. We want to create this effect using CSS, without applying additional classes to our navigation bar.

10.14. The "you are here" effect on the Events page template.



10.15. The "you are here" effect on the About page template.



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Eleven. Working with Browsers Part I: DOCTYPE Switching and Standards Mode

How do today's browsers "know" when you've created a forward-compatible site? How can they display your standards-based site correctly when they must also do a decent job of supporting antiquated sites built with outdated methods?

The answer is that most modern browsers use our old friend the DOCTYPE, whose acquaintance we made in [Chapter 6](#), "XHTML: Restructuring the Web to toggle between Standards mode (where your site works as W3C specs say it should) and backward-compatible Quirks mode (so named because old-school sites are authored to the quirks of variously incompatible browsers). If you want to control how your site looks and behaves, this chapter is for you.

To keep things interesting, Standards mode in Gecko browsers like Mozilla and Netscape works slightly differently from Standards mode in Internet Explorer, and these slight differences can have a profound and unexpected influence on your layout. To smooth over potential problems (and to make things even more interesting), later Gecko browsers such as Mozilla 1.01+ and Netscape 7 add a third mode that works more like IE's Standards mode. (Making life still more exciting, Opera 7 has implemented DOCTYPE switching as well, although how it works is anyone's guess as this book goes to press.)

Gecko browsers call this third, IE-like mode "Almost Standards." The name is not meant to insult IE's Standards mode—at least, we don't think it's meant to do that—but to indicate that in the view of Gecko's standards experts, the visual behavior most of us expect from our browsers differs from the letter of the law laid down by CSS specifications.

This chapter explains how the various modes work and provides a simple method of ensuring that your site looks the way you want it to despite differences in the way good browsers interpret CSS and other specs. If you've converted a transitional (tables plus CSS) site to XHTML, discovered that after doing so some browsers treat your layout differently, and wondered why the browsers changed your layout and what you can do to change it back, this chapter has your name all over it.

"The Saga of DOCTYPE Switching" that follows explains why most browsers needed a toggle to alternate between Standards and Quirks modes, and how they came to choose the humble DOCTYPE as the switch. If you're the kind of person who likes to know where things come from and why they are as they are, read on. If you prefer to bypass the background data and get right to the tips and techniques, kindly skip the next couple of pages and proceed directly to "[Controlling Browser Performance: The DOCTYPE Switch](#)."

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Saga of DOCTYPE Switching

In the late 1990s, as leading browser makers recognized that complete and accurate support for web standards was important to their customers who designed and built websites, they asked themselves how they could fashion new browsers that supported standards correctly without ruining old, noncompliant sites in the process.

After all, existing versions of Netscape and Explorer had persuaded legions of designers to learn their particular quirks, including proprietary extensions to HTML, partial and incorrect CSS implementations, and manufacturer-specific scripting languages. Microsoft and Netscape were willing to do a much better job of supporting standards, but not if it meant breaking billions of dollars worth of existing sites. For a browser maker, that would be suicide.

An example might help explain the browser makers' dilemma.

In the mid-1990s, when early versions of Internet Explorer began to partially support CSS1, they naturally got a few things wrong, such as the box model (explained in [Chapter 12](#), "Working with Browsers Part II: Box Models, Bugs, and Workarounds"). First drafts are always rough, and Microsoft is to be commended for beginning to support CSS at the dawn of 1997.

But commendable or not, Microsoft's early misinterpretation of the CSS box model posed a problem. Tens of thousands of designers had already "learned" the incorrect box model used by IE4.x and IE5.x and had tailored their CSS to display appropriately in those versions of IE. If later versions of IE, let alone other manufacturers' browsers, supported the box model more accurately, existing designs would surely fall apart, to the displeasure of client, builder, and user alike. What to do?

A Switch to Turn Standards On or Off

Even before Netscape and Microsoft agreed to build browsers that supported standards more accurately and completely, an unsung hero had solved the problem of gently handling sites built with noncompliant methods. That hero was user interface technologist Todd Fahrner, a contributor to the W3C's CSS and HTML working groups and cofounder of The Web Standards project. You will find Fahrner's name scattered about the pages of this book, and you will also find it in this book's index if the indexers have done their job.

In early 1998, on an obscure W3C mailing list (all W3C mailing lists are fairly obscure, of course, but the adjective lends a certain mystique to our story), Fahrner proposed that browser makers incorporate a switching mechanism capable of toggling standards-compliant rendering on or off. He suggested that the presence or absence of a DOCTYPE be used as the on/off switch.

If a web page's markup began with a DOCTYPE, Fahrner reasoned, the odds were high that its author knew about and had made an attempt to comply with standards. Browsers should parse such a web page according to W3C specs. By contrast, a DOCTYPE-free page could not pass W3C validation tests (<http://validator.w3.org/>) and would most certainly have been built using old-school methods. Browsers should treat it accordingly; that is, they should display that web page the way older, noncompliant browsers handled such pages.

One problem remained: There were no standards-compatible browsers. The world would have to wait two years to see if DOCTYPE switching held the key to forward and backward compatibility. (But you don't even have to wait

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Controlling Browser Performance: The DOCTYPE Switch

As explained earlier, most modern browsers use the presence or absence of certain DOCTYPEs to toggle between rigorous standards compliance and fault-tolerant, backward-compatible display. Browser implementations and details differ, but the gist of DOCTYPE switching follows the outline Todd Fahrner sketched back in 1998, when standards-compliant browsers were barely a gleam in developers' eyes. Here's a simplified overview of how it works:

- An XHTML DOCTYPE that includes a full URI (a complete web address) tells IE and Gecko browsers to render your page in Standards mode, treating your CSS and XHTML per W3C specs. Some complete HTML 4 DOCTYPEs also trigger Standards mode, as discussed a few pages from now. In Standards mode, the browser assumes that you know what you're doing.
- Using an incomplete or outdated DOCTYPE—or no DOCTYPE at all—throws these same browsers into Quirks mode, where they assume (probably correctly) that you've written old-fashioned, invalid markup and browser-specific, nonstandard code. In this setting, browsers attempt to parse your page in backward-compatible fashion, rendering your CSS as it might have looked in IE4/5 and reverting to a proprietary, browser-specific DOM.

To control which tack the browser takes, simply include or omit a complete DOCTYPE. It's that easy. Almost.

Three Modes for Sister Sara

For reasons we'll explain later in this chapter, newer Gecko browsers toggle between three modes: Quirks (as described earlier), Almost Standards (equivalent to Standards mode in IE), and Standards (slightly different from IE's Standards mode in one or two key particulars). Also, some DOCTYPEs that trigger Standards mode in early Gecko browsers instead trigger Almost Standards mode in later Gecko browsers.

To the uninitiated, this might appear deeply and painfully confusing, but a few pages from now, you will find it cruelly and brutally confusing. Nonetheless, there are legitimate reasons for it, and, just as importantly, there are also good, easy workarounds that help ensure correct display not only in browsers that use various methods of DOCTYPE switching, but also in browsers like Opera 5/6 that don't use DOCTYPE switching at all. Hang in there and read on. Together we will get through this.

Before we probe the fine points of browser differences, let's study what browsers have in common.

Complete and Incomplete DOCTYPEs

Browsers that use DOCTYPE switching look for complete DOCTYPEs. That is to say, they look for DOCTYPEs that include a complete web address such as <http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd>. The following DOCTYPE triggers Standards mode in any modern browser that employs DOCTYPE switching:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Yes, this is a Strict DOCTYPE, and yes, we've been promoting the use of XHTML 1.0 Transitional so far. We use

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Celebrate Browser Diversity! (Or at Least Learn to Live with It)

Gecko's Standards mode differs from IE's in a few details, and this difference can be disagreeable if you're not expecting it. The difference is especially visible and can be especially disagreeable in hybrid (CSS plus table) layouts when viewed in first-generation Gecko browsers.

After converting from HTML 4.01 Transitional to XHTML 1.0 Transitional, making no changes to their markup except those described in "[Converting to XHTML: Simple Rules, Easy Guidelines](#)" ([Chapter 6](#)), designers will expect their layout to look just the way it always has. After all, they're only changing a few markup tags; they're not altering the design. Nevertheless, their layout might look unexpectedly different in early Gecko browsers such as Netscape 6.0 and Mozilla 1.0. If the designers have converted to XHTML Strict, the layout will look different in all Gecko browsers, old and new. There is a standards-based reason for the change—and there is also an easy way to change it back.

The Image Gap Issue in Gecko

[Figure 11.1](#) shows The Daily Report at www.zeldman.com as it looked for several years, when the site was laid out with transitional techniques combining CSS and tables.

11.1. This old hybrid table/CSS layout at www.zeldman.com used a complete DOCTYPE to trigger Standards mode. After converting from HTML to XHTML, it looked the same as it always had in Internet Explorer. (www.zeldman.com/daily/1002a.html).



After converting from HTML 4.01 Transitional to XHTML 1.0 Transitional (and changing from a complete HTML 4.01 DOCTYPE to a complete XHTML 1.0 Transitional DOCTYPE), the site looked the same as it always had in Internet Explorer for Windows and Macintosh. But in Gecko browsers, mysterious, unwanted gaps appeared between images in the table that controlled the site's navigation menu [[11.2](#), [11.3](#)].

11.2. In early Gecko-based browsers, before applying the CSS solution described in this chapter, the site's

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Twelve. Working with Browsers Part II: Box Models, Bugs, and Workarounds

"Create once, publish everywhere" is the Grail of standards-based design and development. We don't learn proper XHTML authoring to win a gold star. We do it so that our sites will work in desktop browsers, text browsers, screen readers, and handheld devices—today, tomorrow, and 10 years from now. Likewise, we don't use CSS exclusively for short-term rewards like reducing bandwidth to save on this month's server costs. We do it primarily to ensure that our sites will look the same in Netscape 14.0 as they do today and that unneeded presentational markup won't impede user experience in non-CSS environments.

Standards-compliant user agents move forward compatibility from the realm of wishful thinking to the forefront of rational, sustainable design strategies. If the web were still viewed mainly in Netscape and Microsoft's 4.0 browsers, forward compatibility would be unattainable by all but the most rudimentary sites. If the leading user agents that succeeded 4.0 browsers had continued to promote proprietary technologies at the expense of baseline standards, the web's future as an open platform would be open to doubt.

Thankfully, all leading and many niche browsers released within the past few years can justifiably be called "standards-compliant." But some are more compliant than others. Coping with compliance hiccups is what this chapter is all about.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

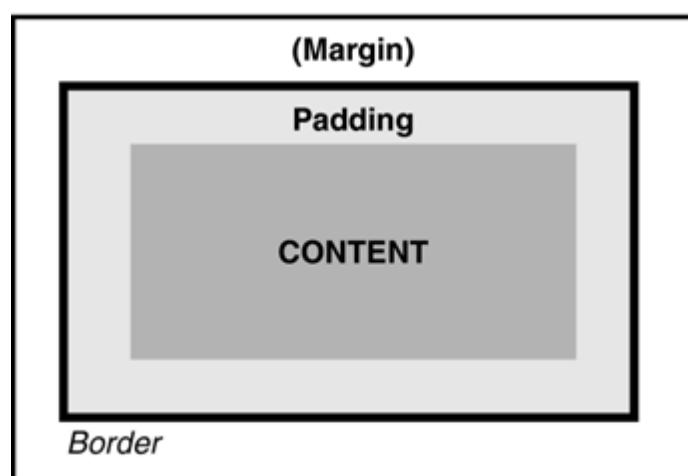
The Box Model and Its Discontents

With the publication of CSS1 in 1996, the W3C proposed that any object on a web page inhabits a box whose properties designers can control by creating rules. It doesn't matter if the object is a paragraph, list, headline, image, or generic block-level element such as `<div id="navigation">`. If it can be inserted into a web page via markup, it lives in a box.

[Figure 12.1](#) illustrates the four areas of the CSS box: content, padding, border, and margin. Content (dark gray in our diagram) is the innermost area, followed by padding (light gray), border (black), and margin (white). These areas should be familiar to you. In [Chapter 10](#), "CSS in Action: A Hybrid Layout (Part II)," while creating a hybrid layout for the i3Forum site, we specified border, padding, and margin values for various page elements. We also assigned size values to the content area when we wrote rules like the one that follows. (Content values are highlighted in bold.)

```
td#home a:link img, td#home a:visited img {
    color: #c30;
    background: transparent url(/images/bgpatt.gif) repeat;
    width: 400px;
    height: 75px;
}
```

12.1. The four areas of the generic CSS box: content, padding, border, and margin. (The outer edge of the margin was artificially darkened for your viewing pleasure.)



In this rule, the width of the content area is 400 pixels and its height is 75 pixels. Notice that we didn't write this:

```
content: 400px;
```

We also didn't write anything like this:

```
content-width: 400px;
content-height: 75px;
```

There are no such rules in CSS. Border, margins, and padding are assigned their values by name; the content area is not. In beginning to learn and use CSS, you might understandably assume that `width: 400px` applies to the entire box (excluding the margin). After all, that is how page layout programs work, it is how designers think, and it is how users understand layouts. If you create a CSS layout containing two divs side by side and assign each a width of 50% of the visitor's browser window, you might expect the two to retain that value as you add padding and borders. But that is not how CSS works.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Whitespace Bug in IE/Windows

Lest we weary you with too much theory, let's turn to a simpler issue with a less complex solution—namely, the whitespace bug in IE/Windows. [Figure 12.9](#) shows the havoc the whitespace bug can wreak in hybrid layouts (compare with [Figure 11.4](#) in [Chapter 11](#), "Working with Browsers Part I: DOCTYPE Switching and Standards Mode"), but it is equally damaging to pure CSS layouts.

12.9. Another day, another display problem. This time, the source is a whitespace bug in IE/Windows. The image has been enlarged to show details. Compare and contrast with [Fig. 11.4](#) ([Chapter 11](#)), which shows the intended display.



Each of the two markup snippets that follow is functionally equivalent. However, because of the snippets' varying use (or nonuse) of whitespace, they will display differently in a browser that wrongly attempts to parse whitespace in XHTML. Thus:

```
<td></td>
```

will display differently from this:

```
<td>
  
</td>
```

The second example—the one with whitespace in its markup—might result in unwanted visual gaps on your web page, as in [Figure 12.9](#). The same is true for the following example. The next markup (with no whitespace)

```
<td><a href="#foo"></a></td>
```

might look different in your browser from the functionally identical

```
<td>
  <a href="#foo">
    
  </a>
</td>
```

Why does this happen? The whitespace bug was a known problem in Netscape Navigator dating back to Version 3.0 (if not earlier). When Microsoft decided to build a competing browser, its engineers emulated much of Netscape's behavior—including, unfortunately, a few of its bugs. As of this writing, IE/Windows browsers right up through IE6 continue to emulate this old Netscape bug. The solution? Remove the whitespace from your markup.

The whitespace bug can be as tough on CSS layouts as it is on hybrid CSS/table layouts. Consider the following list,

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The "Float" Bug in IE6/Windows

Earlier in this chapter we explained how the CSS "float" property allows one element to sit alongside another. For instance, the rule that follows would cause a div with the unique id of "maintext" to float to the left of a sidebar or other right-hand element:

```
#maintext {
  float: left;
}
```

You'd better hope that the text in "maintext" is short and to the point. IE6/Windows has an unfortunate bug whereby long passages of text contained in a floating div are artificially truncated, preventing the reader from seeing the complete text while also causing the scrollbar to disappear. To see the complete text or restore the browser's scrollbar, the reader must (get this) refresh the page or press F11 twice in quick succession. Do your site's visitors routinely press F11 twice in succession upon loading any new web page? Neither do ours.

The problem affects an unknown (and hopefully small) percentage of IE6 users, but that's like saying it affects only a small percentage of the Chinese. The bug was first spotted and reported in 2001, while IE6/Windows was still in beta, and Microsoft's engineers have tried to fix it. However, as this book goes to press in 2003, the bug remains.

Microsoft's engineers might be unable to reproduce the bug in their labs or track down its cause, but the independent design community seems to have discovered the source of the problem, as well as a workaround. Read on.

Stuck on Old Values

Apparently, IE6 has trouble calculating the heights of block-level elements. Eddie Traversa of <http://dhtmlnirvana.com/> found that IE6/Windows caches the values it calculates on one page of a site and then incorrectly applies those values to other pages. For instance, if the "maintext" div happened to be 300px tall on the first page of your site and 1400px tall on the following page, IE6 might display only the first 300px. Manually reloading each page "fixes" this bug by clearing the cache of the stuck initial value—until the reader clicks through to a new page and begins a new cycle of stuck values and unreadable text.

