Rick Darnell, et al.

HTML 4

TIMELY SOLUTIONS

■ HTML 4 ■ Dynamic HTML ■ Cascading Style Sheets ■ Netscape Communicator 4 ■ Internet Explorer 4 ■ JavaScript Style Sheets ■ HTML 3.2 Syntax ■ Frames ■ Tables ■ Forms ■ Hyperlinking ■ Layers ■ Internationalization ■ Extensible Markup Language (XML) ■ Image Maps ■ Interface and Navigation Design ■ CGI ■ Java and JavaScript ■ ActiveX and VBScript ■ Databases ■ Web Servers

EXPERT SOFTWARE

NetObjects™ Fusion 2 demo ■ Microsoft® Internet Explorer™ 3 ■ HotDog ■ BBEdit ■ HomeSite free ■ Cel Assembler ■ ActiveX Control Pad and HTML Layout Control ■ Web graphics ■ Examples and source code

BONUS!

ELECTRONIC LIBRARY

Two books fast-linked in HTML format:

- *Java 1.1 Unleashed, Third Edition*
- *Laura Lemay's Web Workshop: ActiveX and VBScript*

sams net

THE COMPREHENSIVE SOLUTIONS PACKAGE!

UNLEASHED

HTML Unleashed Quick Reference

Explore the Foundations of HTML

- Learn the origins of Hypertext Markup, 10
- Discover the history of HTML, 18
- Understand the role of the World Wide Web Consortium, 32
- Analyze the HTML 4.0 Document Type definition, 55
- Review the structure of HTML markup, 72
- Examine URLs in detail, 102
- Learn system and platform display differences, 113

Learn the Fundamentals of HTML

- Learn how to use the HEAD element and its attributes, 140
- Examine the BODY element and its attributes, 148
- Review basic text formatting, 158
- Discover the power of lists, 189
- Build a table, 204
- Add hyperlinks with the Anchor tag, 223
- Include images in your pages, 234
- Incorporate external multimedia content, 249
- Create and use image maps, 268
- Build an HTML form, 276

Discover HTML 4.0

- Preview HTML 4.0, 320
- Create frames, 344
- Discover how to use layers, 350
- Learn the essentials of style sheets, 366
- Create a business Web page using style sheets, 386
- Explore JavaScript style sheets, 400
- Use events to create Dynamic HTML, 415

Review Web Page Design Techniques

- Brainstorm your site theme, 439
- Use the appropriate layout elements, 453
- Discover different navigation systems, 473
- Examine a corporate Internet site, 487
- Review an e-zine, 498

Integrate Other Technologies with HTML

- Discover what CGI programs do on your Web site, 514
- Create JavaScripts, 547
- Learn how to use Java applets, 574
- Add ActiveX controls to your HTML documents, 605
- Add another dimension with VRML, 644
- Integrate plug-ins into your site, 654

Explore the Features of Web Development Tools

- Explore the basic HTML editors, 692
- Choose a graphics package, 711
- Review advanced Web authoring tools, 729
- Learn how to choose the best Web server, 772

Peer Into the Future of HTML

- Discover the premises of XML, 782
- Learn how to internationalize your HTML, 807
- Debate the pros and cons of Pure HTML, 815
- Is HTML the new interface?, 829

HTML 4

UNLEASHED

Rick Darnell, et al.

sams net

201 West 103rd Street
Indianapolis, IN 46290

International Standard Book Number: 1-57521-299-4

Library of Congress Catalog Card Number: 97-65136

2000 99 98 97 4 3 2

Interpretation of the printing code: the rightmost double-digit number is the year of the book's printing; the rightmost single-digit, the number of the book's printing. For example, a printing code of 96-1 shows that the first printing of the book occurred in 1996.

Composed in AGaramond and MCPdigital by Macmillan Computer Publishing

Printed in the United States of America

President, Sams Publishing *Richard K. Swadley*
Publishing Manager *Mark Taber*
Acquisitions Manager *Beverly M. Eppink*
Director of Editorial Services *Cindy Morrow*
Managing Editor *Brice Gosnell*
Director of Marketing *Kelli Spencer*
Product Marketing Manager *Wendy Gilbride*
Assistant Marketing Managers *Jen Pock*, *Rachel Wolfe*

Acquisitions Editor
David B. Mayhew

Development Editor
Bob Correll

Software Development Specialist
Bob Correll

Production Editor
Ryan Rader

Copy Editors
Kristen Ivanetich
Carolyn Linn

Indexers
Christine Nelsen
Erika Millen
Bruce Clingaman

Technical Reviewers
Karen Clere
Blake Hall

Editorial Coordinators
Mandi Rowell
Katie Wise

Technical Edit Coordinator
Lorraine Schaffer

Resource Coordinator
Deborah Frisby

Editorial Assistants
Carol Ackerman
Andi Richter
Rhonda Tinch-Mize

Cover Designer
Jason Grisham

Cover Production
Aren Howell

Book Designer
Gary Adair

Copy Writer
David Reichwein

Production Team Supervisor
Brad Chinn

Production
Mona Brown
Jennifer Dierdorff
Shawn Ring
Janet Seib

Overview

Contents

Dedication

To Jane, Margaret, and Elizabeth. —Rick Darnell

To my brother, Steve; his wife, JoAnn; and their two daughters, Stephanie and Carrie, for consistently helping me keep two feet anchored to this planet. —Michael Larson

To my wife, Alina, and daughter, Sonya. —Dmitry Kirsanov

Acknowledgments

It's a Saturday night and I'm listening to the News from Lake Wobegon piped through a sound card on my computer. It seems somewhat ironic that I'm listening to stories about vegetable gardens from the "little town that time forgot and can't improve" while I'm finishing a book about the latest in technology standards from the Information Age. Yet, here it is. Words on paper about an electronic medium.

It's a contradiction reminding us that, although the Internet is an amazing set of technologies that enables us to communicate faster than ever before, there's still a real world outside of the wires. Cables can't pull weeds out of the garden, shovel snow from the driveway, or put a daughter to bed. The Internet is only technology. Life is what happens on the outside of the computer, not inside it.

For the last several years, while I've lived my life, I've also had this thing I call a career in writing. There are two really neat aspects about writing. The first is seeing your name in print. It's probably vain and self-serving, but I get great satisfaction from seeing my name attached to a tome such as this.

Second is the people you meet along the way, who make the whole process possible and enjoyable. Writers are pretty pointless without editors, and vice versa. It's one of those dichotomies that results in a certain tension, but also holds incredible satisfaction when both sides reach the end of the process and another book is born.

On this project, David Mayhew was the guiding hand that assembled the components, hired the writers, and made sure we delivered the goods. From what I can see, an Acquisitions Editor has a thankless job and doesn't get nearly the glory we writers do, but David certainly puts in every bit as much work. I'm not really sure why he does it, but he doesn't mind laughing and will even tolerate mindless tangents when we should be talking business.

Although not directly related to this project, Her Majesty Beverly Eppink also deserves a great deal of gratitude. She was generous and supportive as an Acquisitions Editor, and she appears to have retained those qualities during her recent promotion to Acquisitions Manager. She's certainly encouraged my development with editors at Sams.net and is responsible for a great deal of my success.

Many other people on the payroll at Macmillan make these projects happen, from the Publisher and lawyers, to technical and copy editors, production editors, designers, lawyers, secretaries, custodians, and many others. Although I don't know everyone's name, everyone makes a contribution to the finished product. Many thanks to you all.

Then, there are the other authors who participated in this project. I got to be lead author this time, but it wouldn't have been a book without the contributions of the others. It's good to be in the company of this many people working toward the same goal, even though we're in different states and on different continents.

I also extend thanks to Capt. Jess Mickelson of the Missoula (Montana) Rural Fire District, the head of our regional hazardous materials team. He's certainly been supportive and understanding when I've had to miss training for deadlines, and he's been quick to offer opportunities to gain additional experience through conferences and other assignments so that I can maintain my status as a HazMat Technician.

And then there are the many people who have fostered me along the way, giving me the needed guidance so that I could learn how to put the right words in the right order. This list includes Bonnie Montgall, David MacFarland, John Braden, and influences such as Edward Abbey, Kenneth Grahame, and A.A. Milne.

Last in this list of acknowledgments, but certainly not least, is you, the reader. Your investment in this book just made my bank account a few cents bigger. (Royalties aren't all they're cracked up to be.) I hope you find the book useful and worth the money you shelled out for it. If it makes your job easier or gives you new ideas for communicating with your fellow humans, then we've done our jobs on this end. Thank you.

—Rick Darnell

I would like to thank everyone who appreciates the massive potential for enrichment represented by the WWW and who will settle for nothing less than the most up-to-date HTML reference.

I also want to thank the many folks on the Sams.net editorial and publication staff for their direction, comments, dedication, and support. I especially want to acknowledge the primary editors for this compendium, Dave Mayhew and Bob Correll, for their tenacity in continually focusing the content of this book to optimize its usefulness and ensure that we are as far out on the cutting-edge of these topics as possible. Great job, guys.

—Michael Larson

I am thankful to Andrey A. Chernov (`ache@nagual.pp.ru`) for useful comments on the text of Chapter 39, "Internationalizing the HTML Character Set and Language Tags."

—Dmitry Kirsanov

About the Authors

Lead Author

Rick Darnell (`darnell@montana.com`) is a flatlander by nature, currently living with his wife and two daughters in the middle of a bunch of mountains in Montana. He began his print career at a small weekly newspaper after graduating from Kansas State University with a degree in broadcasting. While spending time as a freelance journalist and writer, Rick has seen the full gamut of personal computers since starting out with a Radio Shack Model I in the late 1970s. When not in front of his computer, he serves as a volunteer firefighter and member of a regional hazardous materials response team. Rick has authored several books for Sams.net Publishing and for Que. For this book, he wrote Part II, "Basic HTML," as well as Chapters 28, 29, 34, and 36.

Contributing Authors

Dennis Báthory-Kitsz is a composer, performance artist, author, technologist, and teacher. His early writing about computers promoted open architectures and experimentation, and today he concentrates on uses of technology in business and performing arts. His music has been performed in North America and Europe, and he co-hosts the new-music radio show "Kalvos & Damian's New Music Bazaar," which is heard in New England and on the Internet. Malted/Media (`http://www.maltedmedia.com/`) is his home page. Dennis compiled the glossary of this book.

J. Gregory Bryan has worked for the past 16 years as a consulting scientist, manager, and technical writer in the energy exploration, environmental, and manufacturing industries. Throughout his background of business and technical experience, Gregory has also remained active in the digital processing, analysis, and interpretation of diverse integrated data. Fascinated by finding new methods of effective communication, Gregory currently works as a technical writer and consultant in South Carolina, where he is honing a new set of skills relating to digital video production and the Internet. He can be contacted by e-mail at `GBRYAN1207@aol.com`. Gregory wrote Chapter 37 of this book.

Bruce Campbell (`bdc@hitl.washington.edu`) lives in Seattle, WA and works with technologies related to 3D collaboration, such as VRML (Virtual Reality Modeling Language). He has contributed to three books for Sams.net Publishing: *Teach Yourself VRML 2.0 in 21 Days, Maximum Java 1.1*, and *Java 1.1 Unleashed, 3rd Edition*. When he is not writing, he is either teaching for Catapult training centers, performing VR-related research at the Human Interface Technology Laboratory at the University of Washington, or running around somewhere in North America. He can be found on the Web at `http://www.hitl.washington.edu/people/bdc`. Bruce wrote Chapters 22 and 31.

Bob Correll (bcorrell@sams.mcp.com) graduated from the Air Force Academy with a B.S. in History and served as an Intelligence Officer for more than seven years. After leaving the military, he delivered pizzas and helped in a warehouse for a short time before signing on with Sams.net Publishing as a Software Specialist. Now serving as a Development Editor, Bob discovered the joys and frustrations of HTML with the help of Laura Lemay's first two books, ironically enough, published by the same team within Sams.net of which he is now a part. Bob enjoys using both sides of his brain (in theory, at least) and also expresses his creativity by making bowls, vases, and yo-yos by wood turning. Bob wrote Chapter 17 of this book.

Eric Herrmann is the principal of Practical Internet, an Austin-based consultancy that develops Web sites and provides software support for businesses and other Web page developers. He has been a programmer in the defense industry for 10 years, specializing in TCP/IP and asynchronous parallel processing. Eric wrote Chapter 27.

Molly E. Holzschlag—author of *Laura Lemay's Web Workshop: Designing with Stylesheets, Tables, and Frames* and *Laura Lemay's Guide to Sizzling Web Site Design*—is a writer, instructor, and Web designer. Her Web design books and columns look at Web design holistically, examining the technical, artistic, commercial, and personal issues involved in this emerging field. She holds an M.A. in Media Studies from the New School for Social Research. She teaches Web design to content providers on the Microsoft Network, and she holds seminars and classes on a regular basis. For more information, visit Molly at http://www.molly.com/. Molly wrote chapters 19 and 20 of this book.

John Jung provides technical support for a wide variety of operating systems at his day job. He enjoys reading books, playing video games, and watching TV. In his spare time, he goofs off and write books. He's worked on almost a dozen books in various different capacities. John wrote Chapters 18 and 40.

Will Kelly (willk@tiac.net) is a freelance technical writer and consultant in the Washington, DC area, specializing in the Internet, networking, and emerging technologies. He has written technical documentation and courseware in support of projects for major government agencies and Fortune 1000 corporations. He has a B.S. in English from Frostburg State University. Will steps away from his normal role as technical editor for some Sams.net books to contribute Chapter 21 to this book.

Dmitry Kirsanov graduated in 1996 from St. Petersburg (Russia) University, Faculty of Languages. He worked first as a translator, then pursued the career of freelance writer, and published two books and numerous magazine articles on the Internet and the Web, all in Russian. Recently, he started to write for the Web in English by opening the *Design Lab* monthly column at http://www.webreference.com/dlab/. Dmitry wrote Chapters 3, 38, and 39.

Michael Larson has authored one previous book (*Teach Yourself Web Publishing with Office 97 in a Week*), several technical publications, and numerous technical reports for many audience types.

He has two technical degrees—a Bachelor's degree from Rocky Mountain College in Billings, Montana and a Master's degree from Montana State University in Bozeman, Montana. He can be reached by e-mail at larsonm@ix.netcom.com. His home page is at http://www.webcom.com/larsonm/. Michael's interests are reading, graphics (he is a certified Photoshop addict), Web page building, and hiking. He is currently enjoying a sunshine-filled life in Utah. For this book, Michael authored Part IV, "Effective Web Page Design Using HTML," and Chapters 33 and 35.

Paul Lomax is the technical director of Mentor Web (http://www.mentorweb.net/), a leading Web design and hosting company. He has been a programmer for more than 12 years and has been a dedicated fan of Visual Basic since version 1. Paul is the author of *Laura Lemay's Web Workshop: ActiveX and VBScript*, and he wrote Chapter 30 of this book.

Dick Oliver (dicko@netletter.com and http://netletter.com) is the author and co-author of numerous books on computer graphics and the Internet, including *Teach Yourself HTML in 24 Hours*, *Creating Your Own Web Graphics*, *Web Page Wizardry*, *Netscape Unleashed*, and *Web Publishing Unleashed*. He is also the president of Cedar Software and the publisher of a paper and online newsletter called *The Nonlinear Nonsense Netletter*. Dick lives in Elmore, Vermont, and commutes to work all over the world via the Internet. He wrote Chapter 41 and Appendix A of this book.

Michael Sessums (msessums@fastlane.net) is a Web site developer and publications editor for Lockheed Martin Tactical Aircraft Systems, and he is a charter member of the Lockheed Martin Web Authors Guild. Michael designs logos and Web sites for small businesses. He also conducts Internet training and maintains a community service site called "Dart's Pet Rescue and Adoption Page" (www.startext.net/homes/petpage). His wife, Brenda, keeps him happy with the latest free software upgrades and an endless supply of cats. Michael wrote Chapters 6 and 32.

Phil Stripling (philip@cieux.com) is a lawyer who lives and practices in the San Francisco Bay Area, near the intersection of Law and Internet. With an undergraduate degree in speech/drama, he is living proof that you don't need a degree in computer science to write HTML and that the originators of the World Wide Web got it right. Phil authored Chapters 1, 2, 4, and 5.

Tell Us What You Think!

As a reader, you are the most important critic and commentator of our books. 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. You can help us make strong books that meet your needs and give you the computer guidance you require.

Do you have access to CompuServe or the World Wide Web? Then check out our CompuServe forum by typing **GO SAMS** at any prompt. If you prefer the World Wide Web, check out our site at `http://www.mcp.com`.

NOTE

If you have a technical question about this book, call the technical support line at 317-581-4669.

As the team leader of the group that created this book, I welcome your comments. You can fax, e-mail, 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. Here's the information:

FAX: 317-581-4669

E-mail: `newtech_mgr@sams.mcp.com`

Mail: Mark Taber
Sams Publishing
201 W. 103rd Street
Indianapolis, IN 46290

Introduction

The Internet has progressed at an amazing pace in recent months and years. Once the realm of academics and defense agencies, the Internet is rapidly becoming a mainstream media conduit for communication between individuals, companies, and global dwellers.