Fixed with a Script

The first step is acknowledging that you have a problem. The second is diagnosing that problem's exact nature. In the third step, programmer Aaron Boodman of Youngpup.net wrote a tiny JavaScript function that, in the presence of IE/Windows, iterates an existing property of the floated block, causing the page to reflow and display correctly. Here is Aaron's entire script:

```
if (document.all && window.attachEvent) window.attachEvent("onload", fixWinIE);
function fixWinIE() {
  if (document.body.scrollHeight < document.all.content.offsetHeight) {
    document.all.content.style.display = 'block';
  }
}
```

If you prefer, feel free to copy and paste these few lines of JavaScript from the beginning of <http://www.alistapart.com/styleswitcher.js>. These lines might allow you, too, to create CSS layouts that use float fearlessly. In your own version, replace references to .content with the name of your floated div. For instance, if your

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Flash and QuickTime: Objects of Desire?

Many of us embed multimedia objects such as Flash and QuickTime movies in our sites. There is no standards-compliant way of doing so that works reliably across multiple browsers and platforms. To understand how this can be, we must tell a tale of hubris and vengeance as floridly melodramatic as anything in Shakespeare or Italian opera. Well, okay, not quite that melodramatic, but close.

Embeddable Objects: A Tale of Hubris and Revenge

When the creators of the original Mosaic and Netscape browsers first seized on the brilliant idea of allowing designers to include images in web pages, they "extended" HTML by creating an `` tag specifically for their browsers. The W3C did not approve. It advised web authors to use the object element instead. But millions of websites later, the `` tag was still going strong—and support for the W3C's `<object>` element was nonexistent.

Then came the FutureSplash plug-in (later rechristened Flash) along with other multimedia elements such as Real and QuickTime movies. Again, the W3C suggested that the `<object>` tag be used to embed such content in web pages. But Netscape invented the `<embed>` tag instead—and as competitive browsers came onto the scene, they too supported Netscape's `<embed>` tag.

In the view of Netscape and Microsoft, their customers expected the web to function as a rich multimedia space, and it was up to browser makers to fulfill that desire through innovation—not coincidentally gaining market share in the process.

In the view of the W3C, browser makers were creating their own tags and ignoring perfectly good (standard) specifications. And what was the point of creating useful, open specifications if W3C member companies paid them no heed? In the years to come, the W3C would wreak a bloody double vengeance on those who had ignored its beautiful standards.

(Okay, okay, they wouldn't wreak a bloody double vengeance or anything of the kind. They simply did what their charter told them to do and what they felt certain was right. Namely, they established markup standards that made sense. But it's a lot more fun to tell it the way we're telling it. Indulge us.)

The Double Vengeance of W3C

The W3C's first act of revenge was to avoid including the `<embed>` tag in any official HTML specification, in spite of the fact that hundreds of thousands of designers were using it on millions of sites. `<embed>` was not included in HTML 3.2. It was not added to HTML 4 or to HTML 4.01, whose sole purpose was to include commonly used tags that had been left out of HTML 4. And because XHTML 1 was based on HTML 4.01, `<embed>` was not part of XHTML, either. Therefore, any site that used `<embed>` could not—and still cannot—validate as HTML or XHTML.

That is correct. You read that right. Millions of sites that embed multimedia cannot validate against a W3C spec because the W3C never deigned to recognize the `<embed>` element.

As if the wound still smarts, and yet more vengeance is desired, the W3C has banished the humble `` element from its initial XHTML 2.0 specification (see "[Which XHTML Is Right for You?](#)" in [Chapter 5](#), "Modern Markup").

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

A Workaday, Workaround World

Workarounds help us begin fulfilling the promise of standards ("create once, publish everywhere") in spite of the fact that no browser is perfect and some are less perfect than others. In so doing, workarounds free us to improve our sites' content, design, and usability, instead of wasting costly hours on proprietary, dead-end technologies.

But not everyone approves of workarounds, however practical or beneficial. In the view of some, if browsers don't fully or correctly support a given W3C spec, it is too bad for that browser's users—or the standard should simply be avoided. The problems with this perspective are many, including these:

- If you limit yourself to specifications that are completely and accurately supported by all browsers, guess what? You can't create a web page. Even HTML 3.2 is not universally and accurately supported in its entirety.
- Getting standards right takes time—and time is tight in competitive markets. Commercial pressures sometimes force engineers to release products before nailing every nuance of a particular specification. Occasionally they must release software knowing it is flawed. (Netscape 6.0 and IE 6.0 both contained bugs of which their engineers were aware, such as the IE6 float bug mentioned earlier in this chapter.) If we are too purity conscious to use workarounds, our site's visitors will suffer needlessly.
- Specifications are occasionally vague in places, and some specs change after being published. Take CSS2—please. As mentioned a few pages ago, CSS2 was revised to CSS2.1 in 2002 because the original version contained a few obscure or nonworkable ideas. When goal posts move, teams fumble. Likewise, in the original CSS1 specification, the scaling factor between adjacent font size keywords was 1.5. Netscape 4, of all browsers, implemented keyword scaling exactly the way W3C said browser makers should. And guess what? Small fonts were too small, and big fonts were too big. The spec changed later.
- Most of us must compensate for the flaws of older user agents, particularly if those flawed agents are still widely used—as IE5.x/Windows, with its incorrectly implemented CSS Box model, is.

Readers who fear that by recommending workarounds, we are leading them into a morally questionable universe should note that all workarounds shared in this chapter are perfectly compliant. We are not writing proprietary code or junk markup to work around browser limitations. We are using standards to support standards. For instance, in the Box Model Hack, we are using CSS2 selectors to ensure that browsers with broken box models get our CSS1 layout requirements right. We are removing needless whitespace from our markup and striving to create text that is legible and accessible to all.

There is nothing wrong with these techniques, and much right with them; they allow us to use standards today instead of turning our eyes to a distant horizon filled with perfect browsing software. As browsers improve, and as old browsers fall into disuse, fewer workarounds will be needed, although we might continue to use them to ensure that our sites are as backward compatible as they are forward looking. In the next chapter, we'll conclude our brief survey of browser joys and sorrows by examining one of the most basic aspects of design—namely, the control of typography.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Thirteen. Working with Browsers Part III: Typography

Along with positioning and color, typography is a basic and essential tool of design. Print designers spend years studying the history and application of type. They learn to distinguish between faces that, to the uninitiated, look almost identical, such as Arial versus Helvetica. When these traditionally educated designers come to the web, with its limited and contradictory typographic toolsets, they have often been less well equipped to navigate its tight and rocky shoals than those from a nontraditional design background.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Size Matters

Windows, UNIX, and Macintosh come with different installed fonts, at different default resolutions, and often with different default rendering styles—from pixellated to gently antialiased (as in Mac OS 9) to type so smooth that it borders on Photoshop text (as in Mac OS X Jaguar by default, and Windows XP with ClearType—but only if the user turns on the ClearType option). The old `` thus means something different among operating systems, not only in size but also in appearance. Sadly, most CSS font-sizing methods also result in different sizes from one operating system to the next, despite efforts by leading browser makers to reduce or eliminate cross-browser problems by establishing a standard default font size.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

User Control

In addition to differences between platforms, outdated methods, and CSS renderings, the web differs from print in that the user is supposed to retain some control over what she sees. Accommodating the user is as tricky with standards as it is with old-school methods, due to problems discussed in this chapter. (And oh, boy, will we discuss these problems in this chapter.) It is also difficult for traditionally trained designers to accept the premise of user control. Sadder still, CSS methods (ems, percentages, font size keywords) that are intended to deliver user control suffer from cross-browser and cross-platform problems. For one brief shining moment, these problems were resolved by the good will of browser makers, as they agreed to uniformly support a cross-platform standard size. But new browsers are unwittingly undoing that good work, making it tougher to balance design requirements with the user's need for control.

In this chapter, we will discuss the history and problems of web typography and study two methods of setting text via web standards. Both methods work well but neither one is perfect. We will also look at methods that are even less perfect because they're more fraught with problems. Thirteen is an unlucky number; designers who care about branding and design but who also want to do right by all users face tough and occasionally unfortunate choices, as this chapter will explain. We'll cover the techniques, their benefits, and their occasionally unhappy risks and tradeoffs. As always, you will decide which tradeoffs work best for your audience.

[\[Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[Team LiB \]](#)

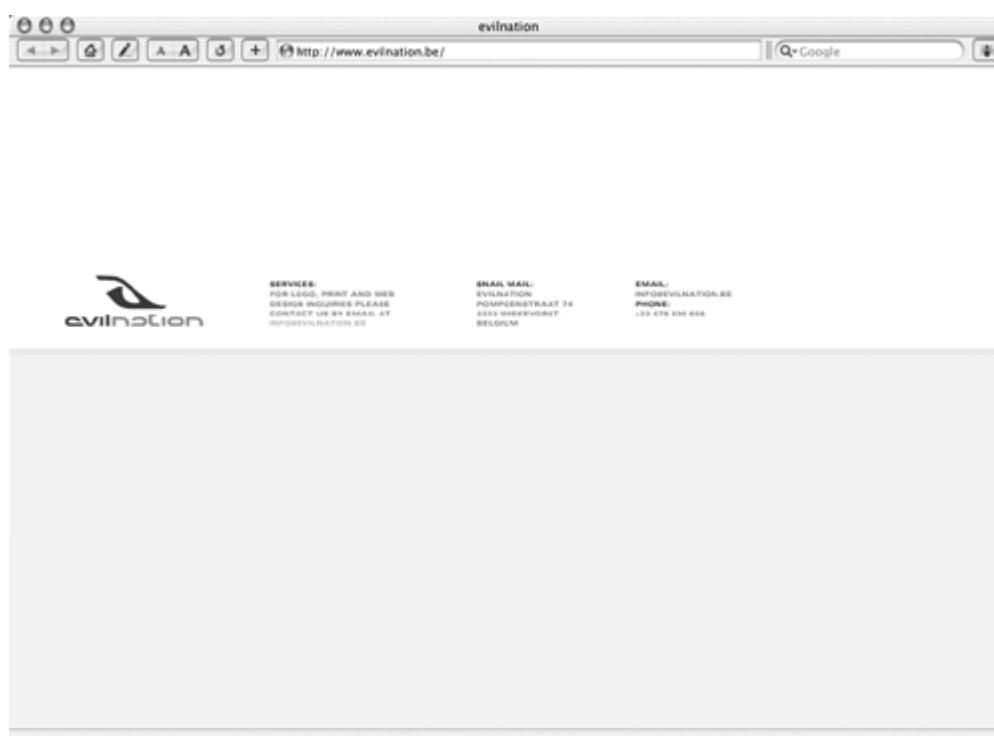
◀ PREVIOUS

NEXT ▶

Old-School Horrors

Tim Berners-Lee, the inventor of the web and the founder of the W3C, viewed his creation as a medium for the easy exchange of text documents; therefore, he included no typographic control in the structured language of HTML. As described in the section "[Jumping Through Hoops](#)" in [Chapter 2](#), "[Designing & Building with Standards](#)," web designers initially used `<tt>`, `<pre>`, `<blockquote>`, and semantically meaningless paragraph tags to vary typefaces, achieve positioning effects, and simulate leading. They next began using GIF or Flash images of text, a practice that continues to this day [[13.1](#)], often at the hands of skilled designers who have difficulty accepting the limitations of CSS and XHTML and the tradeoffs that are inherent in balancing user versus designer needs.

13.1. Every "word" on this page (<http://www.evilnation.be/>) is a Flash vector, not real text, hence not searchable, accessible, or amenable to copying and pasting. Skilled designers who care about typography often have difficulty accepting the limitations of XHTML and CSS and the tradeoffs of user control.



By 1995, with commercial sites springing up right and left and designers hacking HTML to ribbons, something had to be done to provide at least a few basic typographic tools. Netscape gave us the `` tag whose attribute was size. You could specify numbers (``) or relative numbers based on the user's default (``). Designers quickly abandoned paragraph tags and other structural elements and controlled their layouts by combining `` with `
` tags. Not to be outdone, Microsoft gave us ``.

Most readers of this book will remember those days and will also recall the problems—chiefly, those of platform difference. Windows assumed a default base size of 16px at 96ppi (pixels per inch). Macintosh, closely tied to print design, assumed a size of 12px at 72ppi based on the PostScript standard. Font sizes that looked dandy on one platform looked too big or too small on the other.

"Stupid Windows," said the Mac-based designers.

"Stupid Macintosh," said the Windows-based designers.

Points of Difference

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

A Standard Size at Last—But for How Long?

In an effort to transcend platform differences, the makers of Netscape and Microsoft's browsers and Mozilla put their heads together in late 1999 and decided to standardize on a default font size setting of 16px/96ppi across platforms. By putting all platforms on the same page, as it were, browser makers and users could avoid the problems of cross-platform size differences and cruelly useless, illegible text.

The 16px/96ppi Font Size Standard

On a W3C mailing list in 1998, the plucky Todd Fahrner puckishly proposed standardizing on a 16px default per Windows usage. His recommendation was adopted by all leading browser makers in the year 2000. Although Fahrner's concerns were practical in nature (he wanted to ensure that type could be read online), in the quotation that follows, he refers to something that might strike you as bizarre.

The framers of CSS were bound to a pet abstraction wherein the "average" length of a web user's arm was essential to defining the size of a pixel. No, really. Anyway, in advancing his idea about a standardized cross-platform font size, Fahrner was careful to cover the all-important arm length issue along with more pedestrian matters like usability. He wrote:

Since before Mosaic, the default font size value in all major browsers has been set at 12pt. I propose redefining the default as 16px . The current default of 12pt rasterizes very differently across platforms. On Macs, it rasterizes into 12px (logical res fixed at 72ppi). On Wintel PCs, it rasterizes by default into 16px (logical res defaults to 96ppi). All scalable font-size values operate relative to this inconsistent base rasterization. For a designer, this means that the only way to suggest a [consistent cross-platform] font size is to use CSS pixel units, which are not user-scalable, and are thus not optimally user-friendly/portable

The appropriate corrective measure is for Mac (and X11?) browsers to break with tradition and ship with the default value of "medium" text set at 16px, instead of 12pt. This should of course remain subject to user adjustment, but a consistent initial value will at least make the use of scalable font-size values less problematic for designers, as any variance from the default will be due to express user preference rather than capricious legacy OS differences.

If designers tend to believe that 16px is too large as a base, why suggest it as the default?

One reason is pure expediency. The Mac is a smallish minority platform, though very strongly represented in the web design field. (I use a Mac!) It is unrealistic to expect that Windows/X11 browsers will change their defaults to match the Mac's rather quaint limitation to 72ppi logical resolution.

The 1996 CSS1 standard suggests a 1/90" value for a "reference pixel," extrapolated from a visual angle of 0.0227 degrees visual angle at arm's length. [User agents] are expected to scale pixels appropriately if the physical resolution is known to vary significantly from this value. A 1/90" reference pixel would suggest a rasterization of 12pt into 15px, rather than 16. 15 is, of course, much closer to 16 than to 12, however. Because no OS/UA currently assumes a 90ppi logical resolution (nor implements

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Heartbreak of Ems

Accessibility advocates and the creators of CSS have long agreed that ems are the way to go. Sadly, they are often the way to go to hell. Listen to all the lectures, read all the books and articles, and you will come away feeling dirty and ashamed if you use anything other than ems to specify your type sizes. But the beautiful theory of ems breaks down in coarse practice—and not only in browsers that fail to support the common default font size.

On a minor note, there is the problem of old browsers. Netscape 4 ignores em and ex units that are applied to text, although it bizarrely respects these units when they are used for line height. IE3 treats em units as pixels. Thus, 2em is mistranslated as 2 pixels tall. Almost no one uses IE3 any more, but still.

Likewise, older browsers often bungle inheritance on nested elements sized with em units. Because fewer and fewer people are stuck with Netscape 4, we won't waste your time going into the details of that browser's mishandling of relatively sized nested elements. Just know that if you need to support outdated browsers and if you use em units (especially on nested elements), you are letting yourself and your users in for a world of pain [13.11].

13.11. What's in an em? Not cross-platform, cross-browser size consistency, that's for sure (http://www.thenoodleincident.com/tutorials/box_lesson/font/).

Text as 0.8em.	Text as 0.8em.	Text as 0.8em.
Text as 0.8em with base 100%	Text as 0.8em with base 100%	Text as 0.8em with base 100%
Text as keyword small.	Text as keyword small.	Text as keyword small.
Text as keyword small, ALA style.	Text as keyword small, ALA style.	Text as keyword small, ALA style.

User Choices and Em Units

A more common problem with em units is that users often downsize their default font size settings as noted several times in this chapter. Mac users switch back to 12px/72ppi; Windows folks set their browsers' View: Text Size menu to "small" rather than "medium." Such changes make any text sized below 1em smaller than it is supposed to be and might make it too small to be read. In 2002, CSS/DHTML expert Owen Briggs tested every available text sizing method across a vast range of browsers and platforms to find out what worked and what failed. 264 screen shots later, despite hoping to prove that ems were always viable, he had actually discovered the opposite [13.11].

Ems work well as long as you never spec your text below the user's default size. Ems work well as long as users never adjust their preferences. But most designers and many clients favor smaller type and many designs require them. Many users consider the 16px default size uncomfortable for normal reading and change their preference settings accordingly. When em units are used to design sites, the designer's and user's shrinkage efforts compound on one another, resulting in text that might be hard to read or even entirely illegible.

When you set small type with em units (or percentages), you run afoul of a universe of unknowable, uncontrollable user preference settings. What looks elegant on your monitor might be mouse type on your users'. If you commit this act in the name of accessibility, you're kidding yourself.

In the i3Forum site, we tried to minimize the potential damage by sticking to sizes that were only slightly smaller than 1em. But the user's mileage might vary.

Alternatively, (<http://www.elistenport.com/stories/doc/>), client and aesthetics permitting, you can design all your sites

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Pixels Prove Pixels Work

Alone among all CSS values, with a few exceptions to be discussed next, the humble and much-maligned pixel (px) unit works in browsers old and new, compliant and noncompliant, on any platform you care to name. 13px is 13px whether viewed in Windows, Mac OS, or Linux/UNIX. Set your text in pixels, and it will look the same in Gecko/Mac OS X, [13.12], IE6/Windows [13.13], and Opera 7/Windows [13.14].

13.12. The beginnings of a February 2003 redesign of zeldman.com (<http://www.zeldman.com/>). Text is set in pixels; therefore, it is the same size in Chimera for Mac OS X (shown here)



13.13. as it is in IE6 for Windows XP. Not only that



13.14. it even looks the same in Opera 7/Windows, despite some theorists' claims that Opera always abstracts pixel values

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Font Size Keyword Method

Little known and scarcely ever used, CSS1 (and later, CSS2) offered seven font size keywords intended to control text sizes without the absolutism of pixels or the inheritance, cross-platform, and user option hazards of ems and percentages. The seven keywords appear next, and their meaning will be obvious to anyone who has ever bought a tee shirt (<http://www.w3.org/TR/CSS2/fonts.html#value-def-absolute-size>):

xx-small

x-small

small

medium

large

x-large

xx-large

Why Keywords Beat Ems and Percentages

As mentioned earlier in "The Heartbreak of Ems," when you use ems or percentages, there is always the danger that their values will multiply, resulting in text that is too small or too large. By contrast, keyword values do not compound even when the elements nest. If <body> is small, <div> is small, and <p> is small, and p lives inside div, which lives inside body, the three small values do not compound (as ems and percentages do), and the result is still legible. Moreover, the result is still small (not x-small or xx-small). Ems and percentages compound. Keyword values do not compound.

In addition, at least in Gecko and modern IE browsers, xx-small can never be smaller than 9px, which means it can never be illegible. Text might be hard for some users to read, of course, but that is not the same thing as illegibility.

Like ems, keywords are based on the user's default font size. Unlike ems, keywords never descend below the threshold of adequate resolution. If the user's default size happened to be 10px (unlikely, but possible), x-small would be 9px and xx-small would also be 9px. Obviously, in such a case, you would lose the size difference between x-small and xx-small. But you wouldn't lose your readers.

Sounds perfect, doesn't it? We get to specify sizes without smacking up against IE/Windows' inability to resize pixels, and also without inflicting illegibly teeny type on any visitor. Font size keywords seem to balance accessibility with the designer's need for control. So what could go wrong? One word: browsers.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Fourteen. Accessibility Basics

Accessibility and standards have much in common. They are both about ensuring that our work will be useable and available to the largest possible number of readers, visitors, and customers. Accessibility is so closely linked to the other standards discussed in this book that in the 1990s, the W3C launched a Web Accessibility Initiative (WAI) to advise web builders on strategies for achieving it (<http://www.w3.org/WAI/GL/>).

WAI offers three standardized levels of access, from the readily achieved (Priority 1), to one that requires slightly more work (Priority 2), to a master level (Priority 3). The point of the three levels is that accessibility, like the other forms of standards compliance discussed in this book, is a continuum rather than an "all or nothing" affair. Even if we are unready to convert to CSS layout, we can forward-proof our sites and comply with standards using the hybrid methods shown in [Chapters 8](#), "XHTML by Example: A Hybrid Layout (Part I)," [9](#), "CSS Basics," and [10](#), "CSS in Action: A Hybrid Layout (Part II)." So too, with a small and reasonable effort, any of us—even those who are new to accessibility—can attain Priority 1 conformance or something close to it. In so doing, we'll begin making our sites available to those whom we had previously locked out.

Many nations have laws proscribing denial of access to the disabled, and many nations have applied those laws to new media via web accessibility edicts like U.S. Section 508. Some of these national laws adhere closely to WAI Priority 1. Others pick and choose randomly from the access buffet. Some nations' laws would confuse even the members of WAI, who have spent years studying the problems of access. Some laws are vague, and some straightforwardly practical. Many laws go ignored by the very bodies that created them. No wonder designers and developers are confused.

In this chapter, we will cut through confusion, dismiss cobweb-coated myths that befuddle many otherwise sophisticated professionals, and provide a practical overview of common sense, applied accessibility. We'll also discuss tools that can help you incorporate accessibility into your design practice and point out the limitations of those tools.

No single chapter could cover the field of accessibility in its entirety; to claim otherwise would insult not only accessibility experts but more importantly those whose needs they serve. Indeed, few books really get at the heart of the thing we will cover all too briefly in this chapter, and some inadvertently contribute to designers' confusion about and hostility toward access. Two books, however, do focus on accessibility in a designer-friendly way, and we commend them to your attention.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Access by the Books

Many well-intended accessibility books preach fire and brimstone. The smell of sulfur does not inspire designers. All too frequently, these books contain only visually ugly—or completely unrealistic—examples of accessible sites, along with impractical advice such as "never specify type sizes." Some in the field are hostile to design. Others have no experience in developing commercial sites. Designers might come away from these books believing that accessibility is irrelevant.

Other books are well researched and fueled by passionate insight. These are worth the devotee's time. But they are not recommended for the general web professional because they are pitched at readers who live with one or more disabilities. In serving that readership, these books spend much time presenting alternate input methods and assessing the merits and demerits of alternative user agents. Nondisabled designers are likely to feel alienated if not unconsciously fearful that somehow they too will be afflicted. Fear of blindness, paralysis, and other disabilities partly fuels some designers' discomfort with the very concept of accessibility, and such books will not help designers shirk that prejudice.

We recommend the following:

-

Building Accessible Web Sites, by Joe Clark (New Riders: 2002)

Not merely the best and most complete book yet penned on the subject of web accessibility, Joe Clark's Building Accessible Web Sites is also among the most compellingly written web design books ever: witty, opinionated, and truthful. In our strange line of work, we see most new design books and many new computer books. Few are complete, fewer still are entirely lucid, and with very few indeed do we feel that we are in the hands of a master communicator. We devoured Clark's book from cover to cover as if it were the latest James Ellroy novel or a recently unearthed Raymond Chandler. Then we read it again. Not only will you learn everything you need to know from this book, but you can actually read it for pleasure.

Building Accessible Web Sites covers it all, from the basics of writing usable alt attributes to the complexities of captioning rich media. Joe Clark, whom the Atlantic Monthly called "the king of closed captions," has spent 20 years in the field of media access, and it shows. He is uniquely positioned to guide the reader clearly and confidently from the big picture to the smallest detail—offering phased accessibility strategies that fit any budgetary or time constraint, and straight talk that clarifies regulations and debunks myths. Moreover, Clark cares as much about design aesthetics as access, and he shows how the two are compatible. As with Eric Meyer on CSS (see [Chapter 9](#)), we recommend this book unreservedly.

-

Constructing Accessible Web Sites, various authors (Glasshaus: 2002)

Written by multiple subject matter experts including Jim Thatcher, Shawn Layton Henry, Paul Bohman, and Michael Burks, this task-focused book is like a wonderful compendium of best-of-breed magazine articles, each of which covers in thoroughly researched and practical detail a particular aspect of the access puzzle. Among other subjects covered, Constructing Accessible Web Sites includes vital material on the limitations of push-button accessibility testing software, discusses feasible methods of implementing accessibility across large enterprises, and provides detailed tips on accessible Flash MX authoring.

Although it lacks the Clark book's advantage of a single, authorial voice, the choir that Constructing Accessible Web Sites offers instead is compelling in its own right. Both books demand space on the shelf of anyone who designs, builds, owns, or manages websites. Among other benefits, reading these two books will help overturn many mistaken and sadly widespread ideas such as those we are about to discuss.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Widespread Confusion

Presented with the notion of accessibility, many otherwise savvy designers, developers, and site owners tend to spout meaningless aphorisms about serving their customers. Informed of accessibility laws such as U.S. Section 508 of the Workforce Investment Act, they often subside into what we can only characterize as mental incontinence.

The Genius Puts His Foot in It

On more than one stage, lecturing to a professional audience, we have heard a highly respected fellow web designer respond to a spectator's accessibility question with nonsense like this: "We create cutting-edge branding work for the elite consumers our client is trying to reach. That accessibility stuff—that's a very small part of our market, and uh our client doesn't mind losing those few people. I mean, hey, our client makes high-definition wide-screen TVs. Blind folks aren't buying those this year (chuckle, chuckle)."

Actually, blind people might buy those TVs for a sighted partner or family member if the site allowed them to read the specifications and use the online ordering forms. Moreover, the overwhelming majority of visually impaired web users are not entirely blind or even close to it. Most who require access enhancements range from people with low vision to the color blind to the slightly myopic, and any of these might desire and be willing to purchase a high-quality, big picture television set.

The "Blind Billionaire"

Moreover, web crawlers are, in effect, blind users. At a recent conference we attended, a speaker pointed out that the Google search engine is the biggest blind user on the web, and this "user" gives out recommendations in the form of search results to a metric ton of customers every minute of every day.

Looked at (sorry) another way, Google's aggregate readership makes the search engine much like a blind billionaire. How many sites would willingly say no to potential clients who have a few billion dollars to spend? Considered a third way, if you count all the disabled folks in America alone, it's something like the combined populations of the greater metro areas of Los Angeles and New York City. Would it be smart to exclude everyone living in and around those cities from your site?

Access Is Not Limited to the Visually Impaired

But access is not limited to the visually impaired. Motor impaired (partly or completely paralyzed) consumers might want to buy a nice television and might also prefer shopping online to hassling with a trip to a brick and mortar store. Many access enhancements are targeted to that group rather than to the cane-and-tin-cup crowd that invariably springs to mind when the ill-informed contemplate accessibility. Access enhancements also help nondisabled consumers who are attempting to buy that spiffy TV while viewing the site on a Palm Pilot or a web-enabled phone.

In short, the designer or developer or site owner who says "Blind people don't buy our products" is missing the point and the boat. He himself is blind to the true nature of the audience he needlessly rejects—including millions of nondisabled visitors who might have found his site via a search engine if the site had only made an effort to conform to access guidelines. Sadly, he is not unique in misunderstanding what access entails and whom it serves.

A Cloud of Fuzzy Ideas

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Law and the Layout

The volume of confusion was already high when the passage of Section 508 of the U.S. Workforce Investment Act cranked it up to 11. (Note: We write from an American perspective and use American examples in this section, but the same principles apply no matter where you are and no matter what your local laws might be.)

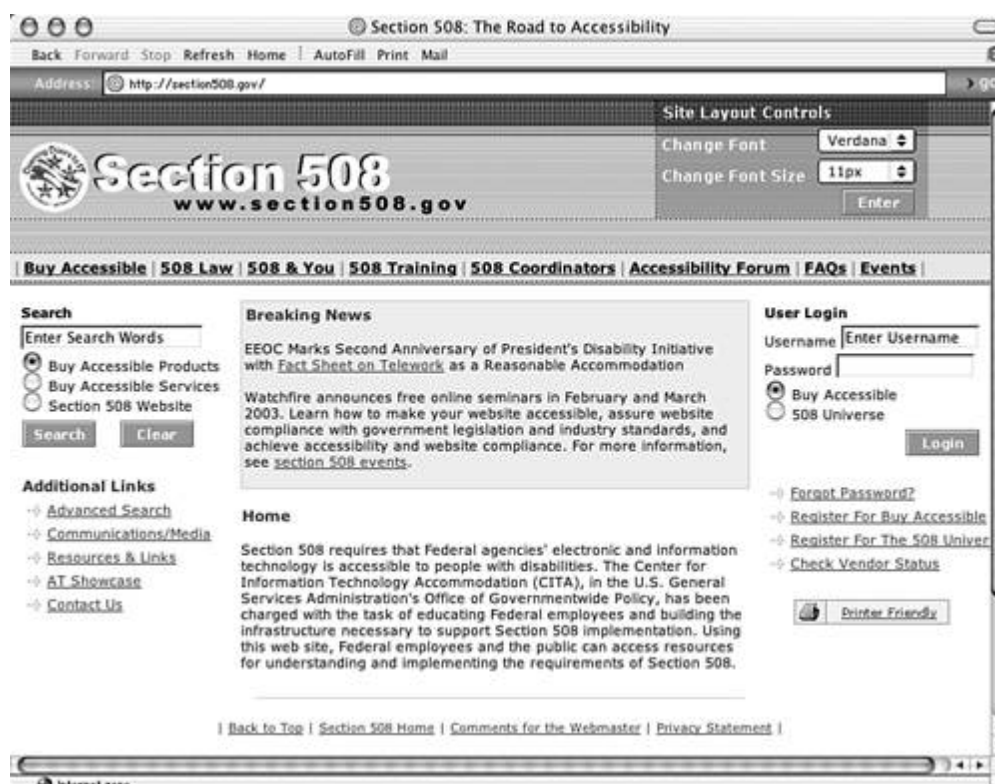
Section 508 requires that many sites accommodate people with disabilities ranging from limited mobility to a vast range of visual impairments, and it spells out what accessible means. (Hint: Adding alt attributes to your images is not enough.) Faced with such a task, many web professionals conclude that accessibility means text-only pages or unattractive, "low-end" design. This isn't so.

Images, CSS, table layouts, JavaScript, and other staples of contemporary web design are entirely compatible with 508 compliance; they simply require a little extra care. As this chapter progresses, we'll examine some of what accessibility and Section 508 compliance specifically entails, and we'll explore how you can use intelligent judgment and available tools to make your sites comply beautifully.

Section 508 Explained

Section 508 [14.2] is part of the Rehabilitation Act of 1973, which is intended to end discrimination against people who have disabilities. Enacted by the U.S. Congress on August 7, 1998, Public Law 105-220 (Rehabilitation Act Amendments of 1998 [<http://www.usworkforce.org/wialaw.txt>]) significantly expanded 508's technology access requirements. The law covers computers, FAX machines, copiers, telephones, transaction machines and kiosks, and other equipment used for transmitting, receiving, or storing information. It also covers many websites.

14.2. The official site of Section 508 complies with its own guidelines (www.section508.gov). This is not always the case for government sites and was not originally the case for section508.gov. Note the ALA-influenced type size switchers at top right, intended to compensate for browsers that don't let users resize text. See [Chapter 15](#), "Working with DOM-Based Scripts," for more about style switchers.



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Accessibility Myths Debunked

The myths that follow are but a few of many that rattle the heads of otherwise clear-thinking web professionals. Our rebuttals are likewise limited in scope due to chapter constraints.

Myth: Accessibility Forces You to Create Two Versions of Your Site

Not true. If you design with web standards and follow certain guidelines, your site should be as accessible to screen readers, Lynx, PDAs, and old browsers as it is to modern, compliant browsers. Standards and accessibility converge in agreeing that one web document should serve all readers and users.