As part of the Internet, the World Wide Web is now the predominant force in growth. Its language is simple, its interface is attractive and friendly, and it is adaptable to a wide variety of uses.

There are now Web sites for selling products, selling ideas, maintaining appearances, informing publics, continuing education and knowledge, and just plain wasting time. And, in a growing trend, the Internet concept is being adapted to internal communications by establishing intranets inside companies.

HTML (Hypertext Markup Language) is the language that puts the face on the Web. It consists of a variety of elements called *tags*, which are used for everything from defining type styles and headings to inserting specialized content such as images, sounds, virtual reality worlds, and Java applets.

HTML is intimidating for many people simply because it includes language in its definition and because it has to do with the Internet. This doesn't need to be the case. HTML is relatively simple to learn, much like the old markup codes for WordPerfect and other word processors prior to WYSIWIG (What-you-see-is-what-you-get) editing.

This leads to the other drawback to working with HTML. Most editors don't display a Web page the way it will appear on a browser. Instead, HTML authors and designers must contend with the content of their page intermixed with the tags that control how the content appears to the user. The good news is that this situation is changing rapidly, with the advent of new WYSIWYG editors, which display a page using the standard accepted by most browsers.

So why learn HTML? There are several reasons. First, it only *looks* complicated. When you start working your way through the tags and using the various elements to build Web pages, you'll discover that HTML is as much an organizational tool as a design tool. The tags give structure and purpose to each part of the page and explain how it relates to the rest of the page.

Second, even some of the best WYSIWYG editors don't support all of the various tags that are a part of HTML at any given time. Sometimes, it's necessary to directly modify the source of the page to add or change tags and attributes. To do this, you need to know how the tags relate to each other.

The final reason for learning HTML is simply for the fun of it. You gain a certain satisfaction from building a Web page from the ground up. You'll know about every brick and board that went into it, and you'll have the know-how to tweak each one so that the result is just what you

wanted. On the other side of that is the fun of seeing other Web pages and knowing how the page author and designer worked to develop the intended effect.

HTML AS GENERAL CONVERSATION

The World Wide Web is becoming more and more common as a topic of conversation at the dinner table, at cocktail parties and banquets, in car pools, and around the water cooler at work. After reading this book, we're sure you'll discover HTML is one of the most fascinating things on the planet and will want to discuss the nuances of each tag with those around you.

Don't do it. It's more than enough in general company to know what *HTML* means. If you try to discuss HTML in any detail, you'll only receive blank stares and suddenly find yourself standing in a corner by yourself holding a small plate of vegetables and runny ranch dressing with the host's dog hanging on your every word.

How This Book Is Organized

This book is divided into seven different sections, plus five appendixes and a glossary. The book begins with basic concepts and foundations of HTML and then delves into the details of standard tags and their attributes, advanced features, and extensions. After the details of the language are covered, additional related topics are covered, including design, supporting technologies, tools, and a crystal ball (in Part VII, "The Future of HTML") for taking a look at what the future holds for new incarnations.

Here's a section-by-section look at what's in store in each Part of the book:

I. **Understanding HTML:** The chapters in this section cover the foundation and history of HTML, beginning with the concept of hypertext and extending into document types, document structure, and how different browsers and platforms fit into the Web picture.

II. **Basic HTML:** This section covers the standard tags supported by the latest HTML 3.2 specification and a peek into upcoming standards, including widely supported features such as text alignment and formatting, images, and anchors. It also shows how HTML supports advanced content such as virtual reality and sound, and how to build forms.

III. **Extending HTML 3.2:** The latest version of HTML includes advanced features that extend the power an author or designer can use to build a page. Frames and layers, style sheets, and dynamic HTML are all explained in these chapters, which give you a handle on some of the newest technology available for Web pages.

IV. **Effective Web Page Design Using HTML:** With the tools and bricks in place from sections II and III, you have the requisite technical knowledge to build Web pages with HTML 3.2. However, a technically correct page is not necessarily a user-friendly page. This section shows you how to build pages that are attractive and usable by including concepts and examples for designing user interfaces and navigation tools.

V. **Associated Technologies and Programming Languages:** HTML's capabilities are extended a great deal through the support of technologies that add interactive features to otherwise lifeless pages. This includes Common Gateway Interface (CGI) scripts for data transfer, Java and JavaScript, ActiveX, and Visual Basic Script. Other extensions to HTML are also included, such as the Virtual Reality Modeling Language (VRML) and browser plug-ins.

VI. **Development and Site Tools:** In the "old days" of the Web, building HTML pages typically involved typing the content and tags into a text editor. It worked, but it wasn't terribly easy to revise or troubleshoot for problems. This situation has changed dramatically with the introduction of a wide variety of programs to create, change, and validate Web pages. This section includes an overview of the best and brightest tools, including HTML editors and Web management tools, site management utilities, Web servers, and graphics programs.

VII. **The Future of HTML:** This section is where we dust off the crystal ball to see what lies in store for HTML in the future. The chapters in this section discuss new languages to augment or replace HTML, along with the evolution of standards for character sets and page representation in an international arena. Issues of standardization among browsers and platforms, and HTML's use in non-Web environments are also covered.

The appendixes in the back of the book include more reference information to help you with your day-to-day work using HTML 3.2. In addition to a quick reference of HTML tags, the appendixes also include the complete HTML 3.2 language and extensions specification, lists of color and character values, and online resources for more information.

We hope the combination of information in the chapters and references in the appendixes makes *HTML Unleashed* the most complete reference you have in your collection.

How to Use This Book

Whether you're an old hand or a newbie, Chapters 1 through 6 are a good introduction and explanation of why HTML looks and acts the way it does, and how it works within the framework of the Internet and intranets.

Chapters 7 through 16 are the meat and potatoes of HTML 3.2. Every tag in the HTML 3.2 specification is covered in this area, along with their various attributes and behaviors. If you're a beginner, working through this section will give you a solid foundation for working with HTML.

Chapters 7, 11, and 12 will quickly get you up to speed in building Web pages with examples of good page structure and the two most-used and most-useful tags you'll see on any Web page. If you've already worked with the language, you should review this area to see what's new and how HTML elements are supposed to behave, including new attributes for text alignment and lists.

For advanced users, Chapters 17 through 22 show how to extend HTML beyond the elements illustrated in the previous chapters. Advanced concepts, including frames, layers, and style sheets are explained and illustrated here.

Web page design is useful for anyone who might need to build a page or entire Web site. Chapters 23 through 26 provide good information about designing pages to work with users, rather than pages that frustrate users. It's good reading for anyone, no matter how you use HTML.

An introduction to other technologies utilized within HTML is covered in Chapters 27 through 33. These chapters aren't meant to be a comprehensive reference or tutorial on working with the various languages, such as CGI, Java, or VBScript. But they do provide a good overview of each technology or language, where it's used, and how it fits within HTML.

If you don't already have a favorite program for working with Web pages, check out Chapters 34 through 37. These chapters include overviews of HTML editors such as HotDog, Hot Metal, and HomeSite, along with complete suites such as FrontPage and Backstage. You'll also find information on utilities to validate HTML pages and hyperlinks, along with Web servers and programs for creating Web graphics.

The last section is another good place to go regardless of your experience with HTML. It's where we take a step into the future to see where this language is going and consider some of the issues that it faces in order to grow. All predictions are presented with a special guarantee: If any turn out to be false, we won't say, "Told you so."

Conventions Used in This Book

Sams.net has spent many years developing and publishing computer books designed for ease of use and containing the most up-to-date information available. With that experience, we've learned what features help you the most. Look for these features throughout the book to help enhance your learning experience and get the most out of HTML.

- Screen messages, code listings, and command samples appear in `monospace type`.
- Uniform Resource Locators (URLs) used to identify pages on the Web and values for HTML attributes appear in `monospace type`.

TIP

Tips present short advice on quick or overlooked procedures. This also includes shortcuts.

NOTE

Notes present useful or interesting information that isn't necessarily essential to the current discussion but might help you understand with background information or advice that relates to the topic.

CAUTION

Cautions warn you about potential problems that a procedure may cause, unexpected results, or mistakes that may prove costly.

Who Should Read This Book?

This book has been planned and designed to fill a wide variety of needs, depending on your level of experience and knowledge with HTML.

For beginners, we offer an introduction into the basics of HTML, including basic page structure and all of the tags needed to build the page. Each HTML element is presented with its corresponding attributes, along with its default behavior and the minimum information it needs to function.

Casual and accomplished users will probably find it easier to jump around to the specific topics they need, such as tables, frames, or design. Remember that this book is a comprehensive resource for HTML 3.2, so you'll still want to glance at the other chapters to see what else you might be missing.

For experts, this book serves as an excellent reference to answer specific questions. The syntax, attribute listings, and examples provide plenty of opportunity to see variations on HTML implementation. The references at the back of the book put the technical information you require within easy reach.

We've worked hard to put together the most comprehensive HTML book available, and we hope you'll agree that it's not only an important addition to your collection, but also a valuable tool you'll use every day.

From all of the authors, thank you for choosing *HTML Unleashed.*

—Rick Darnell
(darnell@montana.com)

PART I

Understanding HTML

IN THIS PART

CHAPTER 1

Hypertext Markup in Theory and Practice

by Philip Stripling

IN THIS CHAPTER

Although many of us think of Hypertext Markup Language as the tags that are put into a document to enable the document to be displayed on the World Wide Web, the concepts behind HTML have a long history. This chapter presents a look at some of the people and ideas that led to our present world of animation, sounds, and applets, considering why other attempts at nonlinear information systems might have failed while the World Wide Web has succeeded beyond its original purpose. I explore briefly the concept of a nonlinear information system—which is the basis of HTML and, perhaps, the key reason for its success.

Webmasters and authors who keep the theories of nonlinear information systems in mind can develop a plan that allows for uniform development of the site and of each page within it. An understanding of the theories of nonlinear information will provide a basis for preparing a Web site that is functional and fully accessible to a worldwide audience.

The Theory of Nonlinear Information

Human beings have always used linear information to communicate and think. Linear information systems have led the world to its present condition; whether that is reason for change or celebration is left for the reader to decide. Because we use, and have always used, linear information in our daily lives, what would we gain by having a nonlinear information system?

Although unrecognized, nonlinear information has always been with us as well, and it often works better than linear information. We have attempted to create nonlinear systems within our linear worlds by using the materials at hand; for example, books and magazines have tables of contents so that we are not required to start at page 1 and read consecutive pages until we get to what we're interested in. (Did you start with page 1 of this book?) Books have indices so that we can find related topics quickly, regardless of their location in the book. Editors often suggest the use of sidebars, margin notes, and tables to present different sets of ideas simultaneously, allowing readers to compare and contrast information quickly. Computers once used magnetic tape to store information, and it was necessary to spin through an entire tape if the desired data was at the end. Computer users quickly saw the need for quicker access, and the disk drive was born, allowing access to any data directly rather than having to step through every sector on the disk to get to the data.

The problem is that although linear information systems worked very well for much of our history, there are some areas today where linear information systems break down, overwhelmed by linear information itself. Imagine a world where all data is stored on spools of microfilm. To obtain information, you must determine which spool contains the desired data, locate and retrieve the spool, and run through the frames until you find the information you seek. It may well be that the index to the spools and the spools themselves are not in your building, so in addition to the time spent looking for the data, you must also spend time and energy getting to the data because, in the world of paper, it does not come to you. This is not an imaginary world, by the way. Title companies use contractors to search recorded deeds, mortgages, and easements before issuing title policies. Not only must the searchers locate the microfiche, but they

also must travel to different locations, often in different towns, for a full title search—the local courthouse, the county clerk's office, and the secretary of state. A visit to the office of the recorder of deeds will show you a nearly Dickensian world of people poring over huge index books filled in by hand, prowling through stacks of books and file cabinets, and sitting before hooded microfiche machines, notepads in hand.

Nonlinear information systems address many of the problems of information flow. They provide an exponential gain over linear systems used in research. In a typical law firm in 1985, a large room was devoted to the law library. Walls of volumes contained reported cases, statutes, regulations, and ordinances. Several companies provided loose-leaf binders with commentary on specialized areas of the laws, with daily, weekly, and monthly updates keeping the librarian always behind in posting the changes. Law clerks sat at carrels with stacks of books, taking notes and making photocopies for lawyers to use in preparing briefs and memoranda. Again, the researcher went to the resource and transcribed the information. A secretary would have used a computer to type and print the manuscript, but no other person would have been able to use a computer for research or for maintenance of the materials in the library.

NOTE

Our lawyer in this example is limited in her resources to the material provided by her firm on its server. For the purposes of the lawyer and her firm, this is not likely a limitation. For others, however, this limitation could be as inhibiting as having to rely on paper-based information systems.

Today it is likely that the library is on CD-ROMs on a server in a closet. The lawyer has a computer on her desk, connected to the server. Research is accomplished by keyword searches. When using information in a nonlinear information system, researchers can request access to certain information and have that information located, retrieved, and displayed directly, without having to wade through information that, in our earlier world, would have "come first." In our nonlinear world, there is no "first," "next," or "previous" in context. Instead of looking up a case and reading through it for the relevant citation, the lawyer requests the display of all instances of the relevant keywords. Instead of manually transcribing or photocopying the material, the lawyer can use the computer to copy and paste the selection directly into her brief. The use of such a nonlinear information system means that much less time is used in getting to the information, locating the particular material required, and retrieving and placing that material in the final product.

Consider an institution where people conduct extremely expensive and time-consuming experiments. The people involved live and work in different countries, so getting together for research and meetings is difficult. The people want to have certain work done, but it might duplicate work done at a similar institution in another country; no information on whether that work has been done exists at the institution. A researcher is tasked to determine whether

certain work was done and, if so, to obtain the raw data and parameters. She logs onto her computer and links to several search engines on the World Wide Web. After she keys in a search string, the search engine presents a list of links to pages containing that string. Now, instead of using a database query, the researcher chooses hyperlinks to other nodes, retrieving further information. This activity becomes the query mechanism; by choosing links, the researcher refines the query until she links to the node with the information sought or until the full answer has been obtained from access to several nodes. (This all sounds much more professional than the usual term—*browsing*.)

The links are to Web pages located in other countries, on other continents. There is no need to go to the data. The data available to the researcher is not limited to that provided to the researcher by the institution. And search engines with *robots* scan the Web looking for additional information, indexing it for the researcher. No one at the institution is required to maintain a library or card catalog of material.

Using nonlinear information, researchers can follow leads that persons creating tables of contents and indices would never imagine as connections. Authors can link by analogy and simple association, leaving a clear path of their thoughts that some may choose to follow and others may choose to ignore. Nonlinear information systems allow us to work as we think. Nonlinear text can provide choices for consumers to follow links to specific information made available by a business about its products. For the casual browser, nonlinear text can provide a truer representation of actual experience. Whether for work or leisure, links can take the reader to ever more specific data, making the world the repository of a database virtually sitting within the reader's monitor.

As the following history points out, key elements were always missing from proposed nonlinear information systems, keeping them from being successful. For some proposals, the hardware was impractical; for some, the software required was proprietary; for others, the available information was limited. The world has been waiting for a system of nonlinear information that is literally boundless, with no limits to the available data, no limits to connectivity, and no requirements of prescribed hardware, software, or data.

The Evolution of Nonlinear Information Systems

Nonlinear information systems have been considered ever since humankind has had the leisure for tinkering. Many of the proposals were high-tech at the time, but fatal limitations that were insurmountable existed until very recently.

Ancient Reading Machines

Most people now associate nonlinear information systems with the World Wide Web, but the concept is much older. In 1588, a book published in Paris called *Le diverse et artificiose machine del Capitano Agostino Ramelli* (*The Various and Artful Machines of Captain Agostino Ramelli*) presents the concept of "the reading wheel." An engraving shows a man sitting in his

study before a wheel that resembles a waterwheel. Instead of paddles over which water would flow, the wheel is made up of lecterns with open books on them. The man could rotate the wheel up or down and have one of several books before him for reading. The books were affixed to the lecterns so that they not only remained on the lectern as the wheel rotated, but stayed open to the appropriate page. Ramelli apparently suggested the reading wheel both for scholars who had much reading and for those with gout who had trouble walking to their shelves for more reading material.

Ramelli seems to have grasped that nonlinear information systems would presume that the reader would not read from beginning to end. In fact, it has been suggested that a hypermedia document should not be read, but explored. In contrast, books and magazines are the archetypal linear information systems, with their numbered, sequential pages. In a magazine, if an article is continued on a later page, that fact is noted in the article, and most readers will turn immediately to the page to finish the article. And although it would be possible to read the pages of a novel in random order until they have all been read, it is unlikely that the reader would remember and understand the development of the plot or any of the characters because the author has presumed sequential reading and created a linear information flow.

Conversely, in a nonlinear information system, there would be no predefined order, and it would be presumed that the reader could mark passages for later study, add notes, and have a history of items read for later return, perhaps to follow other links. The World Wide Web is such a nonlinear information system, and the thread of its development can be followed from more recent times than Captain Ramelli's.

If we imagine our old world of linear information (beginning with such memorized heroic epics as *The Iliad*, and on to illuminated parchment scrolls, and ultimately to printed books) as the unspooling of the thread of the continuous context in which we live, hypertext becomes the web of, not discontinuity, but nonlinear context, in which we aren't required to read or view data in sequence. In life we make our own order out of our experience. With nonlinear text we have that choice as well.

A Modern Attempt at a Nonlinear Information System

Entire philosophies and concepts of nonlinear information systems have been put to paper. But paper was the rub, the shortcoming of all the suggested systems. The requirement that nonlinear information systems be affixed to paper doomed them all to failure.

Some time ago, people kept track of research notes on index cards. One company made index cards with holes around the edges, allowing users to cut notches in the cards. The holes were numbered, so that researchers could assign certain topics to certain numbers. A person could put a knitting needle through a deck of the cards at the appropriate hole and shake the deck, and cards notched at that number would fall out. Voilà! All the research on that topic was in a pile on the desktop next to the typewriter. Those cards had to be subsorted by a second round of notching or by hand to bring them into usable order for drafting the paper.

This kind of nonlinear information system limited researchers in several ways. For one thing, the information did not come to the researcher; it had to be gathered, usually from a library. Thus, a researcher had to locate and travel to a library containing appropriate information for the topic at hand, consult the card catalogue, retrieve books and periodicals from the stacks, and read the appropriate pages. Only that data physically present with the researcher could be consulted and transcribed to the note cards. Each card had to be either written or typed with the notes from the readings. Then an index card had to be prepared, with topics set forth and numbered, and the note cards notched at the appropriate number on the edge. Only those cards physically present with the researcher could be consulted and transcribed to the typewritten final draft. Any errors or missing information required another trip to the library. Paper-based nonlinear information systems posed problems.

With the advent of computers, imaginations were fired anew, but the early thinkers were brought up short by the hardware. Few people had access to mainframes, and mainframes had too little memory and were too slow and too limited in output (either on paper or on CRTs that were monochrome and displayed only text). But as Moore's Law came into play, computers shrank to the desktop, power increased exponentially, prices fell, and monitors glowed in color and showed more than ASCII text.

And, as often happens, the original, creative thinkers were left behind.

Vannevar Bush, the Progenitor of Modern Nonlinear Information Systems

Vannevar Bush is often referred to as the modern progenitor of hypertext. Born in 1890, he taught at Tufts College and at MIT, where he was later appointed dean. He worked on optical devices and on machines for rapid selection of specified spools of microfilm. During World War II, he was appointed by Franklin Roosevelt as the Director of the Office of Scientific Research and Development.

In the 1930s, Vannevar Bush proposed a differential analyzer that he called a *memex*. This machine could store vast amounts of data, and Bush considered various means of providing links so that the information would be accessible to the user. Bush used such terms as *trails* and *footprints* to suggest where one could go and where one had been. While serving as science advisor to President Roosevelt, Bush had the opportunity to refine his thoughts on associative information, and in July of 1945 *The Atlantic Monthly* published the seminal work on nonlinear information, "As We May Think." Given the length of the article's gestation from Bush's conception of the differential analyzer in the 1930s, the maturity of thought expressed in the article should not be a surprise. It was recognized by the editors of the magazine as a call for a new relationship between humankind and knowledge.

As World War II ended, Bush realized that the tremendous advances in knowledge had not only left the individual behind, but were on the threshold of overwhelming the individual, who would retreat to ever-narrower specialties, leaving a chasm between disciplines that no one person could bridge. Bush viewed the loss of interdisciplinary knowledge and communication as no less than catastrophic.

Bush describes the "hardware" of the memex in the archaic terms one would expect of a writer living in 1945. It consists of a desk with translucent projection screens embedded in the top, a keyboard, and buttons and levers. Content is purchased on microfilm, ready for insertion and projection, but provisions are made on the desk for original work to be photographically reduced to microfilm for later display. Books are projected onto the screens, with a lever to control the speed of changing pages, much like microfilm projectors used today. Books and other materials are called up for display by typing in a code. When new material is created, it is assigned a code, and this code is stored in an index. There are several screens, so several items can be viewed at once. The reader is somehow given the ability to make annotations and marginal notes, as one would if holding a book. Bush envisioned miniature cameras the size of walnuts that scientists could wear on their foreheads to photograph the important things that they saw.

Although the descriptions of the desk, its mechanics, and the accessories are decidedly of their time, Bush transcends the 1940s with his next section: associative indexing. Any item at any time can be caused to select another immediately. Bush goes astray by trying to be specific in how links are to be established (dots on microfilm recognizable by a photocell); nevertheless, his concept is electrifying. The user can build a trail; items are forever joined and immediately recallable merely by tapping a button. Original material can be added to existing works. When many items have been thus joined to form a trail, that trail can be followed by the user at will. Trails can be connected, so that readers can branch off into interesting side excursions. Trails can be copied and transferred to acquaintances, who can incorporate them into their own trails, add original materials, and copy and transfer the expanded trails to others for further aggregation. Trails can be handed down to new generations, giving new users access to the knowledge and experience of those who have gone before. It was this concept and the promise of emerging technologies that kept the flame of nonlinear information systems burning in the minds of thinkers for the next 20 years.

Curiously, however, Bush seems to have neglected a key element of his great idea: the method of making the trail available to others. The memex user in Bush's article sits at the desk alone. Microfilm is purchased with material already recorded. Notes and memoranda can be added, but there is no clear idea as to how the trail would be handed on to others. Bush mentions earlier in the article that microfilm can be mailed; we are left to guess that the memex user would make microfilm copies and have them delivered to people. Ultimately, Bush leaves unresolved the problem of access to the nonlinear data. The memex is a one-person nonlinear system with no inherent means of transferring data to and from the desk. It is the user's responsibility to obtain new data (perhaps by mail, perhaps in stores), create trails, and find a method to transfer and share the data with others.

Doug Engelbart, Developer of User Interfaces

Doug Engelbart read "As We May Think" while stationed in the Philippines after World War II. The ideas propounded by Bush remained with him, but it was not until the 1960s that the means of implementing a memex became possible. Working with William K. English and John F. Rulifson, Engelbart created the oN-Line System (called, of course, NLS), a system of collaboration for teams of workers located in different places. As part of NLS, the team created the concepts of using outline editors for developing ideas, what was later to be called hypertext linking, word processing, windowing, online help facilities, and consistency in the user interface; the hardware required and invented included the mouse as a pointing device for selecting areas on a CRT.

It is important to remember that in 1968 computers were mainframes and programming was generally done by punch cards or paper tape. Output was often printed on a teletype. Engelbart and his team saw that this was not an ideal environment for the enhancement of human thought. But they also had the means, for the first time, to create a new environment for the users of computers. As computing power increased, the ease of use increased as well. Engelbart's work was influential in research then being done at Xerox's Palo Alto Research Center—and on Ted Nelson's thinking.

In April of 1997, at the age of 72, Engelbart was awarded the Lemelson-MIT prize in recognition of his invention of the computer mouse and dozens of other devices and ideas that make the user interface easy and convenient. The award was established in 1994 by Jerome H. and Dorothy Lemelson to recognize U.S. inventors and scientific innovators; it is administered by the Massachusetts Institute of Technology. Engelbart now heads the Bootstrap Institute, a nonprofit, tax-exempt organization with offices provided at no charge by Logitech International, a mouse manufacturer.

The Origins of Hypertext Markup

With the advent of computers, the limitations faced by inventors in mechanical times were no longer insurmountable. New obstacles remained to be overcome, however, including overcoming the concept of computers as number crunchers, developing friendly user interfaces, and bringing the nonlinear information system itself to a level where there were few barriers to its use.

Ted Nelson and Xanadu

Ted Nelson is credited with coining the word *hypertext*. Nelson was born in 1934 and received his undergraduate degree in philosophy at Swarthmore College. Returning to college in 1960 for a masters in sociology, Nelson enrolled in a computer science course designed for humanities students. He realized the potential of computers in the areas of thought and writing—areas not often connected with computers at that time. Mainframe computers were not conducive to that use, but he continued to develop his ideas. In 1965, he presented a paper at a conference of the Association of Computing Machinery. Nelson had made the first use ever of the word *hypertext*.

Nelson has written frequently about his vision of hypertext, which he has called Xanadu. His book *Dream Machines* was published in 1974, and the book *Literary Machines* was published in 1988. He has written many articles as well. His conceptions include linked images, branching to other images, hypermaps with layers of detail, and linked moving video. Nelson has shown a remarkable prescience in his view of the world of hypertext.

Nelson's article "A New Home for the Mind" appeared in the March 1982 issue of *DATAMATION* magazine, its 25th anniversary issue. In this article, Nelson proposed nonsequential writing, saying that writing had always been sequential because pages had been sequential, and there was no alternative. (Whether sequential pages were the cause or the effect of sequential writing is left to the reader to ponder.) Nelson proposed using the computer to vault our minds into the hyperspace of thought. He called this new realm of immediately available texts and graphics the *hyperworld* and the storage system a *hyperfile*. (*Hyperbole* was not mentioned in the article.)

Unfortunately, he has failed to bring Xanadu from the hyperspace of thought into this world. He announced its release due in 1976, 1988, 1991, and 1995. Sadly, in juxtaposition to the article in the 25th anniversary issue of *DATAMATION*, *Byte* magazine, in its September 1995 20th anniversary issue, heralded Xanadu as its first example of vaporware.

Nelson named his system after ideas in Samuel Taylor Coleridge's "Kubla Khan":

In Xanadu did Kubla Khan
A stately pleasure-dome decree:
Where Alph, the sacred river, ran
Through caverns measureless to man
Down to a sunless sea.

Legend has it that the poet wrote the poem under the influence of a drug, that he was interrupted before finishing the work, and that he was never able to regain his prior state of inspiration and finish it. Although Nelson intended to provide a system where literary memory never flagged, his choice of Xanadu as a name has come back to haunt him as surely as Xanadu haunted Coleridge, who was unable to regain the magic realm and finish the work.

Among the problems delaying Xanadu are the coinages of terms and the changes in terminology. Nelson proposes *xanalogics*, *humbers*, *docuverses*, and *tumbler arithmetic*. Understanding what Nelson means and keeping up with his new names for his concepts make public acceptance of Xanadu a daunting task. Although the overarching concepts of Xanadu remain alive in today's World Wide Web, Xanadu itself seems to have disappeared into *caverns measureless to man, down to a sunless sea.*

Bill Atkinson and HyperCard

The following HyperCard script from John McDaid, "Uncle Buddy's Phantom Funhouse," published by Eastgate Systems, Inc., shows the ability of a skilled author to create a working

script that also stands alone as a poem. (This code is copyright 1992 by John McDaid and used with permission.)

```
on mouseUp
Global thermoNuclearWar
put the script of me into tightOrbit
put tightOrbit into eventHorizon
put empty into first line of eventHorizon
put empty into last line of eventHorizon
put eventHorizon after line thermoNuclearWar of tightOrbit
set the script of me to tightOrbit
put thermoNuclearWar + 10 into thermoNuclearWar
click at the clickLoc
end mouseUp
```

Bill Atkinson created much wonderful software for Apple's Macintosh computer, including MacPaint and MultiFinder. In 1987, Apple Computer introduced Atkinson's HyperCard to the world. HyperCard uses the graphics capabilities of the Macintosh computer to show on its screen a virtual deck of cards; cards can contain text, sound, video, or pictures, along with buttons and other navigation aids for the user to go from one card to another.

HyperCards are created in an interpreted language called HyperTalk. As shown in the working script at the beginning of this section, HyperTalk is similar to English and can be read and understood without a great deal of study. Using HyperTalk, script authors can create cards, accept input from the user from the keyboard or mouse, act on that input, and display the results. People have scripted complex programs and games using HyperTalk, although it is an interpreted language and, thus, sometimes relatively slow.

HyperCard suffers from several limitations. It has search and history functions built into it, but not links. Worse, the author cannot build links into the text of a card; the author must provide clickable buttons, which must be placed so that they do not obscure the text. Addition of text to a card might require the author to rewrite the location of any navigation buttons. Originally, HyperCard supported only one size of card, only black and white, and only one visible card at a time. The cards could not contain scrolling windows. But HyperCard met an untapped need and became quite popular in spite of its limitations.

Competitors quickly appeared and addressed some of the shortcomings. Apple handed HyperCard off to Claris International, Inc., a subsidiary servicing Apple's software, and Claris has updated and extended HyperCard to include scrolling, variable card sizes, debugging tools, and better printing. However, the program is proprietary, and potential users of stacks must have acquired the program; data stored in stacks is not available to other programs that cannot interpret HyperTalk. Although HyperTalk allows calls to AppleScript, no easy method native to HyperTalk allows connection to other computers for locating and retrieving data. The HyperCard user is, essentially, at the same point as the memex user in 1945. Users can obtain and add to stacks or create their own, but they remain bound by the content physically present on the computer. The program has no inherent method of transferring data to and from the desktop.

HyperCard is now sold as a separate product, but its limitations will restrict its use as a nonlinear information system. It has served well, however, and it has shown the power of the modern computer—coupled with a consistent graphical user interface—to free the user from the linear demands of paper and of the early computers, where steps were relentlessly sequential. With HyperCard, authors could publish text, pictures, and even animation, with different paths for readers to follow. With some knowledge of scripting, readers could add text to the cards, create new cards, and add links to and from existing cards obtained from others. Bush's memex was, at last, in its infancy, built directly upon the work of Engelbart and NLS.

The foundations of hypertext were laid. It remained only to tie in the reality of the computer and software with the vision of Xanadu.

The World Wide Web Becomes the First Practical Nonlinear Information System

With many of the ideas, hardware, and software in place for the first time, nonlinear information systems became workable. The last steps required seeing the pieces as part of a whole, putting them into place, and inventing the mortar to hold them all together to form the foundation for true and unbounded nonlinear use, unconstrained by platform or software.

Tim Berners-Lee and the World Wide Web

Tim Berners-Lee was born in England and graduated from Oxford University with a bachelor of science degree in physics. After working at a few jobs involving text processing and communications, he landed at the European Particle Physics Laboratory at CERN. As Bush foresaw, scientists working for CERN in different countries were having trouble communicating with each other. Although Berners-Lee had been hired to work at one of the high-energy particle accelerators, he became involved in the effort to facilitate communication between the scientists at CERN.

Earlier, he had written a program called Enquire that allowed him to keep track of his personal notes and to create associations between related notes in an effort to make notetaking and notekeeping work more closely to the way we think. That idea carried over into his new concept for using computers to enable users to look at a database of ideas with hypertext as the query mechanism. In Berners-Lee's model, this "meta" database is made up of as many databases as necessary on as many computers as necessary, with no single person in charge of the data or the database. Everyone at CERN could keep his or her data current and make the results available by computer. Everyone would have access to all the data and could build relationships by linking and bookmarking links. There would be no chasm between disciplines; interdisciplinary knowledge would be readily available, searchable, and viewable; and the building of bridges from discipline to discipline would be as easy as making a link. All users at CERN could record everything they saw and did, and everyone else could see the data and make use of

it in their own fields. (As a historical note, scientists at CERN are *not* walking around with walnut-sized cameras on their foreheads photographing things of interest to them. However, students at some universities *are* wearing video cams throughout their day, broadcasting the view live to the World Wide Web.)

Finally, the memex became reality.

It took a while longer and much more work, of course. Berners-Lee proposed the World Wide Web to CERN in 1989 and began work in earnest in 1990. The fear at CERN was one of the problems posited by Bush: the loss of information. The work at CERN is incredibly expensive and time-consuming. People were hired and began work with little introduction to the current state of research, and the average length of stay was two years. Technical details of work were often lost and were difficult and time-consuming to reconstruct. Duplication of effort was rampant.

In a stroke of brilliance, Berners-Lee transcended the problem at hand. He understood several problems that had to be overcome, and he went far beyond the immediate need of CERN with his solution. He saw that a hierarchical system of information would collapse under its own weight, if not in a year or two, then in 10 years; that the information in the system would not be limited to text; and that indexing would limit access to information and hide it from those who needed data from a different perspective. His answer to these problems was distributed hypertext, where people kept their data on their computer, but the computers were linked on a network with the data publicly available on the network. With the network grown to include the Internet, the information available on this system may have no bounds.

NOTE

Although many of us are familiar with URL, a Uniform (or Universal) Resource Locator, the more general term, URI (Uniform Resource Identifier), incorporated not only URLs but also Uniform Resource Names and Uniform Resource Citations. URNs and URCs will have wider use in later versions of HyperText Transfer Protocol. See Chapter 5, "Behind the Scenes: HTTP and URIs."

Given approval, Berners-Lee wrote the server and the client software on his NeXT computer. The software was made available at CERN, then to the world through the `alt.hypertext` newsgroup. His colleagues and he then grappled with the protocols, inventing URIs, HTTP, and HTML along the way. As discussed in later chapters, not all the difficulties have been (or ever will be) ironed out, but with the addition of a graphical user interface, the World Wide Web has exploded across the face of the planet, enabling everyone to become publishers and granting with a vengeance Bush's wish for people to be able to sit at a desk, record their thoughts and observations, and make that record available to following generations. We who now live among the mess ponder whether all those thoughts and observations are worthy of being in the record.