If you design exclusively in Macromedia Director/Shockwave, then, yes, to conform to WAI Priority 1 or Section 508 guidelines, you need to create a separate version. If access is part of your brief, design with standards, and save Shockwave for assignments where it's more appropriate. Note that an upcoming version of Macromedia Director will enable developers to create far more accessible Shockwave files. Whether those files will fully meet WAI or other access guidelines remains to be seen.

Myth: A Text-Only Version Satisfies the Requirement for Equal or Equivalent Access

Not true. Adaptive technology has come a long way, and most anything on a conventional web page can be made fully or at least partially accessible, with no visible alteration to your layout. (Remember: The work takes place under the hood.) Shuttling off disabled visitors to a text-only site assumes that the color blind can't see at all, or that those who have limited mobility have no use for images. It also assumes that these users have no interest in shopping on your commerce site or participating in an online discussion forum. In short, the outdated text-only approach helps no one. Not only that, creating and maintaining text-only pages costs more than simply adding access tags and attributes to your site.

Myth: Accessibility Costs Too Much

Not true. What is the cost of creating a Skip Navigation link or of writing a table summary, as described in [Chapter 8](#)? What is the cost of typing a brief alt text for each image on your page? Such tasks can be accomplished in minutes. You might charge by the hour, but unless you charge millions per hour, the cost of adding most access features required by WAI Priority 1 or U.S. Section 508 is negligible—especially if so doing protects you from antidiscrimination lawsuits. And those can cost plenty.

Higher-level conformance entailing specialized work will, of course, cost more than these simple tasks. For instance, it costs significantly more to author closed captions for Real or QuickTime videos or to caption live streaming media news feeds in real time. But few sites deliver the kind of content that requires such specialized work. As we said at this chapter's outset, access is a continuum; complying with its basic guidelines costs next to nothing.

With large sites whose content is updated by nondevelopment (editorial) personnel, adding access to new pages is often as simple as updating the content management system to prompt for required attributes. On a dynamic site, it might be as easy as reworking the templates that generate pages on the fly. Adding access attributes to forms, structural summaries to table layouts, and making other, similar adjustments to global templates can be a one-time cost that pays off by welcoming new customers and waving garlic before the occasional vampires of the legal profession.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Accessibility Tips, Element by Element

The following guidelines offer approaches to bringing commonly used web page elements into conformance with WAI (or governmental) accessibility guidelines.

Images

Leaving out alt text will cause users of Lynx, screen readers, and other nonmainstream browsers and devices to hear or see [IMAGE] [IMAGE] [IMAGE] [IMAGE] [IMAGE] [IMAGE] or something equally unhelpful. For a visual example, flip back to [Chapter 2, "Designing & Building with Standards," Figure 2.5](#). Lack of alt texts will also be flagged as a WAI access error and an XHTML validation error. Use the alt attribute to the img element (<http://www.w3.org/WAI/GL/WCAG20/checkpoints.html>) to describe the purpose of each image.

Your Friend, the null alt Attribute

For meaningless images, such as spacer GIFs (not that you're still using spacer GIF images), use alt="", also known as the null alt attribute, or null alt text. Do not compound users' problems with literal alt text for meaningless images like alt="pixel spacer gif" or alt="table cell background color gradient". Use the null alt attribute for images that are intended to create purely visual (nonmeaningful, nonsemantic) design effects.

Use alt Attributes That Convey Meaning to Your Visitors

Use alt attributes that convey meaning to your visitors, rather than meaning to you and your colleagues. For instance, on a logo that also works as a link back to the home page, use alt="Smith Company home page" instead of alt="smith_logo_rev3" or alt="Smith Company logo". To a visually disabled user, it is of scant interest to be told that an image she can't see is a "logo." The fact that clicking the image will take her back to the home page is far more significant. If you feel you must, you can hedge your bets by writing something like this:

```
alt="Smith Company home page [logo]"
```

Resist "helpful" software that generates alt attributes for you; this software will most likely generate useless alt texts derived directly from filenames:

```
alt="smith_logo_32x32"
```

In short, never send a robot to do a human being's job. In fact

Don't Trust Software to Do a Human Being's Job

Don't assume that your alt attributes work if your page passes Bobby's WAI or Section 508 accessibility tests. A page that uses alt="mickeymouse" for every image (or alt="" for every image) could pass these tests just fine. No software can tell if your alt texts are appropriate. And frankly, we wouldn't want to live in a world where software could make these kinds of judgments. If you don't know what we're talking about, watch 2001, Blade Runner, The Matrix, Minority Report, or The Music Man. Okay, The Music Man has absolutely nothing to do with it, but wasn't Robert Preston great in that role? Come on, he was really great. Kids, ask your parents about it.

The alt ToolTip Fandango

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Tools of the Trade

If you use a visual editor to create web pages, several tools and plug-ins can simplify conformance with access guidelines:

- SSB Insight LE and GoLive

This free SSB Insight LE plug-in for Adobe GoLive automatically identifies many (but not all) accessibility violations.
- <http://www.adobe.com/products/golive/ssb.html>
- UseableNet LIFT and Dreamweaver

Mentioned earlier in this chapter, UsableNet's \$249 LIFT for Macromedia Dreamweaver 4 offers numerous features in addition to assisting with conformance. LIFT automatically identifies many (though not all) accessibility violations. There are also standalone versions of LIFT, and two recent versions incorporate the usability guidelines recommended by the Nielsen Norman Group.
- http://www.usablenet.com/lift_dw/lift_dw.html
- Dreamweaver MX

Many of LIFT's capabilities have been incorporated in Dreamweaver MX, including a built-in 508 validation checker, a 508 Reference Guide, and tools for adding accessibility features to images, tables, and frames. Remember: No tool catches all problems or offers a fail-safe substitute for your judgment and experience. Like a hammer and nails, tools only help those who know how to use them. Like eggs, milk, and flour okay, never mind, you get the picture.
- Microsoft FrontPage Gets LIFT

UsableNet announced on July 9, 2002 that it had integrated its LIFT product into Microsoft FrontPage, the widely distributed authoring tool: http://www.usablenet.com/frontend/onenews.go?news_id=45

LIFT won't stop FrontPage from generating proprietary, nonstandard markup, but as in Dreamweaver MX, the inclusion of LIFT will enable FrontPage users to check for access problems and guide them toward including accessibility features in images, tables, frames, and so on.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Working with Bobby

Whether you use the products mentioned earlier or mark up and code your sites by hand, Watchfire's Bobby Accessibility Validator should be your next stop:

<http://bobby.watchfire.com/bobby/html/en/index.jsp>

With the touch of a button, Bobby can test any page for access conformance, although the nuances require judgment and analysis. Both WAI and Section 508 rely on a manual checklist to ensure compliance. Unlike the W3C's markup and CSS validation services, Bobby's validation tests cannot provide you with an unconditionally clean bill of health or a list of mistakes to be fixed. Instead, you must interpret Bobby's output. That's where things get tricky. But it's also where they become educational and where you get to earn your paycheck as a knowledgeable designer, developer, or related web specialist.

Understanding Checklists

"If the Section 508 issues listed below do not apply to your page, then it qualifies as Bobby Section 508 Approved," says Bobby after its free online version (or powerful binary version available for purchase) is finished assessing your page. Bobby will never say, "Dude, you've totally nailed it. This page is absolutely 100% compliant with WAI Priority 1 guidelines" or Section 508 or any other published access specification.

Bobby can't say that. Bobby is software. Software relies on algorithms to test for the presence of common problems. And, as we keep saying, many problems evade a machine's understanding.

We don't have time to describe every situation you might encounter when testing your site for access conformance, but this example will show you how to understand and apply the kinds of checklists that Bobby generates. In an encounter with Bobby, a hybrid (tables plus CSS) version of zeldman.com passed the software's Section 508 test, but true approval was contingent on interpreting a checklist that Bobby generated.

Bobby's checklist included this item: "Consider specifying a logical tab order among form controls, links, and objects."

Keeping Tabs: Our Good Friend, the tabindex Attribute

The XHTML tabindex attribute specifies the tabbing navigation order among form controls. If you don't create a logical tab order, people who rely on tabbing (instead of the mouse) will simply tab from link to link in the order that links appear in your XHTML source. This might not be the most useful way to guide them through your site, particularly if your body text contains numerous links or long-winded navigation that occurs early in markup.

Like Skip Navigation and accesskey (discussed in [Chapter 8](#)), tabindex spares screen reader users from the worst aspects of serial navigation, enabling them to quickly skip to content that interests them. Whereas Skip Navigation leapfrogs long lists of links, and accesskey provides command key access (at least theoretically) to various page components, tabindex provides shortcut serial access to various parts of the page—not unlike a DVD's chapter index, which lets movie fans skip ahead to the car chase or back to the love scene.

On commercial sites, after creating a tab order as described next, you would test on real users. On personal or

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Planning for Access: How You Benefit

Although many sites are not legally required to provide access today, they might have to do so tomorrow. One thing we all know about laws is that they continually change. Another thing we know is that we are all subject to laws, whether we like them or not. Applying these enhancements to your site, even if you are not required to do so under today's laws, might protect you from expensive retooling should the laws change next year, and might also protect you from the cost (and bad public relations publicity) of antidiscrimination lawsuits.

Access Serves You

Having trotted out the rationale behind many site owners' sudden interest in accessibility, let us hastily add that fear of lawsuits is the wrong reason to incorporate access into your design practice. These enhancements open any site to new visitors—and whose site could not use more visitors? Those locked out of other sites will be inclined to feel quite loyal to yours if you welcome them into it by making these adjustments to your markup. If other online stores block disabled visitors and nontraditional device users and your store welcomes them, guess who will be selling to those customers, and guess who won't be?

And don't forget, the more accessible your site is to disabled visitors and nontraditional Internet device users, the more available its content will also be to Google, AlltheWeb, and all the other crawler-driven search engines and directories that send visitors your way. Conversely, the less accessible your site, the less traffic it will draw from Google and its brothers. zeldman.com, A List Apart, and most sites designed by Happy Cog over the past few years tend to rank high in search engine results, not because we're doing anything fancy, but because these sites are built with access-enhanced structural markup.

Gosh, we were trying to attain higher moral ground, and we still seem to have offered purely self-interested reasons for implementing accessibility. Here are two more:

Implementing access enhancements can deepen your understanding of "design." Considering things like tab order can take you beyond a vision of design as the decoration of surface appearance ("look and feel") and into the realms of user flow, contingency design, and general usability. These are issues that web designers, information architects, and usability specialists think about anyway. Accessibility is just another aspect of considering how to best build our sites to meet diverse human needs.

Implementing access and honing a conformance strategy can sharpen your development skills and provide fresh perspectives you might never have considered otherwise. Learning the ways of WAI and the particulars of 508 (or any other legally defined access standard) will increase your value as a professional web designer, position your web agency as smarter and more clued-in than its competitors, and help your sites reach more people than ever before. That is what every site owner wants and what every designer or developer strives for. Practicing accessibility will help your visitors reach their goals, yes; but it will also help you reach your own.

Chapter Fifteen. Working with DOM-Based Scripts

In the beginning, Netscape created JavaScript, and it was good. Then Microsoft begat JScript, and it was different. Vast armies clashed by night and the flames of DHTML threatened to engulf all. Salvation arrived with the birth of a standard Document Object Model (DOM), whose first manifestation was called DOM Level 1 (<http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/>). And it was very good indeed. For the first time, the W3C DOM gave designers and builders a standard means of accessing the data, scripts, and presentation layers with which their sites were composed.

In the years since, the W3C has continued to update its DOM specs, and, at the urging of The Web Standards Project (WaSP), browsers have come to support at least most of the DOM Level 1 specification, although they sometimes differ in the ways they support it. (To find out how much DOM your favorite browser upholds, visit <http://www.w3.org/2003/02/06-dom-support.html>.) In this chapter, we will meet the DOM and explore some of the ways it can help us accomplish useful tasks, such as showing or hiding content in response to visitor actions, providing customization and accessibility options, and creating dynamic menus. None of this will be too difficult or technical, we promise.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Meet the DOM

Just what is the DOM? According to the World Wide Web Consortium (W3C) (<http://www.w3.org/DOM/>), the DOM is a browser-independent, platform-neutral, language-neutral interface that allows "programs and scripts to dynamically access and update the content, structure, and style of documents. The document can be further processed, and the results of that processing can be incorporated back into the presented page."

In simple English, the DOM makes other standard components of your page (style sheets, markup elements, and scripts) accessible to manipulation. If your web page were a movie, XHTML would be the screenwriter, CSS would be the art director, scripting languages would be the special effects, and the DOM would be the director who oversees the entire production.

As a bonus, instead of taxing the server and clogging the pipes with HTTP requests, DOM-driven interactivity takes place on the client side (that is, on the visitor's hard drive). It works even if the Internet connection is terminated.

A Standard Way to Make Web Pages Behave Like Applications

Although such usage exceeds the scope of this book, the most exciting aspect of DOM-driven interactivity is that it can mimic the behavior of conventional software. For instance, the visitor can change the sort order of tabular data by clicking on the header, just as she might do in an Excel spreadsheet or in the Macintosh Finder (the application that lets Macintosh users sort, copy, move, rename, delete, or in other ways process various files and folders on their desktops) [[15.1](#), [15.2](#), [15.3](#)].

15.1. The DOM enables web pages to behave like desktop applications. In this demo by Porter Glendinning, data is sorted by album title when the user clicks the Album header (<http://glendinning.org/webbuilder/sortTable/>).

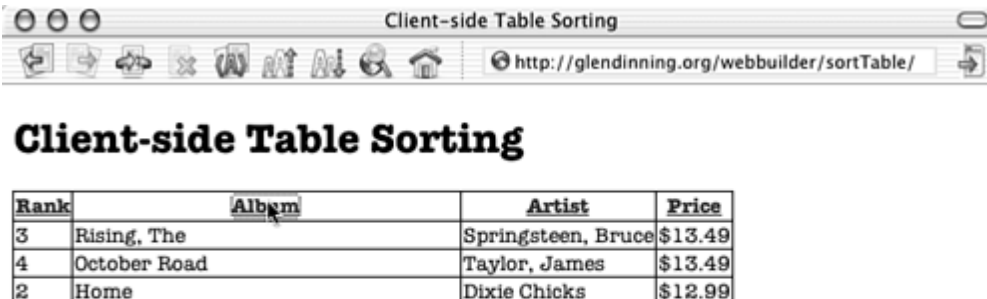


The screenshot shows a web browser window with the title "Client-side Table Sorting". The address bar displays the URL "http://glendinning.org/webbuilder/sortTable/". Below the browser window, the title "Client-side Table Sorting" is repeated. The main content is a table with four columns: Rank, Album, Artist, and Price. The table is sorted by the Album column. The data is as follows:

Rank	Album	Artist	Price
1	Before Your Love/A Moment Like This	Clarkson, Kelly	\$4.49
5	Bounce [Digipack]	Bon Jovi	\$12.99
2	Home	Dixie Chicks	\$12.99
4	October Road	Taylor, James	\$13.49
3	Rising, The	Springsteen, Bruce	\$13.49

Below the table, there is a text area containing the text "javascript:;" and a small icon.

15.2. Clicking the header again reverses the sort order. The changed order is immediately visible on the original page. A separate "results" page is not needed.



The screenshot shows a web browser window with the title "Client-side Table Sorting". The address bar displays the URL "http://glendinning.org/webbuilder/sortTable/". Below the browser window, the title "Client-side Table Sorting" is repeated. The main content is a table with four columns: Rank, Album, Artist, and Price. The table is sorted by the Album column in reverse order. The data is as follows:

Rank	Album	Artist	Price
3	Rising, The	Springsteen, Bruce	\$13.49
4	October Road	Taylor, James	\$13.49
2	Home	Dixie Chicks	\$12.99

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Please, DOM, Don't Hurt 'Em

DOM-based scripts won't do a bit of good for browsers that entirely lack support for the W3C DOM standard. Netscape 4 is our chief problem here. The solution is to sniff for DOM compliance, and in its absence, to serve alternate content or an alternate web page.

That we sometimes need to do so is regrettable. The goal of standards is to serve the same content to all user agents. In the case of HTML and XHTML, we can do exactly that. Where CSS is concerned, in [Chapter 9](#), "CSS Basics," we saw how the Two-Sheet method enables us to send the same files to all user agents, while using the @import directive to protect old browsers from styles they can't handle. And in [Chapters 12](#), "Working with Browsers Part II: Box Models, Bugs, and Workarounds," and [13](#), "Working with Browsers Part III: Typography," we learned how the Box Model Hack can exploit parsing bugs in IE5/Windows and outdated versions of Opera to serve the same styles to all without generating errors in browsers like IE5/Windows that get font sizes and the box model wrong.

In each of these cases, we've managed to serve the same files to all comers, albeit with the occasional stripe of tar to patch compliance potholes; however, we cannot smooth over lack of DOM support in the same way. Although IE5/Windows bungles the box model, it is still a CSS browser that can parse a CSS file. But Netscape 4 is in no way a DOM-compliant browser—and in fairness to the manufacturer, the W3C was still hammering out the DOM specification at the time Netscape 4 was released. Unless you are Nostradamus, you can't support what doesn't yet exist.

We can no more expect a non-DOM-compliant browser to make use of a DOM-driven script than we can demand that a chicken make use of a calculator. What we can do is test for knowledge of the DOM, and, in its absence, send the browser to an alternate page (or, better still, insert alternate content as needed via Server Side Includes (SSI), Active Server Pages (ASP), or any of the other middleware solutions).

How It Works

Dori Smith (www.dori.com) of WaSP invented the DOM sniff in early 2001 as a means of implementing that group's Browser Upgrade Campaign (discussed in [Chapter 4](#), "[XML Conquers the World \[And Other Web Standards Success Stories\]](#)"). But it can be used for any purpose, in most cases replacing complicated browser detection scripts that are nearly impossible to keep up-to-date.

The way the DOM sniff works is simple. After you create a valid page, insert the following script in the <head> of your document or somewhere in a linked JavaScript (.js) file:

```
if (!document.getElementById) {  
    window.location =  
        "http://www.somesite.com/somepage/"  
}
```

where somesite.com represents your website and /somepage/ represents an alternate page whose content Netscape 4 (or any other non-DOM-capable browser) can handle. This alternate page might use JavaScript to emulate the behavior of the "standard" page, or it might more usefully contain script-free content that any browser can handle.

The alternate page might not even do that. Instead, it might advise the user to download Netscape 7 or a similarly compliant browser. This "upgrade now" approach is outdated and will not please Netscape 4 users who are unable to upgrade because of stupid corporate policies. That was the legitimate gripe against WaSP's Browser Upgrade

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

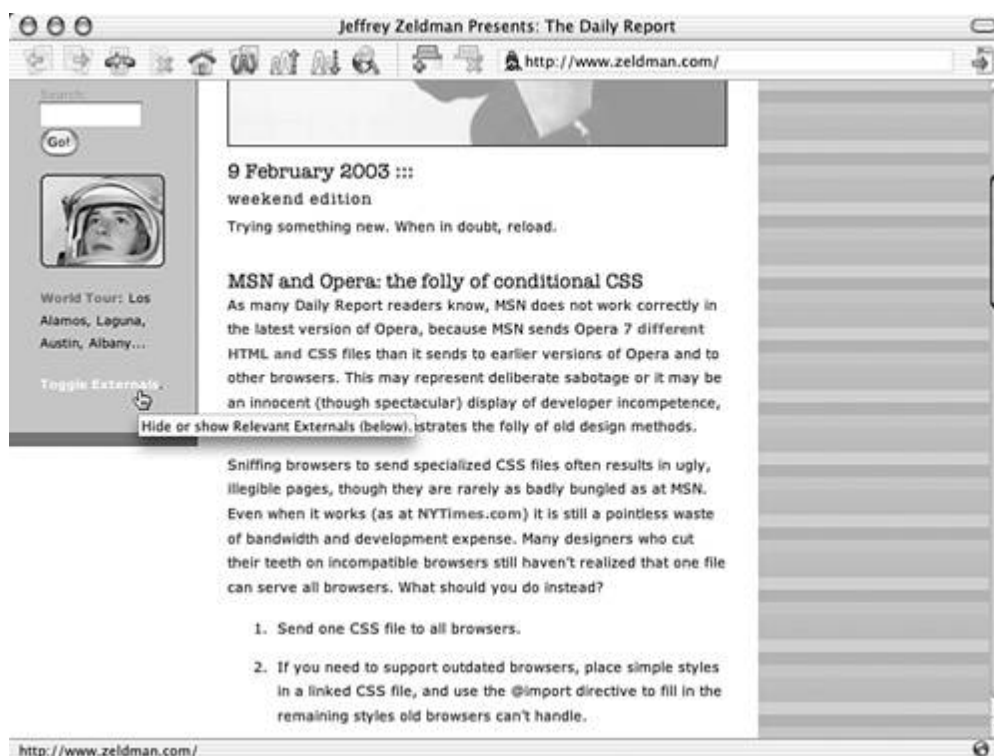
◀ PREVIOUS

NEXT ▶

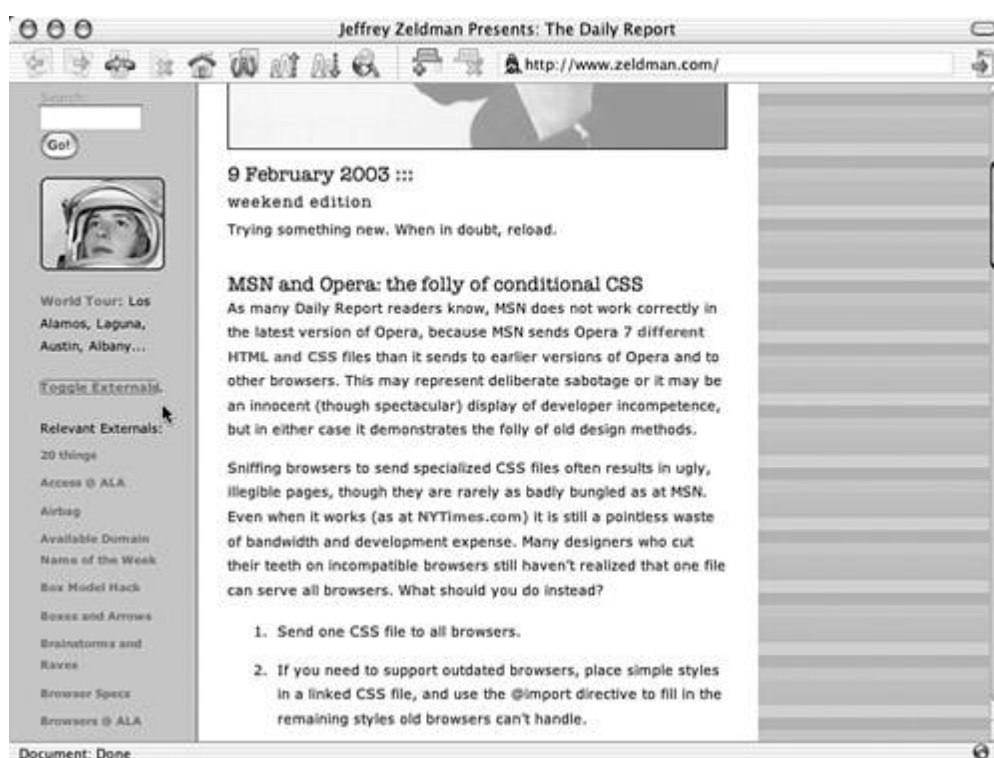
Showing and Hiding

For many reasons, you might want to cloak some of your page's content when the page loads, revealing it in response to a visitor's actions [15.8, 15.9, 15.10, 15.11]. The DOM makes it easy to show and hide content.

15.8. DOM-based showing and hiding at its most basic. The page's sidebar contains a list of links whose display is initially hidden from view (www.zeldman.com).



15.9. When Toggle Externals is clicked, the list is revealed.



15.10. DOM-based showing and hiding can be combined easily with other effects (<http://www.happyvox.com/projects/>)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Dynamic Menus (Drop-Down and Expandable)

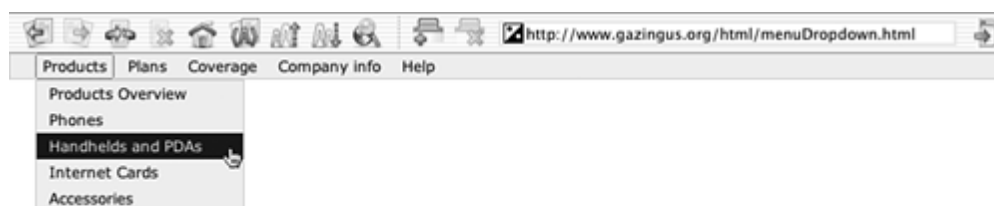
DHTML menus have at least four kinds of stink on them:

- Developers recall the overlarge marketing promises, undersized performance, and woeful incompatibilities of proprietary 1998 "DHTML."
- Committees incapable of basing site architecture on anything other than the egos of their members have given us multilayered menus that frustrate us instead of guiding us to the content we seek.
- Undertested "DHTML" menus that fail in our browsers have prevented us all from using one site or another.
- WYSIWYG-generated DHTML menus have all too frequently "solved" the cross-browser, cross-platform problem by means of bloated, nonstructural markup and gaseous, browser-specific scripts that double or triple page weight (thereby increasing page wait).

Between shortsighted proprietary methods, non-user-oriented architecture, insufficient testing, and sheer bloat, DHTML is widely perceived as outdated and faintly embarrassing. Like Cousin Elmer's GIF animations, DHTML is seen as one of the web's pointless annoyances; and an aroma redolent of sickness on ships hovers over the entire category.

But the DOM corrects many of these problems by working across conformant browsers and by using compact, structural markup and valid CSS to achieve its ends. Here's one of our favorites: David Lindquist's "Using Lists for DHTML Menus" (<http://www.gazingus.org/dhtml/?id=109>) employs unordered lists to create compact DHTML drop-down [15.12] and expandable menus [15.13, 15.14]. Demos are accompanied by downloadable CSS and JavaScript files that you can use as-is or customize to suit your site. (Sorry, not even web standards can help with the committee-driven site architecture problem.)

15.12. It looks like any other drop-down menu; however, instead of proprietary code and bloated, nonstructural markup, this one uses appropriate list markup, valid CSS, and JavaScript (<http://www.gazingus.org/html/menuDropdown.html>). Download the Source files to use on your site.



15.13. An expandable menu sports clickable plus and minus icons (<http://www.gazingus.org/html/menuExpandable2.html>).



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Style Switchers: Aiding Access, Offering Choice

In [Chapter 13](#), we discussed the impossibility of delivering web type without alienating at least some of your potential visitors, and lamented that 13 years into the web's evolution as a medium, pixels are still the most reliable method of sizing type—and the most troublesome for IE/Windows users. What if you could offer your visitors a choice of user-selectable type approaches? What if you could even change your layout while you were at it?

According to CSS, you can. CSS allows you to associate any web page not only with a default (persistent) style sheet, but also with alternate CSS files. In the interest of enhancing accessibility, these alternate style sheets might offer much larger type or a higher-contrast color scheme [[15.15](#), [15.16](#)]. Or they could completely change the site's appearance for purposes of what was once called "user customization."

15.15. The Picture Collection Online is one of The New York Public Library's many image resource sites (<http://digital.nypl.org/mmpco/>). It uses a style switcher to avoid potential problems for users who suffer from visual impairments.



15.16. When the user chooses Larger, not only does the font size bump up, but contrast is increased, and a lovely (but potentially confusing, for low-vision people) layered background is hidden.



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter Sixteen. A CSS Redesign

In this chapter, we'll take much of what we've learned about markup, CSS, accessibility, and the DOM and put it to work in a site redesign that separates structure from presentation. We'll define and meet goals that serve our site's users and might benefit yours as well. For good measure, once we're pleased with the site's look and feel, we'll create a user-selectable alternative layout powered by CSS and the DOM.

Although the visual design of the site shown in this chapter is nothing special, the construction methods are important, for they are the techniques that we'll use to create many sites soon, and nearly all sites eventually. Previous chapters have discussed bits and pieces of the standards puzzle or told how to create hybrid sites that combine classic and modern methods. This chapter will bring you closer to the typical design practices of tomorrow's web.

This chapter will explore techniques used to create CSS layouts—from the basics of establishing page sections to ideas that are far from basic, such as creating menu buttons out of structured lists and generating JavaScript-like image rollover effects without JavaScript and without even using the XHTML `img` element. In addition to covering specific techniques, we will discuss ways of thinking, such as rules-based versus grid-based design.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Defining Goals

In this redesign project, undertaken specifically for this book, we will retool the look, feel, structure, and basic functionality of zeldman.com, the author's personal site. Whether you capture the attention of thousands of readers or simply work to please yourself and a family member or two, a personal site is invaluable, especially when you're experimenting with CSS and other standards you might not be entirely ready to use in client or in-house development.

A personal site can also help you realize that you are not alone as a designer or practitioner. By creating such a site, you will join a thriving community of standards-oriented designers and developers who are surprisingly eager to help one another. Many times, when stymied by a troublesome problem of CSS, markup, or JavaScript, we've shared our bafflement at zeldman.com. Soon afterward, readers have sent email messages containing the seeds of a solution to the question posed on our site. Likewise, after reading other people's sites, we've often been moved to send them a suggestion about the issues that puzzled them.

We are not unique. This happens to anyone who writes about web design or development issues on his personal site. Try it; you'll like it.

Brand Character

Launched in May 1995, zeldman.com offers tutorials, news, and insights about web design along with anything else that interests us. The brand character is irreverent, passionate, and inquisitive; the design sensibility is clean, minimalist, and quirkily stylish. Our redesign must preserve these attributes.

Top 10 Goals

Additional requirements of the redesign are less brand specific. They might be as relevant to your site as to ours. Listed next are our top 10 goals for this project:

1.

The site must be as usable in nongraphical environments as it is in the best and latest browsers by Netscape, Microsoft, Opera, and others. Its content and basic functions must be available to any browser or device; its layout should work in any reasonably CSS-savvy browser.

2.

Markup must validate against the XHTML 1.0 Transitional spec and must avoid presentational elements. We are separating structure from presentation, which means no nonsemantic spans and divs when p or h1 would serve. It means no spacer GIF images, no tables except those used to present tabular data, no outdated or invalid attributes like bgcolor or marginheight, no misuse of <blockquote> to achieve formatting, and no cheating with
 tags when a structural element can create the desired visual effect while also conveying meaning. (If these goals confuse you, reread [Chapter 7](#), "Tighter, Firmer Pages Guaranteed: Structure and Meta-Structure in Strict and Hybrid Markup.")

3.

CSS must validate and should be as compact and as logically arranged as possible (see [Chapters 9](#), "CSS Basics," and [10](#), "[CSS in Action: A Hybrid Layout \[Part II\]](#)").

4.

To help meet our goal of delivering content and basic functions in almost any conceivable browsing environment, the site should strive to be seamlessly accessible. Too often, the desire to attain accessibility

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Establishing Basic Parameters

We've decided that our template will contain three main areas [[16.3](#)]:

- A brand bar at the top will double as a Home button and might also contain the primary navigation. We will call this area "new menu."
- A sidebar at the left will contain secondary navigation, Search, user interface widgets, blurbs, and links. "Secondary nav" seems like an okay label for this area.
- The primary content area, as one might expect, will hold the site's primary content (which we hope will be more interesting than this sentence). Let's get super creative and name this area "primary content."

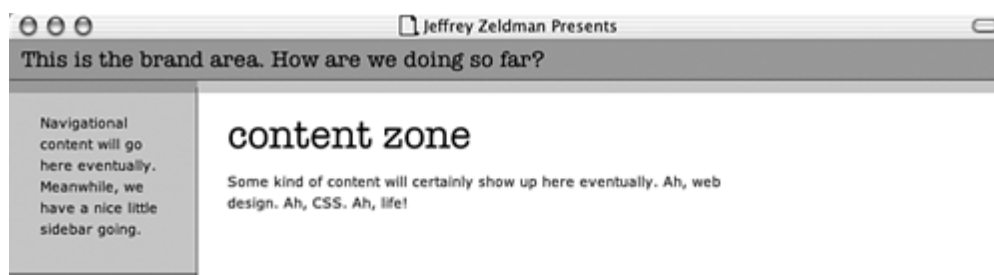
In our XHTML page, following the DOCTYPE, namespace, and meta data, our first, tentative strokes look like this:

```
<div id="newmenu"></div>
<div id="secondarynav"></div>
<div id="primarycontent"></div>
```

Each of these three main areas will eventually flow with the sap of content. But first, each must be styled. In our default (white) style sheet, we establish general layout parameters, grouping them together toward the top. Let's begin with the new menu area, which is visible at the top of [Figure 16.3](#) and stretches all the way from left to right:

```
/* Establish general layout parameters */
#newmenu {
  background: #e0861e;
  color: #000;
  border: 0;
  border-bottom: 1px dotted #000;
  margin: 0;
  padding: 0;
  text-align: left;
  height: 31px;
}
```

16.3. The template will be divided into three main areas.



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Rules-Based Design

We have now laid the groundwork in CSS and in markup for a layout that will work in any browser released in this century, and content that will work in any user agent, period.