A Theory of Practical Hypertext Markup

The author of the nonlinear information system must furnish a method of linking related ideas and a search mechanism for specific queries that may not have immediately obvious answers. Ultimately, the nonlinear information system can have no bounds. With a uniform linking mechanism, the reader is not restricted to an author's work but can follow links to source documents anywhere in the world. These benefits are also drawbacks if they are poorly implemented. Maintaining context in a hypertext world is difficult but necessary.

Divide the Content into Manageable Nodes

As discussed in further detail in Part IV, "Effective Web Page Design Using HTML," effectively designing documents presenting nonlinear information is considerably more difficult than writing for narrative text. Theorists of hypertext have several constructs that are useful in designing Web sites. The first is the division of the text into *chunks* that deal with one theme, topic, or idea. After the material is chunked, it must be written so that it can be read as a standalone node by a reader who may have come to that node from any of several links provided by the author or from a search engine. The author cannot assume the reader has information from "earlier" passages, as is the case with narrative text writing. Poorly chunked nodes can result in a procession of sound bites with no context and little content or in a node that is an unreadable cacophony of links shouting for attention and overwhelming content.

Provide Context for Your Reader in Each Node

Linear text is self-contextual. I can refer later in this paragraph to *chunking* and assume you are familiar with that term from a previous paragraph. In nonlinear information systems such as the Web, formerly simple words such as "later" or "previous" become context traps for the author. While we can mention here what we discuss later, in a hypertext situation such a discussion may never come before a reader who does not follow a particular set of links. (Indeed, a reader might follow a link prior to even finishing reading a node and never return. "Later" never comes.) Each node must, therefore, not only be correctly chunked, but it must also be written so as to provide a set of context clues that a reader with no prior link to that node can use to become oriented to the information being presented.

With the heavy use of search engines, the Web site author can no longer assume that the reader can use the "back" facility of the browser to return to a "previous" page. The reader might have come to a page from out of nowhere (from the author's point of view). It is imperative, therefore, that the page contains links to a hierarchy within the Web site where readers can orient themselves and go on to other relevant nodes. In addition, the links within the document must have a purpose that is obvious to persons coming into the node from another Web site. Each link should have a clear statement of its function in the context of the document and without further explanatory material.

Readers should be aware that they are within a set of nodes as they follow the trail established by the author. By providing a similar look to all the nodes on a Web site, an author can create

a visual context with which the readers will grow comfortable—and they will be more likely to notice if they follow a link to another site with a different appearance. Navigation aids within the site should be consistent in all the pages; readers should not be burdened with having to remember how they got to a page, where to go to get back, or how to determine where next to visit. Having to remember the meaning of icons from one page to another, for example, could be distracting to a reader intent on gleaning information from the author's text. It is aggravating and confusing to find that different icons mean the same thing on different pages or that the same icon means different things.

Provide a Context for the Author and Webmaster

As the author creates documents, the author must retain a clear understanding of the structure of the site. Naming the documents can be a great aid to both the author and the reader. Links to documents with names that indicate the purpose of the Web page are helpful in keeping track of the structure of the site and the location of both the reader and author in the great scheme of things. The site author might find it helpful to keep an outline of the structure, with links and document names handy, ensuring correct citation and the absence of links or documents without corresponding documents or links.

When a hobbyist is creating a site for pleasure, the author may well be able to create and publish the entire site while retaining the overall concepts and presentation in mind. Business sites, on the other hand, often have several persons authoring content and another person creating the Web site, its navigation aids, and overall interface. The group must have a person who can review (or have reviewed by appropriate persons) each document for technical accuracy, legal issues, management purposes, and consistent style in all documents. Such reviews are crucial for professional Web site authors. Additionally, someone must keep track of the Web site as a whole. Web sites have a tendency to grow, making changes more difficult (where are all references to Part Number 234-4433?) and navigation more complex (users might give up when they have to pursue an item that is buried under several layers of pages, each taking time to download). The Web site author planning a site must keep in mind the importance of designing for future maintenance of the site and of each document.

Summary

The World Wide Web works as a nonlinear information system. The ideas and software and hardware that preceded it all lacked one or more crucial facets that are present in the Web. Some of them required proprietary software and ran on only one operating system. Some provided no inherent method of communicating ideas. The World Wide Web has proved successful because it is not proprietary and requires no particular hardware or software. It has inherent in its concept the free exchange of ideas, the capability to provide links to information that may be located anywhere in the world. "Content" becomes available to a reader without the requirement that it be stored in the reader's computer. Nonlinearity without boundaries has finally been achieved.

CHAPTER 2

The History of HTML

by Philip Stripling

IN THIS CHAPTER

When Tim Berners-Lee was faced with the many options available to implement his concepts, he chose Standard Generalized Markup Language (SGML), setting the stage not only for total independence from platform operating systems and languages, but also for often rancorous debate on the issue of just what HTML is supposed to be for.

On the surface, SGML is the ideal choice for the purposes envisioned by Berners-Lee. SGML is an internationally recognized standard for text information processing, providing distribution, search, and retrieval of electronically stored text. Documents marked up in SGML fashion have two elements:

- The content, which is the information to be conveyed.
- Information about the content, identifying the basic structure of the document (headings, paragraphs, lists). The format of the document is not, and cannot be, specified in SGML markup.

This was ideal for Berners-Lee's purposes. A scientist at CERN could create content, SGML markup could be added later, and the resulting document could be made available by network to all persons with network access. Because SGML is platform independent, anyone with a computer and appropriate software (this combination is sometimes called a *client*) that could parse SGML documents could then read the data. The software used to parse and present data marked up in SGML was personalized by the user, taking into account the hardware in use. A person using a dumb terminal had to have a way on that terminal to distinguish text that was marked as Header 1 from Header 2 and Header 3, and also to distinguish lists from paragraphs. Different fonts, different font sizes, colors, italics, and similar conventions now taken for granted were not available to all clients. Thus, users were expected to set their software to meet their needs and available equipment, which quite likely varied from lab to lab. Because SGML markup identified only the nature of the contents, it was the reader's responsibility to determine how to display the different levels of meaning. As a result of the limitations of some displays, the convention arose for client software to ignore markup it could not interpret, displaying the content without the benefit of that particular tag.

Before HTML, the author of a document was never concerned with how the document would appear on someone's monitor. It was accepted that appearance was the province of the user. Unlike authors using `troff`, for example, HTML authors could not specify font names, font sizes, margins, or white space between elements. Users set those specifications to their own satisfaction, allowing for monitor size and available fonts.

The Development of Hypertext Markup Language

One of the requirements of creating a usable nonlinear information system is to make the system easy for non-experts to understand and use. Unlike AppleScript, HyperTalk, and WinHelp files, marking up text for presentation on the Web uses a limited set of tags that are simple and easy to understand. This seeming simplicity, however, was difficult to obtain and is proving difficult to maintain.

A WORLD WIDE WEB TIME LINE

1989

March

Tim Berners-Lee circulates "Information Management: A Proposal" at CERN.

1990

October

Proposal redrafted; World Wide Web named.

November

Berners-Lee creates a WWW browser on his NeXT computer.

1991

February

Project presented to CERN.

March

www, a line mode browser, is written.

May

Stanford Linear Accelerator Laboratory sets up first U.S. server; general release of www over CERN network.

August

Information is posted to several newsgroups providing information on the World Wide Web, including a draft for HTML 1.0.

1992

January

www available generally by ftp.

April

Erwise, a Finnish graphical client for X, is released.

May

Viola graphical browser is released by Pei Wei at Berkeley.

1993

January

Other browsers are available; a few dozen Web servers are online.

February

NCSA Mosaic for X is released.

continues

continued

September
NCSA releases early versions of Mosaic for common operating systems.
October
Web servers are now estimated at 200.
November
HTML+ is made available for informal discussion.

1994

January
General releases of commercial software to allow home users to connect to the World Wide Web over Internet connections.
March
Andreessen, et al., leave NCSA for Palo Alto, California and Netscape.
April
Initial draft of HTML 2.0 released to the public.
July
CERN and MIT announce an agreement to form the W3C.
October
Netscape 1.0 beta released.
December
First meeting of the W3C held at MIT; Netscape 1.0 finalized.

1995

February
HTML 2.0 released as an RFC.
March
HTML 3.0 released as an RFC.
April
Netscape 1.1 released.
May
W3C announces selection of Cascading Style Sheets to provide visual effects and provides initial information on HTML 3.0.
July
Netscape 1.2 released.

August
Internet Explorer 1.0 finalized.
September
RFC for HTML 2.0 approved.
October
NCSA Mosaic 2.0 final beta released; Netscape 2.0b1 released.
November
W3C workshop on style sheets; Internet Explorer 2.0 finalized.

1996

January
NCSA Mosaic 2.1 released.
March
W3C announces that the "market leaders" will work with the W3C to establish interoperability standards; Netscape 2.0 finalized; Internet Explorer 3.0a1 released.
April
NCSA Mosaic 3.0 beta released; Netscape 3.0b1 released.
May
HTML 3.2 released as an RFC.
July
Cougar released to the public.
August
Netscape 3.0 finalized; Internet Explorer 3.0 finalized.
October
Internet Explorer 3.01 released.
December
W3C recommends Cascading Style Sheets, level 1; Netscape 4.0b1 released.

1997

January
W3C recommends HTML 3.2; NCSA ends development with Mosaic 3.0.
February
Netscape 4.0b2 released.
May
Internet Explorer 4.0 Developer Release announced.

Level 0 of HTML

In using SGML to describe how to format content, Berners-Lee employed another aspect of SGML: It is a language used to define other languages. Berners-Lee and his colleagues used SGML to describe the rules for Hypertext Markup Language in a document type definition (DTD), the basis for the structure of documents on the World Wide Web. (DTDs are discussed in detail in Chapter 3, "SGML and the HTML DTD.") Under Berners-Lee, HTML followed the SGML rules of platform-independent content markup with no provisions for representation of the document in HTML itself. Users were to provide formatting through their clients—and the client would never be known at the time of document creation and markup.

In its first implementation in 1990, HTML was at level 0. At that time, the means of communication over the Internet included e-mail, ftp, and Telnet, using the TCP/IP protocol. Gopher was becoming a popular means of indexing information, and Gopher servers were being introduced to the Internet, along with Archie, Veronica, and Jughead. WAIS was also being introduced as a means of providing access, especially to students on college campuses. To allow worldwide access to HTML documents, Berners-Lee and his associates introduced the ideas of the Hypertext Transfer Protocol (HTTP) and URIs to provide addresses and a means for locating the data. (See Chapter 4, "The Structure of an HTML Document," for more information.)

At level 0, HTML provided a platform-independent means of marking data for interchange. The concept was that servers would store and provide data and that clients would retrieve and display them. The servers would send the data via HTTP, but other existing protocols were available; thus, HTML nodes could provide access not only to other HTML nodes but also to Gopher space, ftp, Network News, and so on.

HTML 0.0 was very close to SGML. Oddly, the only required element was the `TITLE` element, and many older pages still remain that start with a title, and then go straight into the text—no `<HTML>`, no `<BODY>`, and no `</BODY></HTML>`. These older documents often use `<P>` at the end of each paragraph; the closing paragraph tag was not required, and the paragraph element was viewed simply as a device to *separate* paragraphs rather than as a *container* of paragraphs. Level 0 provided six levels of header, but expected that each level would be used only once and that the levels would be used sequentially. Thus, `<H1>` would be first, and then, if necessary, `<H2>`, `<H3>`, `<H4>`, and so on. This requirement was imposed by SGML parsers of the day that used certain conventions to read and render the structure indicated by the markup. More than one header of the same magnitude or headers out of sequence could confuse the parsers and cause problems with the display.

NOTE

Although I have referred to "tags" in this chapter and in Chapter 1, "Hypertext Markup in Theory and Practice," it is time to become more formal in describing the parts of SGML and HTML. Based on RFC 1866, an *element* is a component of hierarchical structure defined by a DTD. *Markup* is the syntactically delimited characters added to the data of a document to represent its structure. There are four different kinds of markup: descriptive markup (tags), references, markup declarations, and processing instructions. A *tag* is defined as markup that delimits an element, including a name that refers to an element declaration in the DTD, and may include attributes. *Attributes* are values within tags that allow for further refinement of the markup expressed by the tag. (See Chapter 3 for examples and further explanations.)

Level 0 allowed a `<BODY>` tag (even though it was not often used), and authors could have the following elements contained within the body element: address, anchor, blockquote, break, headings, horizontal rule, image, list (definition, directory, menu, ordered, unordered), paragraph, and preformatted text. Provisions were made for the inclusion of special characters through the use of escapes.

TYPICAL SOURCE MARKUP IN 1992

A typical page of those days would look something like this in its source markup:

```
<title>New Page in a New World</title>
This is a new page in the new world of the Web. In this new world
we do not think of the &lt;p&gt; or paragraph tag as a container,
but as a means of introducing a gap in the flow of text.<p>

As a result, it was usually placed at the end of a paragraph,
telling the reader of the source mark up that the paragraph had
ended rather than that a new one was to begin.<p>

Since only the title element was required, pages were often
remarkably simple when viewed in the source mark up, and were
mainly narrative text explaining things just as in a book. Many
pages had no links, and the authors expected readers to be
approaching the page from another node and returning via the
"back" facility of the browser.
```

Level 1 of HTML

The discussions of HTML 0 brought to light several shortcomings, and in 1992 Dan Connolly began development of HTML 1.0, which was released on the Internet that year. After months of discussions, Berners-Lee wrote an Internet draft Request for Comments (RFC), which was released in mid-1993. The idea of an HTML container was added, with a `HEAD` element separate from the `BODY` element. Opening and closing tags were required for some elements. Along with `TITLE`, the `HEAD` element could contain the attributes of `ISINDEX`, `LINK`, and `BASE`, giving the document a context within a larger universe. Along with the anchor element, the image

element allows (at first) GIF files to be displayed within the text. The horizontal rule was also introduced, beginning the slide down the slippery slope of attempting to define the display.

Level 1 also introduced one of the most useful elements, FORM. Forms enabled authors to have input fields on their nodes, allowing feedback from users, opening the door to considerable interaction via Common Gateway Interface (CGI) scripting. The potential of forms and CGI is still being explored, with HTML 4.0 including additional functions.

Level 1 also added the elements CITE and CODE; certain style elements having nothing to do with marking structure such as emphasis, keyboard, sample, strong, and variable, and the character formatting elements for bold, italic, and teletype (to force a monospaced font where available). While Level 1 was being discussed in the newsgroups and on the e-mail list, additional elements and extensions to HTML were being proposed, and HTML+ was in the wings.

HTML+ and the Introduction of Graphics

Dave Raggett proposed HTML+, incorporating graphical and display elements into HTML, taking fuller advantage of the capabilities of Mosaic and other graphics-based browsers then becoming available. HTML+ offered the means for linking to and accessing PostScript documents, JPEGs, MPEGs, and sounds as well as GIFs. MIME was used to allow the extension of HTML file types; the data available over the World Wide Web was not circumscribed by the 7-bit ASCII conventions of the Internet. Now, not only are there no limits to the number of computers and users and no limits to the amount of data, but there are no limits to the types of data available.

HTML+ was an important conceptual step forward for HTML. Raggett included elements and attributes for superscripting and subscripting, footnotes, margins, inserted and deleted text (for change logs of documents being drafted by different persons at different locations), alignment of content, tabs (with alignment), tables, mathematical formulae, an extended set of character entities (including fractions, and inverted exclamation and question marks), and figures.

Raggett grasped and understood the concept behind the World Wide Web. With HTML+, he sought to extend HTML beyond the pedestrian concept of an article placed on a computer with hyperlinks. He understood the use of the Web not only as a means to accomplish Bush's goal of bridging the knowledge chasm, but also as a distinct medium with uses even Nelson had not dreamed of. Raggett introduced in HTML+ the means to link not only to other text, but to sound files, moving image files, PostScript, and any format later introduced by the use of MIME extensions to describe the new file type.

But he remained true to the SGML background of HTML in this proposed implementation. HTML+ would not deal with instructions on appearance, but would leave those details to the specifications of the user to the client. Elements, attributes, and values remained case insensitive, in accordance with SGML rules. HTML+ parsers ignored tags they did not recognize, allowing authors and users to avail themselves of other SGML markup in HTML

documents for indexing and other processing outside of HTML. To ensure that browsers would detect this new level of HTML and render the document correctly, it was suggested that a document identifier be used and that it precede all other information in the document, including the <HTML> tag.

With HTML+, Hypertext Markup Language took a giant step toward resolving the disparate demands of content-based markup with the newly born demands of authors for control over the layout of a page of text and graphics elements. Unfortunately, the step was too large. The proposed elements and attributes were difficult to implement; considerable effort would have been required to resolve the various issues, including representation of mathematical formulae on dumb terminals. There was disagreement within the developers' community on the need for tables, forms, math, and even inline images. Furthermore, as browser implementation expanded outside the close circle of HTML development, reasonable people disagreed on what interpretations should be given to the HTML DTD, and different browsers displayed the same markup differently.