As we begin to fill our template with for-instance text and images, we add rules that control the presentation of selected markup elements and also control the way they will interact with one another. Although our activity seems humble and simple—and it is—what we are actually doing is evolving from a comp-driven mindset to a new and more web-like kind of design. In short, we are moving toward rules-based design.

We won't cover every little rule in the style sheet (as mentioned earlier, you can explore it on your own at <http://www.zeldman.com/c/wh.css>), but let's look at one of them as a way of easing into an explanation of rules-based design:

```
p {  
    margin-top: 0;  
    margin-bottom: 1em;  
    font: 11px/1.5 Verdana, Trebuchet, Lucida, Arial, sans-serif;  
}
```

The key elements here are a top margin of 0 and a bottom margin of 1em (that is, roughly speaking, one carriage return based on the size of the text). If we failed to specify this margin-bottom value, browsers would be left to guess at our intentions, and they might leave no vertical space between paragraphs.

But the goal is not merely to avoid unexpected behavior in web browsers. The goal is to establish rules that will control the look and feel on a modular level. Rules-based design is a technique for creating modular design. In setting the top margin to 0, we allow subheads to sit snugly on top of paragraphs (especially if we set subhead margin-bottom values to 0, which we do in a subsequent rule). In setting the margin-bottom value to 1em, we differentiate the space between paragraphs from the space between subheads and paragraphs.

It is a small thing, but CSS layout is filled with such tiny decisions, and they make the difference between design that feels pleasing and controlled and layouts whose random quality signifies a lack of care. Best of all, this feeling of control can be achieved without resorting to convoluted presentational markup and its associated bandwidth costs.

Establishing many small rules of this kind also often frees us from the need to fill our markup with class attributes. For instance, the humble paragraph rule shown earlier frees us from the need to write this:

```
<p class="notopmargin">
```

Of such small steps are giant leaps of bandwidth conservation achieved. Gosh, that sounded pretty, didn't it? As we establish rules for various page elements, we include instructions as to how those elements will interact with one another. For instance, we could create a rule that not only wraps a 1px black border around images but also inserts 10px of whitespace below every image. Using id selectors, we can establish one kind of border and spacing for images that occur in Area A of a layout and an entirely different border and spacing treatment for images that occur in Area B. CSS2 even lets us vary an element's positioning depending on which other elements precede it:

```
p+p {  
    text-indent: 2em;  
    margin-top: -1em;
```


[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

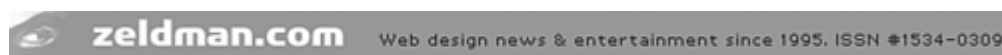
◀ PREVIOUS

NEXT ▶

A Home Button with CSS Rollover Effects

Early in the day, we established a 31px tall "new menu" area. After tinkering with the navigation (discussed later in this chapter), we found that it could be contained entirely in the sidebar. So what should we do with the so-called "new menu" area? We'll use it to hold a branded Home button [16.6].

16.6. Behold one of two similar images that will be used to create a branded Home button with CSS rollover effects.



The Home button will change in response to user mouse activity, and, as in [Chapter 10](#), the rollover effect will be accomplished with CSS, not with JavaScript. As in that chapter, images will be inserted via the CSS background property instead of a conventional `` tag.

In Photoshop, we create two similar images, one of which is shown in [Figure 16.6](#). In the default image, the logo text is off-white (#fdf8f2). When the visitor hovers over the logo, the logo text shifts to pure white (#fff). The effect is too subtle to reproduce in this book, which is why we show one figure instead of two. The background of both GIF images is transparent so that the CSS background color of #newmenu will show through. (Remember: If we tried to match the color in Photoshop, it would be off.)

In our markup, inside the new menu area, we insert a div with the unique identifier "bannerlogoban:"

```
<div id="bannerlogoban"><div>
```

We style it like this:

```
/* Image-free logo banner with rollover */
#bannerlogoban {
  margin: 0;
  padding: 0;
  border: 0;
  width: 600px;
  height: 31px;
  background: url(/i/2003/parts/lbo2.gif) no-repeat;
}
```

The width is 600px to match the actual width of the image. The height is 31px to match the height of the image and the height of the containing element ("new menu"). The background image rule preloads the image in its hover state (lbo2.gif) so there will be no delay when the CSS "rollover" is triggered by the user's mouse movement.

Fahrner Image Replacement (FIR)

Now we have a logo/Home button in its hover state, but it isn't clickable and it doesn't do anything. We actually want the page to show the non-hover state until the visitor's mouse cursor hovers over the image. The next step is kind of mind-bending and is based on a concept pioneered by the oft-mentioned Todd Fahrner. First we create a class called alt whose sole purpose is to be invisible in CSS environments:

```
.alt {
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

A CSS/XHTML Navigation Bar

Consider the sidebar we left naked and aching back in [Figure 16.4](#). Let us now fill it. For instance, let us fill it with navigation so that our visitors can find their way around the site. Conceptually speaking, a nav bar is merely a list of links. Let's honor that notion by typing our links as a standard-issue, unordered list:

[\[View full width\]](#)

```
<ul>
<li id="secondarytop"><a href="/" title="The Daily Report. Web design news and info.
➡">daily report</a></li>
<li id="glam"><a href="/glamorous/" title="My Glamorous Life: Episodes and
recollections.
➡">glamorous</a></li>
<li id="classics"><a href="/classics/" title="Entertainments,
1995&#8211;2002.">classics</
➡a></li>
<li id="about"><a href="/about/" title="History, FAQ, and suchlike.">about</a></li>
<li id="contact"><a href="/contact/" title="Write to us."> contact</a></li>
<li id="essentials"><a href="/essentials/" title="Info for web designers.">essentials</
➡a></li>
<li id="pubs"><a href="/pubs/" title="Zeldman&#8217;s books and articles.">pubs</a></li>
<li id="tour"><a href="/tour/" title="We may be coming to your town: personal
appearances.
➡">tour</a></li>
</ul>
```

That is how any sensible person would format an unordered list. Alas, to avoid the whitespace bug discussed in [Chapter 12](#), we must remove the whitespace, like so:

[\[View full width\]](#)

```
<ul><li id="secondarytop"><a href="/" title="The Daily Report. Web design news and info.
➡">daily report</a></li><li id="glam"><a href="/glamorous/" title="My Glamorous Life:
➡Episodes and recollections.">glamorous</a></li><li id="classics"><a href="/classics/"
➡title="Entertainments, 1995&#8211;2002.">classics</a></li><li id="about"><a
href="/about/
➡" title="History, FAQ, and suchlike.">about</a> </li><li id="contact"><a
href="/contact/"
➡title="Write to us.">contact</a></li><li id="essentials"><a href="/essentials/"
➡title="Info for web designers.">essentials</a></li><li id="pubs"><a href="/pubs/"
➡title="Zeldman&#8217;s books and articles.">pubs</a></li><li id="tour"><a
href="/tour/"
➡title="We may be coming to your town: personal appearances.">tour</a></li></ul>
```

It's pretty darned hard to read, now, isn't it? But we have no choice if we care how the site looks in Internet Explorer for Windows—and we care a heck of a lot about that, of course.

Adding the Style

We've banged out a list. Now we transform it:

```
/* Create buttons */
#secondarynav ul {
  list-style: none;
  padding: 0;
  margin: 15px 0;
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

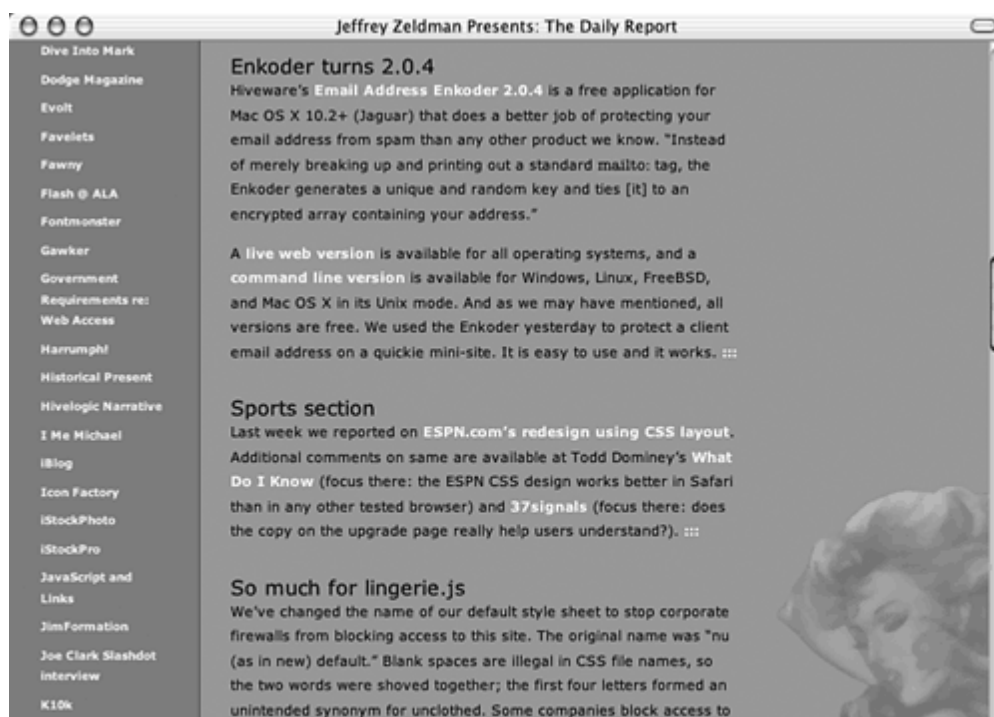
Finishing Up

There is more to the layout than can be covered in one chapter—even a big fat chapter like this one. There are accessibility enhancements made to form elements, including the Search form. There are our old pals Skip Navigation, tabindex, and accesskey. There's the design of a style-switching widget (visible in the center of [Figure 16.10](#)) and the code it rides in on, taken from the open source A List Apart style switcher discussed in [Chapter 15](#), "Working with DOM-Based Scripts." There is the need to actually create an alternate style [[16.11](#)], which largely, although not exclusively, consists of changing color values. And there are details like the "Externals" sidebar [[16.12](#)], which is hidden from view until the visitor decides to open it.

16.11. The alternate ("orange") takes shape.



16.12. The orange layout with the previously hidden "Externals" sidebar switched on. (See [Chapter 15](#)'s discussion of showing and hiding content via the DOM.)



[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Part III: Back End

[A Modern Browsers: The Good, the Bad, and the Ugly](#)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Backend A. Modern Browsers: The Good, the Bad, and the Ugly

As explained in [Chapter 1](#), "99.9% of Websites Are Obsolete," when we refer to "modern" or "standards-compliant" web browsers, we mean browsers that understand and support HTML and XHTML, CSS, ECMAScript, and the W3C Document Object Model (DOM). These baseline standards help designers and developers transcend old-school methods (presentational markup and incompatible scripting languages) and the obsolescence they engender.

No browser yet released is perfectly standards compliant, and none is likely to be. But the dawn of the present millennium saw a crop of browsers that came pretty darned close to full compliance with core baseline standards. As updated versions of these browsers hit the market, they become more compliant and less buggy. They also continue to gain market share.

As of this writing, nearly all web users have updated to one of the browsers listed in the pages that follow or to a subsequent, improved release. This table lists only the most popular browsers and highlights only some of their features (and occasional drawbacks).

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

[\[Team LiB \]](#)

◀ PREVIOUS

Compliant Browsers: The First Wave

Opera 7

Year Released: 2002

Supports HTML/XHTML?: Yes.

Supports CSS?: Almost all of CSS1 and most of CSS2.

Supports ECMAScript/DOM?: Yes, first version of Opera to do so.

Fun Facts: First version of Opera to support W3C DOM, the first truly "standards-compliant" version of Opera. The company has an excellent history of support for earlier standards such as HTML and CSS. Like previous versions of Opera, Opera 7 includes clever Page Zoom feature to help make web text and web graphics more accessible to the visually impaired.

MSIE 5+/Macintosh

Year Released: 2001

Supports HTML/XHTML?: Yes.

Supports CSS?: All of CSS1, some of CSS2.

Supports ECMAScript/DOM?: Yes.

Fun Facts: Released in March 2001. First "standards-compliant" browser to market, first version of IE/Mac to correctly support JavaScript/ ECMAScript, and first browser on any platform to correctly support CSS box model. Includes clever Text Zoom feature to help make web text more accessible to the visually impaired. Supports user style sheets.

Browser's Tasman rendering engine, which is responsible for improved standards support, is beginning to show its age. IE5/Macintosh's DOM support is good, but not as complete as one might hope. It's slow, and its occasional quirks drive some dynamic content authors crazy. Still, the browser supports basic DOM functions and is quite good overall from a standards point of view.

Netscape 6+

Year Released: 2001

[\[Team LiB \]](#)

◀ PREVIOUS

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#) [\[Z\]](#)
]

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#) [\[Z\]](#)
[\]](#)

["you are here" indicator](#) [2nd](#) [3rd](#)

[& \(ampersands\)](#)

[’](#)

[<!N](#)

[*****change standards to web standars for final index*****](#)

[*****Gecko s/b Gecko browsers](#)

[****Bobby s/b Bobby Accessibility Validator](#)

[****less than sign inserted here \(less than sign\)](#)

[****markup and markup language are the same thing. Combine during edit.](#)

[@import](#) [2nd](#)

[font size keywords](#)

[16px at 96ppi](#)

[16px/96ppi](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[1em](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[A List Apart](#). [[See ALA](#)]

[absolute file reference links](#)

[absolute URLs](#)

[absolute values](#)

[pixels](#)

[academics](#)

[versus economics](#) [2nd](#)

[acceptance](#)

[of XML](#) [2nd](#) [3rd](#)

[accessibility](#)

[resources for](#) [2nd](#)

[Building Accessible Web Sites \(s/b ital\)](#) [2nd](#)

[Constructing Accessible Web Sites \(s/b ital\)](#) [2nd](#)

[accessibility](#) [2nd](#) [3rd](#) [4th](#)

[applets](#)

[benefits of](#) [2nd](#) [3rd](#) [4th](#)

[captioning tools](#)

[colors](#) [2nd](#)

[CSS](#) [2nd](#) [3rd](#) [4th](#)

[sizing text](#)

[Dreamweaver](#) [2nd](#)

[Dreamweaver MX](#)

[Flash](#) [2nd](#)

[Flash 4/5](#)

[Flash MX](#) [2nd](#) [3rd](#)

[flashing or blinking elements](#)

[for visually impaired](#)

[forms](#) [2nd](#)

[frames](#)

[FrontPage](#)

[GoLive](#)

[image maps](#) [2nd](#)

[images](#)

[alt attribute](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[background images](#)

[null alt attribute](#)

[misguided opinions of some web professionals](#) [2nd](#) [3rd](#)

[motor impaired](#)

[myths](#)

[accessibility is expensive](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[accessibility is only for the disabled](#) [2nd](#)

[designers can ignore laws if clients tell them to](#)

[sites must be identical in all browsers and agents](#)

[text-only versions satisfy requirements](#)

[there are tools that solve all compliance problems](#) [2nd](#)

[you must create low-end designs](#) [2nd](#)

[you must have two versions of your site](#)

[rollovers](#) [2nd](#)

[non-mouse users](#)

[noscript](#) [2nd](#)

[software](#)

[SSB Insight LE](#)

[standards](#)

[streaming video media](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

background colors

[Netscape 4](#)

[specifying in CSS](#) [2nd](#)

background images

[alt attribute](#)

[removing](#)

backgrounds

[images](#) [2nd](#)

[white backgrounds](#)

[backward compatibility](#) [2nd](#) [3rd](#)

[bandwidth](#) [2nd](#) [3rd](#)

bandwidth

[backward compatibility](#) [2nd](#) [3rd](#)

[image maps](#)

[saving](#) [2nd](#) [3rd](#)

baselines

Standards mode

[Gecko](#) [2nd](#)

[Be Kind to Opera rule](#)

be nice to Opera rule

[Fahrner method](#)

behavior

of sites

[standards](#) [2nd](#)

behaviors

[IE4](#) [2nd](#)