HTML 2

In response to demands for a standard for HTML, the development group, under the leadership of Dan Connolly, proposed HTML 2. The World Wide Web Consortium (W3C) was established to handle implementation of HTML standards, and a Request for Comments was drafted and published, setting the stage for formal acceptance of HTML 2.0 as a recognized standard for Web publishing.

Level 2 implemented the FORM element with INPUT, SELECT, OPTION, and TEXTAREA. It also added BR, for line breaks, and the META element, which allows for the description of the document and provides for indexing and cataloging of the contents. HTML 2.0 changed the descriptions of anchor, base, body, lists, head, image, link, and title.

NOTE

One of the benefits of the World Wide Web is the accessibility of the data. I had intended to mention where in the world Berners-Lee, Raggett, Connolly, and Muldrow were during these times to show how little physical contact there was among the team members. But they kept moving! And through the newsgroups and listserv, contributions were made by people in California, Massachusetts, Australia and—well, you get the picture. The very accessibility of information via the Internet and the Web made keeping track of people's locations impossible.

Working with Berners-Lee, Dan Connolly and Karen Muldrow crafted a workable draft that maintained backwards compatibility with existing proposals for HTML and included forms. The DTD provided for start tags only in some instances, required end tags in others, and made

end tags optional in others. Thus, <P> became an optional container, with the end tag </P> available but inferred from context if omitted.

HTML 2 broke no new ground, essentially implementing the status quo. It was, however, immediately workable for all user agents, it followed the rules established by SGML, and it required no long debate on whether the proposed elements were necessary or workable. Furthermore, it allowed the W3C to make a structured response to Netscape Corporation's ceaseless promulgation of additions to HTML, a profligacy driving many developers of HTML and HTML documents crazy. HTML 2.0 became the benchmark against which browsers and markup were measured.

HTML 3

HTML 3.0 was proposed as an attempt to address the competing demands for a markup language that operated across all platforms and for a page description language that was acceptable to software companies in addition to Netscape. The theory was that HTML 3.0 would be fully SGML-compliant, while allowing "hints" to browsers on how to display certain text. HTML 3.0 was developed under the stewardship of David Raggett, the author of HTML+. Among the elements proposed in 3.0 was the FIG element, with text flow supported around figures. Raggett proposed support for mathematical equations, a very useful feature for a system intended to support cross-discipline scientific research. HTML 3.0 also included the TABLE element, an alternative to the PRE element for formatting tabular data.

The ALIGN attribute was added to several elements, including IMG, P, and HR, allowing authors to provide for left, right, or center justification. On the character level, HTML 3.0 proposed a number of new logical elements, including tags for definitions, quotations, language, inserted text, and deleted text. Some of the proposed physical tags were underlined text, bigger text, smaller text, and subscript and superscript. HTML 3.0 also included attributes for a background image, tabs, footnotes, and banners (a section of the HTML document that would not scroll, but remain at the top of the window).

Furthermore, HTML 3.0 was designed to work with style sheets. By removing the display control elements from HTML and using them in style sheets, several agendas were served. Those who viewed HTML as an SGML language would not have to deal with efforts to control the appearance of a node. Those who wanted to do graphics design on the Web would have a better mechanism for rendering attractive Web pages with graphics, text, animation, sound, and so on, and some control over placement of images and the flow of text. And those without clients capable of displaying graphics elements would have properly marked up HTML for their clients to parse, with control over display safely in the hands of the user. Two methods of incorporating style sheets were considered: providing a link to a separate style sheet, or providing a STYLE element with internal style hints in the HTML document. To alert browsers to the differences between HTML 3.0 and its predecessors, it was proposed that a new extension be used, perhaps .ht3 or .html3.

Several books were written discussing the migration from Level 2.0 to Level 3.0 and the dangers of using markup not yet in an approved standard. The HTML 2.0 draft was formally approved, but the approval date for HTML 3.0 came and went. The draft expired. By this time, Microsoft had its Internet Explorer in the marketplace as a free download at the Microsoft Web site. Netscape's Navigator was nominally for sale at $49, but it was also available for download at the Netscape Web site, with payment on the honor system. Not only Webmasters, but authors of books were having fits keeping track of the varying standards of HTML. There was a great deal of confusion and aggravation in the marketplace.

HTML 3.2

In the end, HTML 3.0 was too ambitious. HTML 3.2 was drafted in acknowledgment of reality, incorporating many of the tags already in heavy use on Web pages around the world. It added the tags <SCRIPT> and <STYLE> to pave the way for client-side scripts such as JavaScript and VBScript and for style sheets. HTML 3.2 formalized such practices as colors for backgrounds, text, and links, and width, height, alignment, and spacing for images.

HTML 3.2 became the standard, replacing HTML 2.0. HTML 3.2 provided many new elements and attributes that enlivened Web pages with animation, colors, and sound. It became possible to have attractive, dynamic pages that were compliant with an established standard. There was some disagreement on just who was setting the standards, but at least formally recognized DTDs existed for HTML.

HTML 4.0

With HTML 4.0, the W3C has begun again to attempt to set standards that will divide the markup of content from the appearance of a Web page. As discussed more fully in Part III, "Extending HTML 3.2," HTML 4.0 separates physical styles from content markup by more reliance on style sheets. With both Microsoft and Netscape in on the discussions, we can hope that browsers will finally implement style sheets. Many see that as the best way to suggest appearance while still allowing full use of HTML as an SGML-compliant method of bridging the chasm between mind and knowledge as the level of data increases.

HTML 4.0, being written by David Raggett and Arnaud Le Hors, again recognizes widely used tags such as <FRAMES> and incorporates them into the standard. But Level 4 represents a change in the concept of HTML. Its elements and attributes are heavily skewed toward the use of style sheets, and they provide more diversity in languages and in meeting special needs. In addition, with HTML 4.0, the W3C introduces OBJECT, which is a new element taking the place of IMG, and also makes additional use of LINK, adds functions to forms, and incorporates frames. These new elements and attributes are covered in detail in Part II, "Basic HTML."

Other Proposals

In addition to the new concepts introduced in HTML 4.0, style sheets using CSS1 will be more widely supported in this new version of HTML, and Netscape and Microsoft are proposing

incompatible methods of bringing Web page construction and rendering to the client, called Dynamic HTML. Dynamic HTML should reduce the times now required to download graphic elements many authors use in an attempt to control layout. With Dynamic HTML, an author can use a combination of embedded objects and scripting to exert some influence over the contents and layout of a Web page. Contemplated uses of Dynamic HTML include determination of browser window size with the construction of layout to match and dynamic transfer of content depending on user requests with the construction of a table of contents to match the data.

The use of cascading style sheets should also lessen server loads. Some authors are using transparent GIF images to simulate indentation of paragraphs, as well as using other image devices to influence the layout of their pages. Aside from not working well on all brands of browsers, downloading the artwork takes up as much as half the total time spent transferring data. (The savings in arguments posted to the Usenet newsgroups on indenting paragraphs alone makes the use of style sheets to indent paragraphs worth hundreds of thousands of megabytes of postings.)

Graphical Browsers

Access to the World Wide Web was on text-based browsers. The pages themselves were almost entirely text. There was no method of displaying images quickly and as part of the Web page with www or Lynx. With little but text available, the Web was not much better than the Internet and had nothing to recommend using it instead of Gopher, another text-based method of organizing information on the Internet. The advent of graphical browsers set off the explosion of the World Wide Web.

The Early Browsers

In 1991, Pei Wei had written a program called Viola. It was a HyperText system somewhat like HyperCard and ran under UNIX for X11 workstations. In early 1992, Pei reworked Viola to be a graphical browser for HTML. Viola allowed the user to have more than one font, links were boxed and needed a click of the mouse button for actuation, there was a "history" for ready access to pages already visited, and Viola had the capability to bookmark pages and to display the source markup for the page being viewed. In addition to Viola, there were www, Erwise, and a CERN browser for the Macintosh. In January of 1992, an estimated 50 Web servers were in existence.

In the United States, the National Center for Supercomputing Applications (NCSA) had been given as one of its missions the creation of freely available software to aid the scientific research community. The NCSA at Urbana-Champaign, Illinois, began a project to create a graphical interface to the World Wide Web. In February of 1992, NCSA's Software Design Group (with Marc Andreessen) released an alpha version of Mosaic for the X Window System (Mosaic for X). By September, NCSA had released versions of Mosaic for X, Microsoft Windows, and Macintosh computers. In January of 1993, O'Reilly & Associates and Spry (licensee of the commercial version of NCSA Mosaic) released software aimed at providing all the software

necessary for the home computer user to connect to the Internet and the World Wide Web. In February 1993, Andreessen and some of his colleagues left NCSA and formed Mosaic Communications Corp. (After a brief discussion with NCSA, the company was renamed Netscape.) By October, the Second International WWW Conference had to turn away hundreds of would-be attendees due to lack of room.

In three years, the World Wide Web had become reality—truly worldwide.

The free availability of Mosaic coupled with the capability to transmit graphics for (more or less) immediate display laid the groundwork for the explosion of the World Wide Web. At the time, browsers were freely available for downloading over the Web or by ftp through the good graces of programmers attuned to the convention of providing free software on the Internet or through companies that wished to provide data over the Web and that realized their potential customers needed software clients.

Web browsers were small in both hard disk and RAM requirements. Text and GIF images were generally displayed by the browser, but other data types required "helper applications," as the concept was to keep the browser agile and fast. Virtual Reality Modeling Language (VRML) had been proposed by this time, and JPEG images were often used instead of GIFs because of the larger color palettes and smaller image sizes (in some circumstances). Cottage industries sprang up writing free or shareware helper applications or converting text-based programs to run under graphical interfaces as helper apps.

Mozilla Roars to Life

When Andreessen and his colleagues founded the predecessor to Netscape, they referred to their prototype browser as Mozilla and used a green dragon as their icon. The programmers frequented newsgroups, and their SIG files often used "It's spelled Netscape, but it's pronounced Mozilla." The rumor was that "Mozilla" meant "Mosaic killer." Mosaic at that time had the lion's share of the browser market, but the Netscape team was determined to build a better browser, hoping the world would beat a path to its doorstep. By introducing extensions to HTML that were specific to the Netscape browser, Netscape the company hoped to sell server software that could make full use of those extensions. <BLINK> was born.

A NOTE ON NETSCAPE'S NAME

Netscape's name can become a point of confusion, so here is a little history to make it clear. When the company was formed as Mosaic Communications, the browser was called Netscape. When the company changed its name to Netscape, the browser was called Netscape for a while; in fact, one of the most stable releases was Netscape 1.1N. Some of you might remember the "breathing N" logo in the upper-right corner of the browser window. With the release of version 2, the browser was called Navigator and had a ship's wheel as its logo. With the program called Navigator, I can stop trying to distinguish between Netscape the browser and Netscape the company.

Netscape began its practice of releasing beta software to the public over the Internet and using early adopters to help in the beta testing. Netscape went through several iterations of its browser (by then called Navigator) in versions numbered less than 1. The new browser introduced, in addition to the BLINK element, attributes to the horizontal rule element, allowing the specification of its height and of its width in pixels or as a percentage of window width. Netscape aggravated many SGML users by providing proprietary attributes for presentation control over lists and line breaks and by providing for the REFRESH attribute, which automatically transferred the user to a specified link. Netscape also introduced CENTER, ALIGN, VSPACE and HSPACE, HEIGHT and WIDTH, SIZE, and TYPE (for bulleted lists), among other "unauthorized" extensions. Netscape further angered the World Wide Web community by failing to publish its DTDs for these tags, making it impossible to determine how the extensions should be treated by other browsers.

Netscape unilaterally implemented the following proprietary attributes, which are not included in the official RFC for HTML 2.0:

- <ISINDEX>: An attribute for a prompt that the author wished to have a user see.
- <HR>: The attribute SIZE=*n*, where *n* is an integer from 1 to 5, suggesting additional thickness to the rule. The attribute WIDTH=*n*¦%, where *n* is an integer giving a length in pixels or % is an integer giving the length as a percentage of the window's width. The attribute ALIGN=*left*¦*right*¦*center*, where a horizontal rule less than the entire window width can be aligned with the left or right margin or centered in the window. And finally, NO SHADE, where the rule is a solid color rather than giving the illusion of depth with shading.
- <UL>: The attribute TYPE, allowing the author to suggest a *disk*, *circle*, or *square* as the bullet.
- <OL>: The attribute TYPE, allowing the author to suggest numbering by lowercase or capital roman numerals or letters (in addition to the standard numbers). The attribute START, allowing the author to suggest beginning the sequence with other than *1*, *i*, or *a*.
- <IMG>: This element received the most attention, being given the attribute ALIGN with the following possibilities:

  left¦right¦top¦texttop¦middle¦absmiddle¦baseline¦bottom¦absbottom

  The attributes of WIDTH and HEIGHT actually provided a helpful addition to HTML by allowing the browser to download text and format the text around the space an image would occupy without having to wait for the actual image to finish downloading. Netscape also added the attributes of BORDER, VSPACE, and HSPACE to provide some relief around the image instead of having it in contact with surrounding text.
-
: Netscape added the attribute CLEAR, suggesting that a line break would continue down past an image to the left margin of the window.

- <NOBR>: Strings of characters between the opening and closing tags would not be wrapped by the browser.
- <FONT SIZE=*n*>: *n* could be an integer from 1 to 7, either positive or negative; 3 was considered the default, allowing text sizes to be larger or smaller than the default. Another element, BASEFONT, was introduced to set the base font size in the document as something other than the default.

Among the problems created by these additions were the lack of default values specified for many of the elements, such as HR—what was the default size, and what amount of increase in size did each increase in value suggest? Attribute values in SGML are not case sensitive, yet Netscape proposed having ordered lists be lettered, and it distinguished between the values TYPE=A and TYPE=a. IMG has no default alignment in HTML, and Netscape did not specify one as appropriate behavior when no attribute was given. Netscape also provided for characters in the attribute values that SGML required to be quoted, for example, +, -, and %. Netscape, however, ignored this requirement. Case sensitivity and violation of the SGML TOKEN rules meant that Netscape's additions could not be validated under SGML rules. Worse for people trying to make a living in the new World Wide Web, the onslaught of new additions meant that customers demanding the latest and trendiest features could never be satisfied. For those Webmasters attempting to maintain some compatibility with non-Netscape browsers, the Netscape additions often resulted in displays that were unreadable in Mosaic or Lynx.

With Version 2 of the Navigator, Netscape added the capability to run Java applets and a scripting language named JavaScript, along with several HTML 3.0 tags. The final release of Version 2 was followed the next month by the first release of Version 3.0b1, which added plug-ins and support for color backgrounds in tables. Other beta versions followed, incorporating the FACE attribute for FONT, and support for columns and spacing. The final release of 3.0 was in August 1996; version 4.0 saw its initial release in December of that year. Version 4.0b1 introduced the LAYER tag, which is an attempt to control the layout of elements of a Web page. Version 4.0b2 added support for cascading style sheets and Netscape's proprietary JavaScript style sheets. LAYER has been proposed to the W3C for acceptance as a standard for layout control.

NOTE

Debates raged in the newly formed newsgroup comp.infosystems.www.authoring.html. It was often stated that attempts to control appearance of a page using HTML were "doomed to failure." In the early 11th century, Canute was king of England, Norway, and Denmark. People began to say that King Canute was divine, which was a heresy. To prove his subjects wrong, Canute had his throne taken to a beach at low tide and placed near the water. He commanded the tide to remain at ebb. As the incoming tide reached him sitting in his throne, he made a kingly retreat to his castle, having no need to prove further his lack of divinity. Although people looked to the foundation of the World Wide Web Consortium (W3C) to bring commercial proprietariness to a halt, the commercial tide continued, and it was the W3C that retreated to the castle.

The Sleeping Giant Is Awakened: Microsoft

Microsoft's Bill Gates had been saying that the Internet had insufficient bandwidth to become viable as a consumer marketplace. When he changed his mind, Microsoft turned on the proverbial dime and within a month made the Internet an extension of the Windows desktop. Microsoft started MSN, an online content provider with Internet access, provoking complaints to the United States Justice Department by competitors because Windows 95 loaded with an icon for MSN on the desktop for users to link to automatically.

Microsoft licensed Mosaic from Spyglass, rewrote some of the appearance features, and released it as the Internet Explorer. Microsoft quickly refined its version of Mosaic, making it more Microsoft and introducing the Microsoft way of doing things, such as using tags proprietary to the Internet Explorer. As time passed, more and more proprietary features were added, without the blessing of the W3C. Microsoft introduced its answer to `BLINK`, the relentlessly scrolling `MARQUEE`, background sounds that played over the computer, new controls for `IMG`, new attributes for background images, new coloring schemes for tables, ActiveX controls, and on and on.