[Netscape 4](#) [2nd](#)

benefits

[of accessibility](#) [2nd](#) [3rd](#) [4th](#)

[of best-case scenario](#) [2nd](#)

[of CSS](#) [2nd](#) [3rd](#)

[of standards](#) [2nd](#) [3rd](#) [4th](#)

[of strict forward compatibility](#)

[of transitional forward compatibility](#)

[of transitional methods](#) [2nd](#) [3rd](#) [4th](#)

[of Two-Sheet Method](#) [2nd](#)

[Berners-Lee, Tim](#) [2nd](#)

[bgcolor attribute](#)

[blind.](#) [See [visually impaired](#)]

blinking elements

[accessibility](#)

[block](#) [2nd](#)

[block-level elements](#) [2nd](#)

[IE6](#)

[primenav](#) [2nd](#) [3rd](#) [4th](#)

[XHTML](#)

blocking users

[by designing for only one browser](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[blocking.](#) [See also [excluding](#)]

[Blue Robot's Layour Reservoir](#)

[Bobby \(Watchfire\)](#)

[Bobby Accessibility Validator](#) [2nd](#)

[building and testing](#) [2nd](#) [3rd](#)

[checklists](#) [2nd](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[Camino.](#) [[See](#) [Chimera](#)]

capitalization

[CSS](#)

[lowercase.](#) [[See](#) [lowercase](#)]

[of attribute values or content](#) [2nd](#)

[captioning tools](#)

[Carter, Matthew](#)

[Cascading Style Sheets.](#) [[See](#) [CSS](#)]

case insensitivity

[CSS](#)

[Celik, Tantek](#) [2nd](#)

[Box Model Hack](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

cells

[tables](#)

[errors in hybrid markup](#) [2nd](#) [3rd](#)

centering

[text](#)

changing

[to standards](#)

[character encoding](#)

character sets

[ISO 8859](#)

[mapping to Unicode](#) [2nd](#)

[Unicode](#)

checklists

[Bobby Accessibility Validator](#) [2nd](#)

children

[inheritance](#)

[CSS](#)

[Chimera](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

choosing

[DOCTYPE](#) [2nd](#) [3rd](#) [4th](#)

Clark, Joe

[Building Accessible Web Sites \(s/b ital\)](#) [2nd](#)

class

[versus id](#) [2nd](#)

[class selectors](#) [2nd](#) [3rd](#)

classes

[errors in hybrid markup](#) [2nd](#)

[hide](#)

[classitis](#) [2nd](#)

[visual editors](#)

[Cleartype](#)

[client side image maps](#)

clients

[contracts](#)

[showing them what you plan to do](#) [2nd](#)

[showing them your work](#) [2nd](#)

clock faces

[CSS values](#)

closing

[tags](#)

[XHTML](#) [2nd](#) [3rd](#) [4th](#)

code

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[debugging](#)

declarations

[CSS](#)

[multiple declarations](#) [2nd](#)

[properties and values](#)

[semicolons](#)

declaring

content type

[XHTML](#)

[deprecated HTML elements](#)

[descendant selectors](#) [2nd](#) [3rd](#)

design methods

[best-case scenario](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#)

[two-sheet method](#)

[Two-Sheet Method](#) [2nd](#) [3rd](#)

[designing.](#) [See also [redesign projects](#)]

designs

cost

[before standards](#) [2nd](#)

[rules-based designs](#) [2nd](#) [3rd](#) [4th](#)

[detection scripts](#)

[DevEdge](#)

DHTML

[Internet Explorer 4](#)

[menus](#) [2nd](#) [3rd](#)

[Netscape 4](#)

[DHTML \(Dynamic HTML\)](#)

disabilities

[impaired mobility](#)

[disabilities.](#) [See [accessibility](#)]

displays

[toggling](#) [2nd](#) [3rd](#)

[div](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[errors in hybrid markup](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[id](#)

[reason for using](#) [2nd](#)

[divitis](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

DOCTYPE

[choosing](#) [2nd](#) [3rd](#) [4th](#)

namespaces

[XHTML](#) [2nd](#)

[opening with](#)

[relative links](#)

[DOCTYPE switching](#) [2nd](#) [3rd](#)

[Gecko browsers](#)

[history of](#) [2nd](#) [3rd](#) [4th](#)

[IE5/Mac](#)

[IE6](#)

[Opera](#)

[toggling between modes](#) [2nd](#)

DOCTYPEs

[complete and incomplete](#) [2nd](#) [3rd](#) [4th](#)

[XHTML.](#) [See [XHTML DOCTYPEs](#)]

[Document Object Model.](#) [See [DOM](#)]

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[ECMA](#)

[JavaScript](#) [2nd](#)

ECMA (European Computer Manufacturer's Association)

[ISO 8859](#)

[ECMA \(European Computer Manufacturers Association\)](#)

economics

[versus academics](#) [2nd](#)

editors

[visual editors](#) [2nd](#)

[classitis](#)

[WYSIWYG](#)

Eisenberg, David

[DOM](#)

[Spanish-to-English translation](#) [2nd](#)

[Eisenberg, J. David](#)

elements

[block-level elements](#) [2nd](#)

[div.](#) [See [div](#)]

[inline elements](#)

[navigation elements.](#) [See [navigation elements](#)]

[object](#) [2nd](#)

[primenav.](#) [See [primenav](#)]

[SPAN](#)

[structural elements.](#) [See [structural elements](#)]

[using according to meaning](#) [2nd](#) [3rd](#) [4th](#)

[embed tag](#)

[embedded style sheets](#) [2nd](#) [3rd](#)

embedded styles

[moving to external CSS files](#) [2nd](#) [3rd](#) [4th](#)

embedding

Flash content

[document.write](#) [2nd](#) [3rd](#)

[multimedia objects](#) [2nd](#) [3rd](#) [4th](#)

[object failures](#) [2nd](#) [3rd](#)

[while supporting standards](#) [2nd](#)

empty tags

closing

[in XHTML](#) [2nd](#)

[ems](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[users who change font sizes](#) [2nd](#) [3rd](#) [4th](#)

[versus keywords](#) [2nd](#)

[Eric Meyer on CSS \(s/b ital\)](#)

errors

[in hybrid layout](#) [2nd](#)

[in hybrid markup](#) [2nd](#) [3rd](#)

[classes](#) [2nd](#)

[div](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[id](#) [2nd](#)

[table cells](#) [2nd](#) [3rd](#)

in mainstreaming standards

[Wired Digital site](#)

[in markup](#) [2nd](#) [3rd](#)

[ESPN.com](#) [2nd](#)

establishing

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

faddishness

[CSS/XHTML](#) [2nd](#)

[Fahrner method](#) [2nd](#) [3rd](#) [4th](#)

[be nice to Opera rule](#)

[problems with](#) [2nd](#)

[Fahrner, **find first name](#)

[Fahrner, Todd](#)

[font standards](#)

fonts

[points](#)

[image replacement \(FIR\)](#)

[img-free method](#) [2nd](#) [3rd](#) [4th](#)

rollover effects

[non-hover state until cursor hovers over image](#)

[sizing fonts](#) [2nd](#) [3rd](#) [4th](#)

[switching mechanisms](#)

[Favlets](#)

file references

relative file references

[CSS](#)

[file transfers](#) [2nd](#)

[unlimited file transfers](#)

files

[cutting size of](#)

[naming conventions](#)

final markup

[hybrid layout](#) [2nd](#)

finishing touches

[to redesign projects](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[Fireworks](#)

[Flash](#)

[accessibility](#) [2nd](#)

embedding content

[document.write](#) [2nd](#) [3rd](#)

[HTML](#)

Flash 4

[accessibility](#)

Flash 5

[accessibility](#)

Flash MX

[accessibility](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[Flash.](#) [See [Macromedia Flash](#)]

flashing elements

[accessibility](#)

[float](#) [2nd](#)

float bug

[IE6/Windows](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

float property

[CSS](#)

font size keywords

[@import](#)

[font tags](#) [2nd](#)

fonts

[alignment of text](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#) [\[Z\]](#)
[\]](#)

gaps

in images

[Gecko browsers](#) [2nd](#) [3rd](#)

gaps in images

Gecko browsers

[CSS](#) [2nd](#)

[Garbage in, garbage out.](#)

Gecko

[Almost Standards mode](#) [2nd](#)

[difference from IE](#) [2nd](#)

[gaps in images](#) [2nd](#) [3rd](#)

[CSS](#) [2nd](#)

[Standards mode](#) [2nd](#)

standards mode

[baselines and whitespace](#) [2nd](#)

[XHTML DOCTYPE triggers](#) [2nd](#)

[Gecko browsers](#)

[DOCTYPE switching](#)

[GIF format](#)

GIF images

[text](#)

[Gilmore Keyboard Festival](#)

[Glendinning, Porter](#)

goals

[of redesign projects](#) [2nd](#) [3rd](#) [4th](#)

[GoLive](#)

[GoLive.](#) [See [Adobe GoLive](#)]

government sites

[mainstreaming standards](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

grouped selectors

[CSS](#)

guidelines

[for XHTML.](#) [See also rules of, XHTML]

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[h1](#)

hacks

[presentational hacks](#) [2nd](#)

[Happy Cog](#) [2nd](#)

headlines

[font sizes](#)

helping

[fellow designers solve problems](#)

[hide](#)

hiding

content

[DOM](#) [2nd](#) [3rd](#) [4th](#)

content (DOM)

[combining with other techniques](#) [2nd](#) [3rd](#)

[home](#)

Home buttons

[with CSS rollover effects](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

hover

[IE/Windows](#)

[hover effect.](#) [See also [rollover effects](#)]

HTML

[converting tables to CSS](#)

[deprecated elements](#)

[Flash](#)

[versus XML](#) [2nd](#)

HTML Tidy

[quoting attribute values](#)

[HTMLEncode](#)

hybrid layout

browsers

[old browsers](#) [2nd](#)

[content markup](#) [2nd](#)

[errors in](#) [2nd](#)

[final markup](#) [2nd](#)

[navigational markup](#) [2nd](#)

[overview of](#) [2nd](#)

Skip Navigation

[accesskey](#) [2nd](#) [3rd](#)

[Skip Navigation link](#) [2nd](#)

[tables](#)

[summary attribute](#)

[hybrid layouts](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

hybrid markup

[errors in](#) [2nd](#) [3rd](#)

[classes](#) [2nd](#)

[div](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[id](#) [2nd](#)

[table cells](#) [2nd](#) [3rd](#)

[hypertext](#)

[[Team LiB](#)]

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

i3Forum

[font sizes](#)

iCab

[accesskey](#)

id [2nd](#)

[capacities of](#) [2nd](#)

[CSS](#)

[errors in hybrid markup](#) [2nd](#)

[images](#) [2nd](#)

[labels](#)

[rules of](#)

[sticky note theory](#) [2nd](#)

[versus class](#) [2nd](#)

id attribute [2nd](#)

id attributes

[content](#) [2nd](#)

[including or not including](#) [2nd](#)

[nav](#) [2nd](#)

ID names

[font keywords](#)

id selectors

[contextual id selectors](#) [2nd](#) [3rd](#) [4th](#)

IE

[XHTML DOCTYPE triggers](#) [2nd](#)

IE/Windows

fonts

[ignoring fonts option](#) [2nd](#) [3rd](#)

[keywords](#) [2nd](#)

[object failures](#) [2nd](#) [3rd](#)

[pixels](#) [2nd](#)

[pseudo-classes](#) [2nd](#)

[streaming problems](#)

[whitespace bug](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

IE3

[CSS](#) [2nd](#) [3rd](#)

[ems](#)

fonts

[point size](#)

IE4

[behaviors](#) [2nd](#)

[CSS](#)

IE5.x/Windows

[broken box model](#) [2nd](#)

IE5/Mac

[DOCTYPE switching](#)

[font sizes](#)

[Text Zoom](#) [2nd](#) [3rd](#)

IE5/Macintosh

[Text Zoom](#)

IE55/Windows

IE6

[block-level elements](#)

[standards compliant](#)

IE6/Windows

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#) [\[Z\]](#)
[\]](#)

[JavaScript](#) [2nd](#) [3rd](#)

[code to ensure links work without JavaScript](#) [2nd](#) [3rd](#)

[document.write](#) [2nd](#) [3rd](#)

[ECMA](#) [2nd](#)

[versus CSS](#) [2nd](#)

[JavaServer Pages.](#) [\[See JSP\]](#)

[JScript](#) [2nd](#)

[JSP \(JavaServer Pages\)](#)

[Juxt Interactive](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#) [\[Z\]](#)
]

[Kaliber 10000 \(K10k\)](#)

keywords

[sizing fonts](#) [2nd](#)

[@import](#)

[advantages of](#) [2nd](#)

[problems with](#)

[problems with IE/Windows](#) [2nd](#)

[problems with implementing](#) [2nd](#) [3rd](#)

Kmleon

[DOM](#)

[Koch, Peter-Paul](#)

[DOM Compatibility Overview](#)

Konqueror

[DOM](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

labels

[id](#)

[menu labels](#)

languages

[ActiveX](#)

[ASP](#)

[ColdFusion](#)

[JavaScript](#)

[JScript](#)

[JSP](#)

[PHP](#) [2nd](#) [3rd](#) [4th](#)

[XHTML](#). [[See XHTML](#)]

[XML](#). [[See XML](#)][2nd](#) [[See XML](#)][3rd](#) [[See XML](#)]

large

[font keywords](#)

[Latin-1](#)

[Latin-2](#)

laws

[U.S. Section 508](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#)

layout

controlling

[typography](#) [2nd](#)

CSS

[page divisions](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

of tables

[accessibility](#)

layouts

[hybrid layouts](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[liquid layouts](#)

[tables](#) [2nd](#) [3rd](#)

[leading](#). [[See line-height](#)][2nd](#) [[See line-height](#)]

[left-aligned text](#)

[legacy rendering](#) [2nd](#) [3rd](#)

[less than signs \(**insert here**\)](#)

LIFT

[Microsoft FrontPage](#)

[LIFT \(UsableNet\)](#)

[LIFT accessibility tool](#)

limitations

[of Almost Standards mode](#) [2nd](#)

Lindquist, David

[Using Lists for DHTML Menus \(s/b in quotes](#)

[line-height](#) [2nd](#)

[link, visited, hover, active \(LVHA\)](#)

linking

[external style sheets](#) [2nd](#) [3rd](#)

[links](#) [2nd](#) [3rd](#)

[coloring](#) [2nd](#)

[controlling](#)

[placement of](#)

relative links

[DOCTYPE](#)

[rollover effects](#)

sizes

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

Mac users

[fonts](#) [2nd](#)

Macintosh

[IE5/MAC](#)

Macintosh OS X

[browsers for](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

Macromedia

[Macromedia ColdFusion.](#) [See [ColdFusion](#)]

Macromedia Dreamweaver

[standards](#) [2nd](#)

[Macromedia Dreamweaver MX](#)

[XML](#)

[Macromedia Dreamweaver.](#) [See [Dreamweaver](#)]

Macromedia Fireworks

[Macromedia Flash](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#)

[accessibility](#) [2nd](#) [3rd](#)

[problems with](#) [2nd](#) [3rd](#)

[SVG](#)

Macromedia Flash MX

[accessibility](#) [2nd](#) [3rd](#)

mailing lists

[W3C DOM Mailing List](#)

mainstreaming

[CSS](#)

standards

[commercial sites](#) [2nd](#)

[government sites](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[publishing systems](#) [2nd](#)

[Sired Digital site](#) [2nd](#) [3rd](#) [4th](#)

mapping

character sets

[to Unicode](#) [2nd](#)

[margins](#) [2nd](#) [3rd](#) [4th](#)

[box model \(CSS\)](#)

[CSS](#)

[rules-based designs](#) [2nd](#) [3rd](#) [4th](#)

[Marine Center, The](#)

markup

bad markup

[cost of](#) [2nd](#) [3rd](#) [4th](#)

[condensed versus compressed markup](#) [2nd](#)

content markup

[hybrid layout](#) [2nd](#)

[customized markup](#)

[errors in](#) [2nd](#) [3rd](#)

final markup

[hybrid layout](#) [2nd](#)

[multiple versions of](#) [2nd](#) [3rd](#) [4th](#)

navigational markup

[hybrid layout](#) [2nd](#)

outdated markup

[cost of](#) [2nd](#) [3rd](#)

quality of

[long-term effects of sites](#) [2nd](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

namespaces

[DOCTYPE](#)