Version 1 of the Internet Explorer was followed in only a few months by Version 2, which added support for tables and additional HTML 3.2 tags. In another few months, Internet Explorer 3.0 alpha and beta versions were released, adding proprietary tags for table specifications, frames, support for scripting (JScript, a Netscape JavaScript-compatible language, and VBScript, based on Microsoft's own Visual Basic), and limited support for style sheets. Version 4 was announced in May of 1997 and provides support for proprietary dynamic HTML, which Microsoft has implemented and proposed to the W3C as a standard.

Both Microsoft and Netscape joined the W3C, and their competition entered a new arena. They each proposed competing additions to HTML, but implemented them unilaterally without approval. Each pledged to support all approved standards. The W3C was overwhelmed.

The World Wide Web Consortium

During a workshop in July of 1993, Berners-Lee proposed founding a consortium to handle the problems inherent in establishing the standards for HTML and HTTP. Berners-Lee and CERN were not the proper entities for this development. It was suggested that the consortium establish a reference model for the standards, test proposals to determine compliance, and provide a stamp of approval.

Charter of the W3C

In 1994, the W3C was formed to attempt to bring some much needed order to the world of HTML. Founded as an industry consortium, W3C has as its goal developing common standards for the evolution of the World Wide Web. Members of the consortium may propose

standards for adoption; once adopted, those standards are to be implemented by members in their software, whether server or client. The W3C is attempting also to promulgate protocols that are faster and more efficient in the use of the Web for images, sounds, and video.

The Work of the W3C

With both Netscape and Microsoft on the W3C, that body became the battleground for the hearts and minds of the World Wide Web. The charter of the consortium to bring order to the growth of HTML and HTTP protocols took a backseat to the commercial interests of some of the consortium's members. Although many had predicted that bringing Netscape into the fold would force it to give up its renegade ways, the tide of commerce inexorably flowed into the conference room, lapping at the feet of the conferees. A retreat to the castle was crafted: HTML 3.2 was proposed in lieu of 3.0. Level 3.2, being a compromise, pleased no one, but it bowed to reality and incorporated formerly proprietary Microsoft and Netscape tags. That round of the debate of HTML for appearance or content was given to appearance.

With HTML 4.0, known informally as Cougar, the W3C has begun again to attempt to set standards that will divide the markup of content from the appearance of a Web page. As discussed more fully in Part III, "Extending HTML 3.2," Cougar proposes the separation of physical styles from content markup by more reliance on style sheets. With both Microsoft and Netscape involved in the discussions, we can hope that browsers will finally implement style sheets. Many see that as the best way to suggest appearance while still allowing full use of HTML as an SGML-compliant method of bridging the chasm between mind and knowledge as the level of data increases.

Summary

A combination of three factors made Berners-Lee the creator of the Web. First, he worked during a time when computers were powerful enough, small enough, and cheap enough to be used as the infrastructure of the World Wide Web. That was chance. The second factor was the ability to see beyond a particular need and a particular solution to that need; to see instead the realization of a great theme reaching down through time to the right moment with the right tools; to see instead an overarching answer to a question that was unasked at the time, but which had been asked and unanswered for generations. By chance, Berners-Lee was at the right place and time to have the tools lacking to Vannevar Bush. Berners-Lee had Bush's insight, but Berners-Lee also had the means. The third factor, however, is what distinguishes Berners-Lee from many great thinkers and idealists: Berners-Lee saw the answer and put it into effect. Instead of talking and writing about his concept, Berners-Lee went about the daily grind of using what existed, inventing what did not, writing software, drafting proposals, and coordinating the efforts of a team to put his idea into working order and to get it used. We must not neglect the team that worked with him. I have mentioned Dan Connolly, Karen Muldrow, and David Raggett, but there were many others who made major contributions of time, thought, and effort.

I cannot overstate the importance of the theories that preceded Berners-Lee, especially those of Bush and Nelson. It is not incorrect to say that this is another case of vision increased by standing on the shoulders of giants. But Berners-Lee then got down off their shoulders, dug the foundation, laid the footings, and began bricking in the walls of his vision. As a result of this founding work, the World Wide Web spans the seas, linking not countries, but people who are now empowered for the first time in history with the ability to publish work with a worldwide audience without having to own the printing presses and the means of distribution.

CHAPTER 3

SGML and the HTML DTD

by Dmitry Kirsanov

IN THIS CHAPTER

SGML is the substratum on which HTML was conceived and, therefore, is responsible for many of HTML's strengths and weaknesses. SGML stands for Standard Generalized Markup Language; this is a formal system designed for building text markup languages. It is not a markup system by itself, however; think of it as a programming language to build working programs (HTML being one of them) rather than a program by itself.

In this chapter, you'll learn the foundations of SGML to see how (and why) HTML was built on top of it. It's very instructive and engaging to trace the roots of the language and explore the conceptions of its creators. In fact, you can't say you know HTML unless you're at least sketchily acquainted with its SGML heritage.

We'll analyze the definition of HTML in terms of SGML, consisting of SGML Declaration and document type definition, both for HTML Version 4.0. You'll see what valuable information can be elicited from these formal constructs and how they can be used for authoritative reference on HTML topics. You'll also learn why an HTML document should conform to a DTD and how to ensure this using a validation service.

Knowledge of basic SGML concepts and syntax will provide you with a solid foundation for mastering HTML and will help you understand some of the peculiarities of the language. The goal of this chapter, however, is not to teach you SGML or how to write SGML applications, but to show you how understanding SGML may aid in learning and applying HTML.

Procedural and Descriptive Markup

Any document can be thought of as consisting of *content* and *markup*. The content is relatively straightforward; it bands together all the characters of text, images, and the rest of the meat that delivers the message of your document. However, if you just add together all this stuff you won't get a readable document. You need to mark it up—that is, to introduce some new information into it that couldn't be automatically deduced from the content.

In fact, even a plain ASCII text, often cited as an example of pure text without any formatting, contains a fair amount of markup. For instance, it is markup that allows you to determine what the width of text column in a plain text file is, or where the boundaries between paragraphs lie. Of course, the inventory of markup instructions (called *tags*) that you can apply is determined by the format of your document and the tools you use to work with it.

So what is the purpose of markup? In other words, what information can, and should, be conveyed by a document beyond its content? This extra information can be divided into two groups: *structure* and *formatting*. Structure tells us how the document logically breaks down into parts (paragraphs, sections, chapters) and how these parts are hierarchically organized, from the document itself at the very top to the atoms of content at the very bottom. Formatting (in a broad sense) governs presentation of the document: which fonts to use to display it, in what tone of voice to read it aloud, how to break it into pages for printing, and so on.

Here enters the Great Markup Controversy that was one of the main driving forces behind the creation of SGML and whose backwashes are still disturbing the HTML community. Its essence, without the fear of oversimplification, can be reduced to the question of which of the two markup types is more important and should be given priority.

Obviously, many contemporary text processing systems are heavily biased toward marking up formatting aspects of a document in the first place (this sort of markup is often called *procedural* or *presentational*). The reason is obvious: What an average user needs most often is a formatted document, not a structural diagram of its parts.

However, the bitter truth is that procedural markup may actually impede using and reusing the document if not accompanied, and even preceded, by proper structural (often called *descriptive* or *generic*) markup. For instance, if a document file contains instructions to set a line of text in Times Roman, size 12 pt, and left-aligned, but never hints that this line is a heading or a figure caption, then this markup is very restrictive.

You won't be able to change formatting of all headings in a document at once. You'll have difficulty exporting the document into another format or another medium (such as a voice synthesis system), which may be using different means of formatting headings. You cannot even automatically generate a table of contents. To put it simply, unless you know what this part of text is supposed to *be*, information on how to *render* it is of very limited value.

On the other hand, after you have added information about the logical role of every element in the text, the natural next step is to attach the presentational markup to these logical tags rather than actual parts of the text. Now you don't have to specify font size for a heading in your text anymore; it is enough to mark it up as a heading, and the rest is taken care of automatically (provided that someone has associated certain formatting parameters with the structural heading tag).

This concept, called "separation of presentation from content," is the major advantage of all systems that put descriptive markup first. When separated, both content and formatting can be developed by different people and modified much more easily. Thus, one of the roles of descriptive markup is to serve as an intermediate layer separating content and formatting of a document.

It should be admitted that text processing systems that are in common use now (such as office word processors) do not completely ignore the benefits of descriptive markup. The named styles that you apply to paragraphs in, say, Microsoft Word, represent some sort of descriptive markup units with certain formatting tags associated with each of them. Moreover, users can create new styles as needed for their documents. This provides for a certain level of separation of presentation from content.

However, such a solution is only partial because style tags do not impose any restrictions on the structure of your document. For example, it's no problem to assign a heading style to a

paragraph inside a figure caption or a footnote—which is pretty much senseless. Also, you are in no way discouraged from making direct changes to text attributes, such as font face or size, thereby overriding their values in a style. Styles in word processors are mere containers for presentation attributes and not a means to impose some prescribed structure on a document being created or processed.

You might wonder, do we really need to impose any structure on the document contents? Yes, and here's why: You can't predict what uses will be made of your document tomorrow or in a year, what formats it'll need to be converted to, or what media it'll be put onto. By using a strictly defined set of hierarchical descriptive tags, you ensure that the text can be processed automatically without any need to manually disambiguate cases such as a heading inside a footnote. I could say that descriptive markup reveals the immaterial soul of a document so that any program or person can then conveniently incarnate it into a body of choice.

The provision for automatic processing is the advantage that outshines all others. It is difficult to imagine how many resources humankind spends annually on preparation, processing, and interchange of documents. Office computing and desktop publishing software made this work easier, but, in many cases, proprietary and presentation-oriented tools put more handicaps than benefits in the flow of documentation. An open and extensible system of descriptive markup would thus be invaluable in many situations.

To summarize, what we need is a markup system focusing on structure of a document rather than its formatting. It should allow us to build a hierarchy of descriptive tags so that they could serve not only to separate and describe different parts of a document but also to formally prescribe its structure.

An equally important requirement is that the system should provide for easy extension and modification. Ideally, a user should be able to define a completely new set of tags if such a need arises. Finally, this system should not be proprietary; it is important that anyone be free to create and use markup tools based on this system and to produce software implementing these tools.

SGML is the system designed to satisfy all of these requirements, as well as many others. SGML is strictly descriptive and contains no means to mark up presentational aspects of documents. However, SGML can be easily interfaced to external procedural markup systems and style sheets.

It is the customizability area where SGML reveals its real power. In fact, SGML is not a markup system by itself; it is, rather, a *metasystem* enabling users to create such systems for particular types of documents. Its flexible syntax makes it possible to build markup languages (HTML being one of the examples) to match any imaginable demand. Moreover, any single SGML document can be provided with its own "local" markup definitions fine-tuned for the particular purpose.

Just like HTML, SGML is a computer language rather than a data format. This means that you can create SGML files manually in a text editor, although there exist software tools that facilitate the task. A piece of software that reads and analyzes an SGML document (for example, for transformation or validation) is called an SGML parser. A parser by itself, however, is not very useful because of the purely descriptive nature of SGML, so most often a parser is a part of a bigger document processing or browsing application.

A Brief History of SGML

The roots of SGML go back to the late 1960s when the concept of descriptive markup saw the light for the first time. After companies started using computers for document processing, it soon became obvious that a storage format should contain not only formatting codes interpreted by the computer itself, but also descriptive human-legible information about the nature and role of every element in a document.

The first working system that used these concepts was the Generalized Markup Language (GML) developed by Charles Goldfarb, Edward Mosher, and Raymond Lorie at IBM. This system was the direct predecessor of SGML and contained prototypes for many of its major features, such as hierarchical document structure and document type definition. IBM adopted GML and built mainframe-based publishing systems on it that were widely used to produce technical documentation in the corporation.

In 1978, American National Standards Institute (ANSI) started research in the field of generic document markup pursuing the goal of establishing a nationwide standard for information interchange. Later, Dr. Goldfarb joined the ANSI working group, and in 1980 the first draft was published. It was finalized in 1983 as an industry standard named GCA 101-1983.

In 1984, the International Standards Organization (ISO) joined the activity and started preparation of an international version of the standard. The first draft of the ISO standard was published in 1985 and the final version appeared a year later. The standard bears the full name "ISO 8879:1986 Information processing—Text and office systems—Standard Generalized Markup Language (SGML)." The full text of the specification is available from ISO for a fee. (See `http://www.iso.ch/cate/d16387.html`.)

The best use of SGML is generally made in big corporations and agencies that produce a lot of documents and can afford to introduce a single standard format for internal use. Besides IBM, early applications of the new technology include the projects developed by Association of American Publishers (AAP) and the U.S. Department of Defense. Finally, in 1991, researchers at the European Laboratory for Particle Physics (CERN) chose to build a hypertext markup language that they called HTML as an SGML application. But that's another story...

How to Define an SGML Application

From SGML's point of view, a document is a hierarchical structure of nested *elements* (chapters, sections, paragraphs, and so on). SGML has no means—and was not intended to have—for specifying any presentational aspects of these elements. However, strictly speaking, SGML

cannot tell you about the meaning or role of any element, either. This information is implied by the creator of an SGML application and is usually provided in comments or in the documentation accompanying the formal specification.

SGML realizes the maxim of Wittgenstein, who said, "The meaning of a word is its use." In SGML, the only information that can be formally communicated about an element is in what contexts and levels of document hierarchy it can or must occur. This means that you cannot build an interpreter that could apply a meaningful formatting to a document based only on its SGML markup. However, the purely formal dissection that SGML performs on a document is still surprisingly useful in many situations.

All documents that can be marked up with the same hierarchy of elements are said to belong to a certain *document type*. Rather than describe a set of tools to mark up documents, SGML defines the structure of a particular type of documents via what is called document type definition (DTD). A part of this chapter (see the section called "Document Type Definition for HTML 4.0") is devoted to analyzing the DTD for one particular SGML application, HTML 4.0 (code-named Cougar). Besides (and before) the DTD, some general features of an SGML application are specified in another formal construct called the SGML declaration, which is detailed in the next section, "SGML Declaration for HTML 4.0."

As for SGML syntax, suffice it to say beforehand that it is pretty close to the syntax of HTML. You will see that SGML statements, like HTML tags, are enclosed in angle brackets (<>) and contain a keyword or name followed by one or more parameters separated by spaces. The only consistent difference is that SGML statements commonly have the `!` character inserted between the open delimiter < and the statement keyword, for example:

```
<!ELEMENT IMG - O EMPTY -- Embedded image -->
```

You must already be familiar with one type of statement that uses the `<!` syntax, namely comments in HTML documents that are enclosed in `<!--` and `-->`. That's because the comment syntax of HTML is directly borrowed from SGML, where everything within a `<!` statement enclosed in double hyphens (`--`) is ignored by the SGML parser. For example, the words `Embedded image` in the preceding code line are intended as a comment for human readers only.

One more `<!`-type declaration that needs to appear in HTML files is `DOCTYPE`, discussed briefly later in this chapter in the "Public Identifiers" section.

SGML Declaration for HTML 4.0

SGML declaration is a formal construct used to specify some general information about an SGML application and its associated document type. The following sections list and analyze the SGML declaration for HTML 4.0 provided by W3C. (It can be found at `http://www.w3.org/TR/WD-html40/`.

The SGML declaration is contained in the SGML statement, which has the following syntax:

```
<!SGML  "ISO 8879:1986" ...  >
```

The ellipsis here represents the body of SGML declaration, and the string `ISO 8879:1986` is meant to denote the *level* of SGML standard that this declaration conforms to. In our case, this is the original ISO specification published in 1986.

In the body of the declaration, first comes the comment part:

```
--
     SGML Declaration for HyperText Markup Language version 4.0

     With support for Unicode UCS-2 and increased limits
     for tag and literal lengths etc.
--
```

The rest of the declaration body is divided into sections that are described next.

CHARSET Section

The `CHARSET` section of SGML declaration is used to specify the *character set* to be used by the documents conforming to this document type. So what is a character set?

You probably know that the characters that appear on your display are coded inside the computer by some bit combinations, usually *bytes* consisting of eight bits. Unfortunately, different computers and operating systems sometimes use the same bytes to represent different characters on the screen. The most frequent reason for this is that localized versions of programs and operating systems need to represent non-Latin characters of a particular language's alphabet (such as Cyrillic alphabet for Russian).

Thus, to make the SGML document as unambiguous as possible, SGML declaration defines exactly what *character set* it uses, that is, what bit combinations (or *codes*) are allowed within a conforming document and what characters they are intended to mean. To define a character set, you need to specify three things: first, the set of codes used; second, the set of characters represented; and third, the mapping between these two sets.

The set of codes is easy to specify by simply listing these codes in decimal or hexadecimal form. The set of characters, or *character repertoire*, is more tricky. You cannot simply "draw" a character in the specification because the SGML declaration itself is represented by a plain-text file where every character is coded by a bit combination not guaranteed to mean the same on all systems. One possible way to overcome this difficulty is to give a textual description for every character in the repertoire (for example, "CYRILLIC CAPITAL LETTER A").

However, SGML creators have chosen a less complicated way of dealing with the problem. SGML declaration makes use of other character set standards that already have been adopted by standard-setting bodies (mostly ISO) and that can provide us with a full specification of nearly any character in the world. Having made a reference to such a standard, you can then

use character numbers in that standard to clearly identify what character you need for your document's character set. Here's how this is done.

First comes the CHARSET keyword that marks the beginning of the corresponding section. It is followed by the BASESET keyword that contains the name of the character set standard referred to thereupon:

```
CHARSET
      BASESET  "ISO 646:1983//CHARSET
               International Reference Version
               (IRV)//ESC 2/5 4/0"
```

The standard specified here, commonly referred to as "ISO 646," is practically indistinguishable from what is called "7-bit ASCII." Its 128 characters cover all Latin alphabet characters, digits, punctuation, and some special characters. It is the greatest common subset for nearly all character sets in use now, and you're unlikely to find a computer or a program (even a localized version) that uses something other than ISO 646 for its first 128 byte codes.

However, SGML declaration for HTML 4.0 does not use all this character set, but only a certain part of it. The selection is done using the DESCSET keyword:

```
DESCSET
        0   9   UNUSED
        9   2   9
        11  2   UNUSED
        13  1   13
        14  18  UNUSED
        32  95  32
        127 1   UNUSED
```

Here, the target HTML character set that we need to define is divided into subranges, with a clear identification of where characters in each subrange come from. The first number in each line specifies the starting code of the subrange; the second, its length; and the third position is occupied either by a number that identifies the code to start copying characters from the reference standard, or the UNUSED keyword, which means that the characters in this subrange are not allowed.

Thus, the first line in the preceding code means that the codes in the range 0–8 inclusive (decimal) cannot be used within documents conforming to the HTML 4.0 specification. The second line says that, starting from code 9 onward, we borrow 2 characters that are coded 9 and 10 in the ISO 646 standard (in other words, within this two-character range our character set is identical to ISO 646). The next two characters are again unused, and then we take one character with code 13, skip 18 more characters, and so on.

So, we have defined the first 128 characters of the HTML 4.0 character set. However, to specify the remainder of the code table, we have to refer to another standard. The syntax of the CHARSET section allows the specification of as many external standards as needed and the borrowing of characters from each of them (that is, to have as many BASESET/DESCSET pairs as necessary).


```
BASESET  "ISO Registration Number 176//CHARSET
           ISO/IEC 10646-1:1993 UCS-2 with
           implementation level 3//ESC 2/5 2/15 4/5"
DESCSET  128 32    UNUSED
         160 65375 160
```

The previous version of HTML, 3.2, referred to the standard named "ISO 8859-1" or "ISO Latin-1" to define the characters beyond 7-bit ASCII. ISO Latin-1 uses 8-bit codes and, therefore, accommodates the total of 256 characters (coded 0–255 inclusive), with the range 128–255 containing letters with diacritical marks used in different European languages as well as some special symbols (trademark, copyright, fractions, and so on). The first 128 characters of Latin-1 are identical to those of 7-bit ASCII.

However, the need for better support of languages other than English and Western European languages led to the development of a set of provisions commonly referred to as *HTML Internationalization*, initially described in RFC 2070 (which can be found at `http://ds.internic.net/rfc/rfc2070.txt`) and then incorporated into HTML 4.0. One of the key features of the internationalized HTML is the extended character set that makes use of the Unicode coding standard. Unicode uses 16-bit (two bytes) codes and, therefore, covers as many as 65,536 characters, including nearly all national alphabets of the world and hordes of special symbols.

More precisely, HTML 4.0 refers to the ISO standard named "ISO/IEC 10646-1:1993" or simply "ISO 10646," which is a superset of Unicode and generally uses four-byte codes. However, the UCS-2 in the preceding BASESET statement identifies a special mode of ISO 10646, which uses two-byte codes and is in effect indistinguishable from Unicode. All these coding standards and related issues are covered in much more detail in Chapter 39, "Internationalizing the HTML Character Set and Language Tags."

One question that you might have by now, however, needs to be answered immediately. Does the SGML declaration imply that with HTML 4.0, you have to use Unicode for your documents? No, because the *document character set* we're defining is different from the *external character encoding* that the document is in when created, stored, and served over the network. For the external character encoding, you can use any character set standard that is best suited for the document's content. In practice, the only area affected by the document character set as per SGML declaration is numerical character references such as ` ` that must, in HTML 4.0, point to Unicode code positions. Again, for more details on these issues, refer to Chapter 39.

Unicode itself is a superset of Latin-1, because the first 256 characters of Unicode are identical to those of Latin-1. Also, the latter is likely to remain for a long time the most popular choice for the external character encoding of HTML documents. Table 3.1 lists the first 256 characters of the HTML document character set specified by SGML declaration for HTML 4.0.

Table 3.1. The first 256 characters of the HTML document character set as specified by SGML declaration for HTML 4.0.

Character (or its name for nonprinting characters)	*Code (in the form of character reference)*	*Mnemonic Entity (if any)*
UNUSED	&#0;–&#8;	
[tabulation]		
[line feed]
UNUSED	&#11;–	
[carriage return]
UNUSED	&#14;–&#31;	
[space]		
!	!	
"	"	"
#	#	
$	$	
%	%	
&	&	&
'	'	
(	(	
)	)	
*	*	
+	+	
,	,	
-	-	
.	.	
/	/	
0	0	
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	

Character (or its name for nonprinting characters)	*Code (in the form of character reference)*	*Mnemonic Entity (if any)*
7	7	
8	8	
9	9	
:	:	
;	;	
<	<	<
=	=	
>	>	>
?	?	
@	@	
A	A	
B	B	
C	C	
D	D	
E	E	
F	F	
G	G	
H	H	
I	I	
J	J	
K	K	
L	L	
M	M	
N	N	
O	O	
P	P	
Q	Q	
R	R	
S	S	
T	T	

continues

Table 3.1. continued

Character (or its name for nonprinting characters)	*Code (in the form of character reference)*	*Mnemonic Entity (if any)*
U	U	
V	V	
W	W	
X	X	
Y	Y	
Z	Z	
[	[	
\	\	
]	]	
^	^	
_	_	
`	`	
a	a	
b	b	
c	c	
d	d	
e	e	
f	f	
g	g	
h	h	
i	i	
j	j	
k	k	
l	l	
m	m	
n	n	
o	o	
p	p	
q	q	
r	r	

Character (or its name for nonprinting characters)	*Code (in the form of character reference)*	*Mnemonic Entity (if any)*
s	s	
t	t	
u	u	
v	v	
w	w	
x	x	
y	y	
z	z	
{	{	
|			
}	}	
~	~	
UNUSED	&#127;–&#159	
[nonbreaking space]		
¡	¡	¡
¢	¢	¢
£	£	£
¤	¤	¤
¥	¥	¥
¦	¦	¦
»	§	§
¨	¨	¨
©	©	©
ª	ª	ª
«	«	«
¬	¬	¬
-		
®	®	®
¯	¯	¯
°	°	°

continues

Table 3.1. continued

Character (or its name for nonprinting characters)	*Code (in the form of character reference)*	*Mnemonic Entity (if any)*
±	±	±
²	²	²
³	³	³
´	´	´
µ	µ	µ
¶	¶	¶
·	·	·
¸	¸	¸
¹	¹	¹
º	º	º
»	»	»
¼	¼	¼
½	½	½
¾	¾	¾
¿	¿	¿
À	À	À
Á	Á	Á
Â	Â	Â
À	Ã	Ã
Ä	Ä	Ä
Å	Å	Å
Æ	Æ	Æ
Ç	Ç	Ç
È	È	È
É	É	É
Ê	Ê	Ê
Ë	Ë	Ë
Ì	Ì	Ì
Í	Í	Í
Î	Î	Î

Character (or its name for nonprinting characters)	*Code (in the form of character reference)*	*Mnemonic Entity (if any)*
Ï	Ï	Ï
Ð	Ð	Ð
Ñ	Ñ	Ñ
Ò	Ò	Ò
Ó	Ó	Ó
Ô	Ô	Ô
Õ	Õ	Õ
Ö	Ö	Ö
×	×	×
Ø	Ø	Ø
Ù	Ù	Ù
Ú	Ú	Ú
Û	Û	Û
Ü	Ü	Ü
Ý	Ý	Ý
Þ	Þ	Þ
ß	ß	ß
à	à	à
á	á	á
â	â	â
ã	ã	ã
ä	ä	ä
å	å	å
æ	æ	æ
ç	ç	ç
è	è	è
é	é	é
ê	ê	ê

continues

Table 3.1. continued

Character (or its name for nonprinting characters)	*Code (in the form of character reference)*	*Mnemonic Entity (if any)*
ë	ë	ë
ì	ì	ì
í	í	í
î	î	î
ï	ï	ï
ð	ð	ð
ñ	ñ	ñ
ò	ò	ò
ó	ó	ó
ô	ô	ô
õ	õ	õ
ö	ö	ö
÷	÷	÷
ø	ø	ø
ù	ù	ù
ú	ú	ú
û	û	û
ü	ü	ü
ý	ý	ý
þ	þ	þ
ÿ	ÿ	ÿ

NOTE

Note that the commonly deployed symbols of trademark (™), en dash (–), and em dash (—) do not belong to the Latin-1 range and therefore are not included in the table. To access these symbols, you must use their Unicode codes, which yield ™, –, and — correspondingly or their mnemonic entities ™, –, and —.

CAPACITY Section

This section is meant to provide a rough estimate of the system resources (more specifically, different types of memory) that an SGML parser will need to allocate in order to process the DTD. This is not very reliable information, however, because the memory usage is largely dependent on the internal architecture of the parsing application. Most SGML parsers do not take these values into account, and HTML creators simply assigned big enough numbers to these parameters to ensure that processing the DTD won't be aborted because of exceeding one of the CAPACITY values. The CAPACITY parameters are not discussed individually here; you can refer to the SGML specification for details.

```
CAPACITY    SGMLREF
            TOTALCAP        150000
            GRPCAP          150000
            ENTCAP          150000
```

The SGMLREF keyword means that all CAPACITY types that are not indicated here should take their default values from the SGML reference concrete syntax. (See the next section for more on this.)

SYNTAX Section

The next major section of SGML declaration is introduced by the SYNTAX keyword. It is provided to define various syntax features of the SGML application, such as naming rules, delimiter and control characters, reserved names and limits used by the DTD and conforming SGML documents. This syntax is called "application concrete syntax" as opposed to "reference concrete syntax" of SGML itself, which is used in the SGML declaration (but not the DTD, as specified by the SCOPE parameter). As you'll see shortly, in the case of HTML, the differences between these syntaxes are minimal.

SCOPE Declaration

Immediately before the SYNTAX section comes the SCOPE DOCUMENT declaration:

```
SCOPE     DOCUMENT
```

Its sole purpose is to specify that the application concrete syntax to be declared will be used not only by the conforming SGML documents but also by the DTD of this SGML application.

Shunned Characters Declaration

The SYNTAX section starts with the list of shunned characters' codes preceded by the SHUNCHAR keyword:

```
SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
         17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 127
```

"Shunned" doesn't mean "prohibited," and the list of shunned character codes doesn't fully coincide with the UNUSED codes in the character set declaration. In fact, some of the shunned

characters (for example, the carriage return and line feed characters) are outright necessary in any text file, SGML document being no exception.

However, these characters should be used with care as their meaning and usage may depend on the computer environment in which the text is processed (for example, although a text line in MS-DOS and Windows is terminated by a pair of carriage return and line feed characters, UNIX systems use single carriage return). The keyword CONTROLS means that if a particular computer system uses some other characters as control codes (and not displayable characters), these should be added to the SHUNCHAR list.

Syntax Character Set Declaration

Next comes what may be considered a duplicate of the CHARSET section—a BASESET/DESCSET pair defining a character set (see "CHARSET Section," earlier in this chapter):

```
BASESET  "ISO 646:1983//CHARSET
          International Reference Version
          (IRV)//ESC 2/5 4/0"
DESCSET  0 128 0
```

What is the purpose of this additional definition?

The character set defined in the SYNTAX section is used only within that section and nowhere else. This reminds us once again of the fact that any text document, SGML declaration included, is actually nothing but a sequence of codes, and to get to the meaning we need to know which character corresponds to each code. Having provided a separate character set declaration within the SYNTAX section, we can ensure that the syntax definition is completely independent of the document character set (defined in the CHARSET section). In other words, we won't have to rewrite the SYNTAX section when the content of CHARSET section is changed.

Function Characters Declaration

The FUNCTION keyword is used to identify the character codes for so-called *function characters*:

```
FUNCTION
        RE            13
        RS            10
        SPACE         32
        TAB SEPCHAR    9
```

Function characters are special characters that can affect on syntax. All function characters defined here are *separators* whose role is identical to that of a white space. The RE and RS identifiers denote simply carriage return and line feed characters; they are short for Record End and Record Start, respectively. (In SGML, a line in a text file is sometimes termed a *record*, similar in a way to a database record.) TAB (tabulation character) is not recognized as a separator by SGML standard, that is why it is accompanied by the additional classifier SEPCHAR.

Naming Rules Declaration

Next comes the NAMING declaration, which regulates usage of characters in element and entity names and as names' start characters:

```
NAMING   LCNMSTRT ""
         UCNMSTRT ""
         LCNMCHAR ".-"     -- ?include "~/_" for URLs? --
         UCNMCHAR ".-"
```

To facilitate recognition of a name by the parser, the repertoire of characters allowed in the first position of a name is limited as compared to the rest of the name. SGML standard itself allows Latin letters only as name start characters and Latin letters plus digits as ordinary name characters, so here we only need to specify additions to these sets. The characters are specified by using strings in quotes (called *literals*), and separate parameters are provided for indicating uppercase and lowercase character versions in each class.

Thus, the preceding lines tell us that in HTML, only Latin letters are allowed as name first characters (the corresponding parameter strings are empty) while the repertoire of ordinary name characters is extended by the period and the hyphen. These characters are caseless and thus are shown the same in both LCNMCHAR (LowerCase NaMe CHARacters) and UCNMCHAR (UpperCase NaMe CHARacters) parameters.

```
NAMECASE GENERAL YES
         ENTITY  NO
```

The NAMECASE declaration governs case sensitivity of the SGML application concrete syntax. It is further subdivided into ENTITY, which applies to entity names only (for more on entities, see the "Entities" section later in this chapter), and GENERAL, which covers all the rest, including element names. Here's the answer to the question of why <img> and <IMG> are treated the same in HTML while &eacute and É aren't.

Delimiters Declaration

The DELIM declaration allows you to change the character sequences used as tag delimiters in the SGML application.

```
DELIM    GENERAL  SGMLREF
         SHORTREF SGMLREF
```

The values SGMLREF indicate that in this respect HTML syntax is no different from SGML syntax; you use < as start delimiter of an opening tag, </ as start delimiter of a closing tag, > as end tag delimiter, and so on. This part of SGML declaration adds very little information on HTML syntax, so it need not be discussed in detail.

Reserved Names Declaration

The NAMES keyword can be used to change some of the reserved SGML names that will be used in DTD declarations.

```
NAMES    SGMLREF
```

Again, the SGMLREF value indicates that the list of these reserved names is exactly that provided by SGML specification. Many of these reserved names are discussed later in the section on DTD.

Quantity Limits Declaration

The last declaration in the SYNTAX section is the QUANTITY declaration:

```
QUANTITY SGMLREF
         ATTSPLEN 65536    -- Implementors are recommended --
         LITLEN   65536    -- to avoid fixed limits but --
         NAMELEN  65536    -- this is the best we can say here --
         PILEN    65536
         TAGLVL   100
         TAGLEN   65536
         ATTCNT   100
         GRPGTCNT 150
         GRPCNT   64
```

This declaration sets limits for some lengths and counters used by the parser in processing the DTD and conforming documents. Just like in the CAPACITY section, many of these parameters are assigned arbitrary big values that effectively mean "no limit at all;" it is difficult to imagine that one might need, for example, an element name (governed by the NAMELEN parameter) that is 65,536 characters long. Most HTML browsers disregard these limitations (or have their own instead), so the different QUANTITY parameters aren't discussed here.

FEATURES Section

The section of SGML declaration introduced by the FEATURES keyword contains parameters that turn on or off some of the features of SGML syntax; that is, they allow or disallow using these features in the SGML application being defined. These features are divided into three classes: MINIMIZE, LINK, and OTHER. Following the HTML-oriented approach used throughout the chapter, only those features that are turned on in the SGML declaration for HTML 4.0 are considered here.

MINIMIZE Class

The MINIMIZE class contains the markup minimization features that are intended to facilitate using SGML markup and to make it more readable for humans. Minimization features allow you to omit tags and other markup instructions in certain situations where context is sufficient to resolve the resulting ambiguity.

- The OMITTAG YES feature allows the DTD to specify that for certain elements, start and end tags can be omitted. Such an element will be opened or closed based on matching the context against the corresponding content model. (See the upcoming "Elements" section.) The most common example in HTML is the <P> tag, whose closing tag </P> can always be safely omitted.
- The SHORTTAG YES feature is very interesting. In fact, it contains a whole bunch of different features that could save a lot of typing when marking up a document. With SHORTTAG YES, you can use empty open tag <>, empty closing tag </>, type pairs of tags in the form <TAGNAME/.../, omit attribute names, and so on, with all missing information implied by the parser through simple and effective rules. Unfortunately, common browsers do not support these features, so they are mostly of theoretical interest for HTML users.