[XHTML](#) [2nd](#)

naming conventions

[for files](#)

naming files

[conventions for](#)

[nav](#) [2nd](#)

navigation

[Skip Navigation link](#). [See [Skip Navigation link](#)]

skipping

[with accesskey attribute](#) [2nd](#)

navigation bars

[CSS/XHTML](#) [2nd](#) [3rd](#)

[adding styles](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[navigation elements](#)

[borders](#)

[navigational elements](#) [2nd](#)

[controlling links](#)

CSS

[final pass](#) [2nd](#) [3rd](#)

[first pass](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[second pass](#) [2nd](#) [3rd](#)

[links](#)

[rollover effects](#)

[specifying sizes of link effects](#) [2nd](#) [3rd](#)

text

[centering and aligning](#)

navigational markup

[hybrid layout](#) [2nd](#)

[navigational table layouts](#) [2nd](#)

[needless images](#) [2nd](#)

Netscape

[Almost Standards mode](#) [2nd](#)

[DevEdge](#)

[embed tag](#)

[rollovers](#)

[standards compliant](#) [2nd](#)

[Netscape 4](#) [2nd](#)

[behaviors](#) [2nd](#)

[CSS](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[declaring background colors](#)

[DHTML](#)

[DOM](#)

[ems](#)

[inheritance](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[pixels](#) [2nd](#)

[Netscape 6+](#)

Netscape 7

[alternate styles](#)

[New York Public Library](#)

Newhouse, Mark

[Real World Style](#)

non-CSS environments

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#) [\[Z\]](#)

[object element](#) [2nd](#)

[object tags](#)

[objects](#)

[embedding multimedia objects](#) [2nd](#) [3rd](#) [4th](#)

[object failures](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[while supporting standards](#) [2nd](#)

[obsolescence](#)

[ending the cycle of](#) [2nd](#)

[obsolete websites](#) [2nd](#)

[one9ine](#)

[opening](#)

[with DOCTYPE](#)

[Opera](#)

[box model](#)

[DOCTYPE switching](#)

[DOM](#)

[DOM sniff](#) [2nd](#)

[Page Zoom](#) [2nd](#) [3rd](#)

[pixels](#) [2nd](#) [3rd](#)

[Opera 6](#)

[standards compliant](#) [2nd](#)

[Opera 7](#)

[DOM](#)

[font sizes](#)

[order](#)

[of pseudo-classes](#)

[tabbing order](#)

[Bobby Accessibility Validator](#) [2nd](#) [3rd](#) [4th](#)

[order of fonts](#)

[in CSS](#)

[organization](#)

[outlines](#) [2nd](#)

[outdated markup](#)

[cost of](#) [2nd](#) [3rd](#)

[outdated methods](#)

[CSS](#)

[bad CSS](#) [2nd](#) [3rd](#)

[image maps](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[inline stylels](#)

[CSS](#) [2nd](#)

[slicing and dicing](#) [2nd](#) [3rd](#) [4th](#)

[table layouts](#) [2nd](#)

[tables](#)

[redundancies](#) [2nd](#) [3rd](#)

[outlines](#) [2nd](#)

[overlines](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[padding](#)

page divisions

[CSS](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[Page Zoom](#) [2nd](#) [3rd](#)

pages

[redesigning](#) [2nd](#)

[testing in different browsers](#) [2nd](#) [3rd](#) [4th](#)

[Pair.com](#)

[paragraph tags](#)

parameters

[establishing for redesign projects](#) [2nd](#) [3rd](#)

[creating color bars](#) [2nd](#)

[creating content areas](#) [2nd](#)

[installing sidebars](#) [2nd](#)

[positioning sidebars](#) [2nd](#)

percentages

[versus keywords](#) [2nd](#)

perceptions

[of standard compliance](#) [2nd](#)

[Photoshop ImageReady](#)

[PHP](#) [2nd](#) [3rd](#) [4th](#)

[physically impaired.](#) [See [motor impaired](#)]

pixels

[font sizes](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[fonts](#)

[IE/Windows](#) [2nd](#)

[images](#)

[Netscape 4](#) [2nd](#)

[Opera](#) [2nd](#) [3rd](#)

[reference pixels](#) [2nd](#)

[Text Zoom](#) [2nd](#) [3rd](#)

[pixels \(px\)](#)

[PixelSurgeon](#)

placement

[of links](#)

platforms

[fonts](#) [2nd](#) [3rd](#)

[point-driven style sheets](#)

[points \(pts\)](#) [2nd](#)

popularity

[of XML](#) [2nd](#)

positioning

[sidebars](#)

[establishing parameters for redesign projects](#) [2nd](#)

presentation

[of sites](#)

[standards](#) [2nd](#)

[presentational hacks](#) [2nd](#)

[primenav](#) [2nd](#) [3rd](#) [4th](#)

[print style sheets](#) [2nd](#)

problems

[color matching](#) [2nd](#)

[multiple versions of markup and code](#) [2nd](#) [3rd](#) [4th](#)

[with Flash](#) [2nd](#) [3rd](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#) [\[Z\]](#)
]

quality

of markup

[long-term effects on sites](#) [2nd](#)

Quality Assurance group

[W3C](#) [2nd](#)

[QuickTime](#)

[Quirks mode](#)

[IE6/Windows](#)

quotation marks

[CSS multiname fonts](#)

quoting

attribute values

[XHTML](#) [2nd](#) [3rd](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[Raiderettes.com](#) [2nd](#)

[RDF \(Resource Description Framework\)](#)

[Real World Style](#)

reasons

[for acceptance of XML](#) [2nd](#) [3rd](#)

[for choosing XML](#) [2nd](#)

Recommendations

[W3C](#)

[redesign projects](#) [2nd](#)

[alternate styles](#) [2nd](#)

[brand character](#)

[color matching](#) [2nd](#)

[establishing parameters](#) [2nd](#) [3rd](#)

[creating color bars](#) [2nd](#)

[creating content areas](#) [2nd](#)

[installing sidebars](#) [2nd](#)

[positioning sidebars](#) [2nd](#)

[evolution of](#) [2nd](#)

[finishing touches](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[goals of](#) [2nd](#) [3rd](#) [4th](#)

[Home button with CSS rollover effects](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[methods of redesigning](#) [2nd](#) [3rd](#)

[navigation bars](#) [2nd](#) [3rd](#)

[adding styles](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[rules-based designs](#) [2nd](#) [3rd](#) [4th](#)

redesigning

[pages](#) [2nd](#)

redundancies

[tables](#) [2nd](#) [3rd](#)

[reference pixels](#) [2nd](#)

[reformulation](#) [2nd](#) [3rd](#)

relative file references

[CSS](#)

[relative font sizes](#)

[relative image file reference links](#)

relative links

[DOCTYPE](#)

relative values

[pixels](#)

removing

[background images](#)

rendering

[between browsers](#)

[legacy rendering](#) [2nd](#) [3rd](#)

[Resource Description Framework.](#) [See [RDF](#)]

resources

[for accessibility](#) [2nd](#)

[Building Accessible Web Sites \(s/b ital\)](#) [2nd](#)

[Constructing Accessible Web Sites \(s/b ital\)](#) [2nd](#)

[for DOM](#) [2nd](#) [3rd](#)

[RGB](#)

[percentages](#) [2nd](#)

[Rich Site Summary \(RSS\)](#)

rollover effects

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[Safari](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#)

[DOM](#)

[image latency problems](#) [2nd](#) [3rd](#)

saving

[bandwidth](#) [2nd](#) [3rd](#)

[Scalable Vector Graphics \(SVG\)](#)

scripting

[standards](#) [2nd](#)

scripts

[detection scripts](#)

search engines

[mainstreaming standards](#)

selectors

[CSS](#)

[class selectors](#) [2nd](#) [3rd](#)

[combining](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[contextual \(descendant\) selectors](#) [2nd](#) [3rd](#)

[contextual id selectors](#) [2nd](#) [3rd](#) [4th](#)

[grouped selectors](#)

[pseudo-class selectors](#)

selling

[standards](#) [2nd](#)

semicolons

declarations

[CSS](#)

[serif faces](#)

[Server Side Includes \(SSI\)](#)

[server-side image maps](#)

shortcuts

[accesskey](#)

[shorthand](#)

[margins](#) [2nd](#)

showing

content

[DOM](#) [2nd](#) [3rd](#) [4th](#)

content (DOM)

[combining with other techniques](#) [2nd](#) [3rd](#)

[sidebar](#) [2nd](#) [3rd](#)

sidebars

establishing parameters for redesign projects

[installing](#) [2nd](#)

[positioning](#) [2nd](#)

[Simple Object Access Protocol \(SOAP\)](#) [2nd](#)

size

of files

[cutting](#)

sizes

[of fonts](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#)

of link effects

[specifying](#) [2nd](#) [3rd](#)

sizing

[fonts](#)

[effect of nonstandard sizes](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[ems](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

Tab key

tabbing order

[Bobby Accessibility Validator](#) [2nd](#) [3rd](#) [4th](#)

[tabindex attribute](#) [2nd](#)

[table layouts](#)

tables

[accessibility](#) [2nd](#) [3rd](#) [4th](#)

cells

[errors in hybrid markup](#) [2nd](#) [3rd](#)

[hybrid layout](#)

[summary attribute](#)

[layouts](#) [2nd](#)

[redundant tables](#) [2nd](#) [3rd](#)

tabs

[embed](#)

tags

closing

[in XHTML](#) [2nd](#) [3rd](#) [4th](#)

[embed](#)

[font](#)

[font tags](#)

[img](#)

[object](#)

[paragraph](#)

writing in lowercase

[XHTML](#) [2nd](#)

[Tantek method.](#) [See Box Model hack]

[Tasman rendering engine](#)

[TDC \(Type Directors Club\)](#)

[Team Rahal, Inc.](#)

test suites

[for XML](#) [2nd](#)

testing

[Bobby Accessibility Validator](#) [2nd](#) [3rd](#)

[for DOM](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[how it works](#) [2nd](#) [3rd](#)

[pages in browsers](#) [2nd](#) [3rd](#) [4th](#)

Two-Sheet Method

[in noncompliant browsers](#)

testings

[styles in different browsers](#)

[Texas Parks & Wildlife](#)

text

[alignment of](#)

[avoiding wall-to-wall text](#) [2nd](#) [3rd](#)

[centering and aligning](#)

[GIF images](#)

sizing

[CSS and accessibility](#)

[Text Zoom](#)

[IE5/Mac](#) [2nd](#) [3rd](#)

[pixels](#) [2nd](#) [3rd](#)

[text-decoration property](#)

time

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#) [\[Z\]](#)

[U.S Section 508 of Workforce Investment Act](#)

[U.S. Section 508](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[underlines](#)

[Unicode](#)

[mapping character sets to](#) [2nd](#)

[unmetered file transfers](#)

[updating](#)

[pages](#) [2nd](#)

[upgrading](#)

[browsers](#)

[URLEncodedFormat\(\)](#)

[usability.](#) [\[See](#) [accessibility\]](#)

[UsableNet](#)

[LIFT](#)

[FrontPage](#)

[UseableNet](#)

[LIFT](#)

[UserLand Software](#)

[XML-RPC](#)

[users](#)

[blocking](#)

[by designing for only one browser](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[changing font sizes](#)

[ems](#) [2nd](#) [3rd](#) [4th](#)

[control over fonts](#)

[excluding](#)

[who change font sizes](#) [2nd](#)

[Fahrner method](#) [2nd](#)

[Using Lists for DHTML Menus \(s/b in quotes\)](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#) [\[Z\]](#)
[\]](#)

validation test results

[trusting](#)

[Validator](#) [2nd](#)

values

CSS

[clock faces](#)

declarations

CSS

[Veen, Jeffrey](#)

versions

[cost of writing for multiple](#) [2nd](#) [3rd](#)

video

[streaming video media](#)

[visual editors](#) [2nd](#)

[classitis](#)

[visual element](#)

[visual elements](#)

[visually impaired](#) [2nd](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[W3C](#)

[CSS2](#) [2nd](#) [3rd](#)

[DOM](#)

[embedding multimedia objects](#) [2nd](#)

[Markup Validation Service](#) [2nd](#)

[Quality Assurance group](#) [2nd](#)

[W3C \(World Wide Web Consortium\)](#)

[W3C DOM Mailing List](#)

[W3C validator](#)

[detecting errors](#)

[WAI \(Web Accessibility Initiative\)](#) [2nd](#)

[WaSP](#)

[DOM](#)

[websites](#)

[WaSP \(Web Standards Project\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[WaSP Dreamweaver Task Force](#)

[Watchfire](#)

[Bobby](#)

[Bobby Accessibility Validator](#) [2nd](#)

[building and testing](#) [2nd](#) [3rd](#)

[checklists](#) [2nd](#)

[one page, two designs](#)

[tabbing order](#) [2nd](#) [3rd](#) [4th](#)

[tabindex attribute](#) [2nd](#)

[web](#)

[coming of age](#) [2nd](#)

[Web Accessibility Initiative.](#) [[See WAI](#)][2nd](#) [[See WAI](#)]

[web authoring](#)

[FrontPage](#) [2nd](#)

[web pages](#)

[DOM](#)

[making web pages behave like applications](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[web professionals](#)

[misguided opinions about accessibility](#) [2nd](#) [3rd](#)

[web publishing tools](#) [2nd](#) [3rd](#)

[Web Services](#)

[XML](#) [2nd](#)

[web standards](#)

[Web Standards Project](#)

[Browser Upgrade campaign](#) [2nd](#)

[benefits of](#) [2nd](#) [3rd](#)

[bouncing visitors away from sites](#) [2nd](#)

[Web Standards Project's Dreamweaver Task Force](#) [2nd](#) [3rd](#)

[Web Standards Project.](#) [[See WaSP](#)][2nd](#) [[See WaSP](#)][3rd](#) [[See WaSP](#)][4th](#) [[See WaSP](#)][5th](#) [[See WaSP](#)]

[web standards.](#) [[See standards](#)]

[websites](#)

[Adobe](#) [2nd](#)

[Blue Robot's Layout Reservoir](#)

[commercial sites](#)

[mainstreaming standards](#) [2nd](#)

[Communication Arts Interactive Awards](#)

[Connected Earth](#) [2nd](#)

[cost of upkeep](#) [2nd](#) [3rd](#) [4th](#)

[ECMA](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

x-large

[font keywords](#)

x-small

[font keywords](#)

[XHTML](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[accessibility](#)

[bgcolor attribute](#)

[choosing the right one for you](#) [2nd](#) [3rd](#)

[CSS](#) [2nd](#)

DOCTYPE

[namespaces](#) [2nd](#)

[DTDs](#)

elements

[div.](#) [See [div](#)]

embedding Flash content

[with document.write](#) [2nd](#) [3rd](#)

[guidelines for](#)

links

[block-level elements](#)

[markup](#) [2nd](#)

[navigation bars](#) [2nd](#) [3rd](#)

[adding styles](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[outlines](#) [2nd](#)

[reasons for converting to](#) [2nd](#) [3rd](#)

[reasons for not converting to](#)

[rules of](#)

[ampersands and less than signs](#) [2nd](#)

[attribute values](#) [2nd](#)

[closing empty tags](#) [2nd](#)

[closing tags](#) [2nd](#)

[declaring content type](#)

[double dashes within comments](#)

[opening with DOCTYPE](#)

[quoting attribute values](#) [2nd](#) [3rd](#)

[writing tags in lowercase](#) [2nd](#)

[tabindex attribute](#) [2nd](#)

XHTML

[visual elements](#)

XHTML

[XML prologs](#) [2nd](#) [3rd](#) [4th](#)

[XHTML \(Extensible Hypertext Markup Language\)](#)

XHTML 1 Transitional

[DOM](#)

XHTML 1.0 DTD

[XHTML 1.0 Frameset](#)

[XHTML 1.0 Strict](#)

[XHTML 1.0 Transitional](#)

[XHTML 1.0 Frameset](#)

[XHTML 1.0 Strict](#)

[XHTML 1.0 Transitional](#) [2nd](#)

[DOCTYPE](#)

[XHTML 1.1 \(Strict by definition\)](#)

XHTML 1.1 DTD

[XHTML 1.1 \(Strict by definition\)](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#) [\[Z\]](#)
]

[Yahoo](#)

[you are here \(s/b in quotes\) effect](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#) [\[Z\]](#)
]

[zeldman.com](#)

[image gaps](#) [2nd](#)

zero

[percentages in CSS](#) [2nd](#)

[\[Team LiB \]](#)