LINK Class

The LINK class contains features that affect processing attributes of elements. None of these is allowed in HTML.

OTHER Class

The OTHER class contains miscellaneous features that didn't fit into MINIMIZE or LINK classes:

- The FORMAL YES feature indicates that the PUBLIC entity declarations (covered in the "Public Identifiers" section later in this chapter) should use formal syntax of public identifiers to enable automatic substitution of external sources by the parser.

Document Type Definition for HTML 4.0

Now that we've examined the SGML declaration and found answers to a number of general questions about HTML formation, it's time to get to the details of its tags, entities, and the related document structure. All of this is defined in document type definition (DTD) for HTML 4.0. (See http://www.w3.org/pub/WWW/MarkUp/Cougar/Cougar.dtd and Appendix B in this book.)

The HTML DTD analyzed in this chapter is too long to be listed in its entirety. Instead of going through the DTD from top to bottom, I discuss the major concepts and syntax features of an SGML DTD in their logical order exemplifying them by excerpts from the HTML 4.0 DTD. This approach will enable you to understand any given part of the DTD without the chapter being too encumbered.

Entities

Before we start investigating elements that form an HTML document and the tags that delimit these elements, let's discuss another SGML concept named *entities*. If tags can be likened to named styles in word processors, then entities are a direct analog of *macros* that may expand to text strings or markup instructions.

In HTML documents, entities are used to invoke characters that either are absent on a computer keyboard (such as é) or have special meaning and thus cannot be typed directly (such as <). In the DTD itself, as you'll see later, entities play a more important role helping to make all sorts of declarations more concise and readable. The entities used in DTD are called *parameter entities*, as opposed to *general entities* intended for use in HTML documents and not in DTD. These two types of entities are declared in a slightly different manner, as shown in the next three sections.

Parameter Entities

The very first declaration in the HTML 4.0 DTD is an entity that expands into a formal reference (in this case, a URL) of the DTD:

```
<!ENTITY % HTML.Version
 "http://www.w3.org/pub/WWW/MarkUp/Cougar/Cougar.dtd"
   -- Typical usage:

      <!DOCTYPE HTML SYSTEM "http://www.w3.org/pub/WWW/MarkUp/Cougar/Cougar.dtd">
      <html>
      ...
      </html>
   --
  >
```

Let's consider, in this example, the syntax of an entity declaration. It uses the `ENTITY` statement that, like all other SGML statements, requires a `!` after the start delimiter `<`. After the `ENTITY` keyword comes the `%` character indicating that the entity in question is a parameter entity rather than a general entity.

Separated from `%` by one or more spaces is the entity name that is later used to invoke the entity. Note that the name contains a period, thus making use of the `NAMING` section settings in the SGML declaration. (See "Naming Rules Declaration," earlier in this chapter.) Also recollect that entity names are different from element names in that they are case sensitive.

The last obligatory component of an entity declaration is the string enclosed in quotation marks (*data string*) that shows what this entity stands for and what it will expand to when invoked. Here's how the entity we have defined can be used later in the DTD:

```
%HTML.Version;
```

Note that this time, there is no space between the `%` and the entity name. The trailing semicolon may be omitted in certain contexts.

Unfortunately, this part of SGML syntax clashes with one of the Netscape HTML extensions, namely using the `%` character for specifying sizes of images and other elements as percentages of window dimensions. This is why HTML validators that check an HTML document against a DTD sometimes have trouble with this feature. (For more information, see `http://www.webtechs.com/html/mozilla.html#percent`.)

Public Identifiers

The last part of the `%HTML.Version;` entity declaration is the comment that reminds us about the necessity (unambiguously stated in HTML specification) to start any HTML document that is intended to be a valid SGML document with a `DOCTYPE` declaration. This allows an SGML parser to know at once that the structure and tags of the document it's about to process are described in the DTD identified by the string `"http://www.w3.org/pub/WWW/MarkUp/Cougar/Cougar.dtd"`. Of course, HTML (not SGML) browsers could also make use of this information to select the level of HTML support needed for the document (although only a few of them really do).

The use of a URL as a DTD identifier is rather unusual. (It is probably explained by the fact that, at the time of this writing, HTML 4.0 DTD was still evolving.) More often, to refer to external information sources, SGML documents use *public identifiers* of a special form. For example, in HTML 3.2 DTD the `%HTML.Version;` entity expands into the string `-//W3C//DTD HTML 3.2 Final//EN`, which is the public identifier of this version's DTD. Another example is the identifier string of a character set standard used for the `BASESET` parameter in SGML declaration. (See "`CHARSET` Section," earlier in this chapter.) Any DTD or related standard has a unique public identifier assigned in order to allow referring to this standard from other SGML documents. Such references are usually made via parameter entities.

If the data string in an entity declaration is preceded by the additional keyword `PUBLIC`, this means that the string is not the entity value but a public identifier pointing to an external information source. For example, the HTML 4.0 DTD is accompanied by a set of general entities for accessing characters of ISO Latin-1 (as discussed under "`CHARSET` Section," earlier in this chapter) isolated in a separate document (detailed at the address `http://www.w3.org/pub/WWW/MarkUp/Cougar/ISOlat1.ent`) with its unique public identifier. Here's how this document is incorporated into HTML DTD:

```
<!ENTITY % HTMLlat1 PUBLIC
   "-//W3C//ENTITIES Latin1//EN//HTML">
```

Here, the string in quotes contains the public identifier of the external resource whose contents will be substituted for each occurrence of the entity `%HTMLlat1;`. To make the mentioned document part of the DTD, it is now enough to invoke the defined entity (actually, this is done right after its declaration):

```
%HTMLlat1;
```

Formal rules for constructing public identifiers need not be detailed here. A fairly complete catalog of public identifiers can be found at `http://www.webtechs.com/html-tk/src/lib/catalog`.

General Entities

General entities are declared in the DTD similarly to parameter entities, but they have a number of differences:

- A general entity cannot be used in the DTD but only in the documents conforming to this document type (in our case, the HTML documents).
- A general entity does not have the `%` character in its declaration.
- A general entity is invoked using the `&` character rather than the `%` character used for parameter entities—for instance,

  ```
  &lt;
  ```

  As with general entities, the trailing semicolon sometimes may be omitted (although I wouldn't recommend doing this).
- A general entity usually contains the `CDATA` keyword, inserted in its declaration before the data string. This keyword indicates that the string should not be interpreted as

SGML data; that is, any markup instructions it might contain should be ignored and treated as ordinary text characters.

For an example, consider the entity declarations provided in the DTD for accessing four special characters:

```
<!ENTITY amp    CDATA "&#38;"   -- ampersand            -->
<!ENTITY gt     CDATA "&#62;"   -- greater than         -->
<!ENTITY lt     CDATA "&#60;"   -- less than            -->
<!ENTITY quot   CDATA "&#34;"   -- double quote         -->
```

This example also shows us one special kind of entity called *character reference* that does not require any declaration. If the entity opening delimiter `&` is immediately followed by the `#` character and a number, this number is interpreted as character code (from the document character set as defined in the SGML declaration, see "CHARSET Section") and the whole entity is replaced by the character having this code. This is one of the two methods to access characters that are beyond the reach of a computer keyboard; the other method uses the *mnemonic* character entities defined in the DTD, such as `&` or `é`.

You might wonder how the entities in the preceding example could expand to special characters if the CDATA keyword prohibits any SGML instructions, character references included, from having effect in the data string. The answer is that this string is in fact read twice: the first time when the entity declaration is interpreted, and the second time when the entity is used in the document and its data string is substituted. The CDATA keyword affects only the first reading. As a result, the DTD is protected from the special characters, while in the document the references are expanded to the characters intended.

Elements

As I've already mentioned in the "How to Define an SGML Application" section earlier in this chapter, a document marked up with an SGML application is thought of as consisting of a hierarchy of nested elements. A marked up element is usually enclosed in a pair of start and end tags. The ELEMENT statement in SGML defines both start and end tags (but not their attributes) and prescribes what may be the content of this element by defining its content model.

Here's an example of element declaration:

```
<!ELEMENT P - O (%text)*>
```

Here, P is the element name (short for Paragraph). The two characters following the element name are *minimization indicators* specifying whether it is possible to omit start or end tags for this element. The first indicator refers to the start tag, and the second, to the end tag.

In place of a minimization indicator, you can put either a hyphen (-), meaning that the tag is obligatory, or the letter O, meaning that the tag is omissible. Thus, the preceding statement declares that a P element (a paragraph) must be preceded by the <P> start tag, but the </P> end tag can be omitted.

It is possible to have both start and end tags omissible. For example, the declaration

```
<!ELEMENT HTML O O (%html.content)>
```

indicates that both <HTML> and </HTML> tags around the content of an HTML document can be dropped.

Content Model Keywords

The last component in the previous section's element declarations is the *content model* specification. (Here, it is done via parameter entities, and to see what they expand to we should find the corresponding ENTITY statements in the DTD.) Content model declares what can, what must, and what must not go inside the element.

The simplest type of content model is specified by a single keyword from the following list:

CDATA	Stands for Character DATA. This keyword means that the SGML parser suspends its processing for the content of the element. Whatever other tags or entities are contained in the element, they won't have any effect and will be treated as ordinary data characters. The only tag that SGML parser reacts to when skipping over CDATA content is the end tag of the element that switched to CDATA mode. HTML DTD uses CDATA content model for the obsolete elements XMP, LISTING, and PLAINTEXT that were intended for inserting preformatted text into HTML document without the need to escape any special characters. Also, the CDATA mode is used for STYLE and SCRIPT elements whose content is to be processed by external programs rather than SGML parser.
RCDATA	Stands for Replaceable Character DATA. This keyword introduces content model that is only different from CDATA in that it expands all general entities and character references, but ignores markup statements. RCDATA is not used in HTML DTD.
EMPTY	Means that the content of the element is empty. Naturally, this is always accompanied by the permission to omit the end tag. Here is an example: `<!ELEMENT IMG - O EMPTY -- Embedded image -->`
ANY	Allows any markup and data characters within the element. ANY is not used in HTML DTD.

Content Model Groups

Sometimes, however, it is necessary to be more specific in defining content model of an element. This is done via *content model groups* whose syntax deserves a more thorough examination.

The simplest model group is one element name enclosed in parentheses, which means that the element being defined must contain one occurrence of the element specified in content model and nothing else. This is a rather artificial situation, as more often a model group contains two or more element names—for example,

```
<!ELEMENT HTML O O  (HEAD, BODY)>
```

Here, the comma between HEAD and BODY is a *connector* used to indicate the relations between the elements listed. Possible connectors include the following:

- A comma (,) indicates that the elements listed in the content model should both be present within the element exactly in the order specified.
- A vertical bar (|) is the "exclusive or" connector. It indicates that one and only one of the elements can occur. However, it is often more practical to use the "simple or" relation allowing any one, or both, or even none of the elements to be present. This is why | is often combined with the occurrence indicator *, for example:

  ```
  <!ELEMENT APPLET - - (PARAM | %text)*>
  ```

  Here the content model specification says that within the APPLET element, any number of PARAM elements mixed with any number of text fragments (this is what the %text; entity effectively expands to) may occur.
- An ampersand (&) is the "and" connector. It indicates that all the elements listed must occur, but in any order. It is often combined with the ? occurrence indicator. Here's how the DTD defines the %head.content; parameter entity that is later used in content model specification for the HEAD element:

  ```
  <!ENTITY % head.content "TITLE & ISINDEX? & BASE?">
  ```

Here's the list of occurrence indicators used to show how many times the elements can occur in a content model:

- A question mark (?) means that the element may occur either once or not at all.
- A plus sign (+) means that the element may occur one or more times. Here is an example:

  ```
  <!ELEMENT OL - -  (LI)+>
  ```

  This means that an OL element may consist of an arbitrary number of LI elements, but at least one must be present in any case.
- An asterisk (*) means that the element may occur any number of times or not at all.

Model groups can be nested, and the occurrence indicators may apply to an entire group rather than a single element:

```
<!ELEMENT DL - - (DT|DD)+>
```


This means that within a DL (Definition List) element, at least one (but possibly more) DT or DD elements must be present.

Besides element names, you can use the #PCDATA (Parsed Character DATA) keyword in model groups. It refers to "usual" characters of the document without any markup tags and can be used to explicitly allow or disallow plain text within an element.

It is different, however, from the CDATA keyword discussed earlier. First, #PCDATA can be used only within a model group and not on its own as CDATA (that is, #PCDATA should be enclosed in parentheses even when it stands alone). And second, #PCDATA does not imply ignoring markup; if a tag is encountered in the context where only #PCDATA is allowed, a compliant SGML parser should fix an error rather than ignore this tag.

Together with the connectors and occurrence indicators listed, #PCDATA can limit the set of elements allowed inside another element without prohibiting plain text from appearing there. For example, here's how the %text; entity is defined via a number of subordinate classifying entities:

```
<!ENTITY % font "TT | I | B  | U | S | BIG | SMALL | SUB | SUP">
<!ENTITY % phrase "EM | STRONG | DFN | CODE | SAMP | KBD | VAR | CITE">
<!ENTITY % special
   "A | IMG | APPLET | OBJECT | FONT | BASEFONT | BR | SCRIPT |
    MAP | Q | SPAN | INS | DEL | BDO | IFRAME">
<!ENTITY % formctrl "INPUT | SELECT | TEXTAREA | LABEL | BUTTON">
<!ENTITY % text "#PCDATA | %font | %phrase | %special | %formctrl">
```

Thus the %text; entity stands for, in plain English, "either a chunk of text or one of all these listed elements." Obviously, it'll most often be used with the * occurrence indicator. For an example, see how the preceding declarations are used once more to define quite a number of elements in one snap:

```
<!ELEMENT (%font|%phrase) - - (%text)*>
```

As you see here, both parameter entities and groups can be used for specifying element names in declarations, not only in their content models.

SGML syntax also allows notation of the addition or subtraction of model groups, which is very convenient if these groups are specified via entity references. For instance, the FORM element is allowed to contain anything that can occur within a block-level element (that is, an element that starts a new paragraph) except for the FORM element itself (which means that FORMs cannot be nested). Rather than define the new content group from scratch, we can make use of the already defined %block.content; entity by subtracting the single FORM element from it:

```
<!ELEMENT FORM - - %block.content -(FORM)>
```

Analogously, we can sum up two model groups:

```
<!ELEMENT HEAD O O  (%head.content) +(%head.misc)>
```

Attributes

An element is not fully described by its name and content model. Many elements have associated *attributes* that serve to provide additional information for rendering the element. Attributes for each element should be declared in the DTD via ATTLIST statements.

Here's a typical attribute declaration for an element:

```
<!ATTLIST AREA
  shape       %SHAPE    rect      -- controls interpretation of coords --
  coords      %COORDS   #IMPLIED  -- comma separated list of values --
  href        %URL      #IMPLIED  -- this region acts as hypertext link --
  target      CDATA     #IMPLIED  -- where to render linked resource --
  nohref      (nohref)  #IMPLIED  -- this region has no action --
  alt         CDATA     #REQUIRED -- description for text only browsers --
  tabindex    NUMBER    #IMPLIED  -- position in tabbing order --
  onClick     %script   #IMPLIED  -- intrinsic event --
  onMouseOver %script   #IMPLIED  -- intrinsic event --
  onMouseOut  %script   #IMPLIED  -- intrinsic event --
  >
```

Right after the ATTLIST keyword, the name of the element for which we're defining attributes is specified. Next comes several three-component groups, each defining one attribute. The first identifier in each group is the attribute name. The other two specify the type of value for the attribute and its default value, as detailed in the next sections.

Type of Attribute Value

After the name of each attribute in the ATTLIST declaration comes a keyword describing its type. This keyword is usually taken from the following list:

CDATA — Here again, CDATA means that the value of this attribute may be any string of characters (as well as an empty string) and should be ignored by the parser. CDATA is used in situations where it is impossible to force more strict limitations on the attribute value with one of the following keywords.

NAME — This keyword indicates that the value of the attribute is a name conforming to SGML naming rules as defined by the SGML declaration. (See the "Naming Rules Declaration" section, earlier in this chapter.) The following fragment of an ATTLIST declaration is an example:

```
<!ATTLIST META
...
  http-equiv  NAME       #IMPLIED  -- HTTP response header name  --
  name        NAME       #IMPLIED  -- metainformation name --
...
  >
```

NMTOKEN — This keyword is similar to NAME with the exception that there's no requirement to start the name with the name start character. (See "Naming Rules Declaration," earlier in this chapter.) This keyword is not used in HTML 4.0 DTD.

NUMBER — This keyword allows the parameter to take numeric values. The following ATTLIST fragment is an example:

```
<!ATTLIST OL -- ordered lists --
...
  compact     (compact)  #IMPLIED  -- reduced interitem spacing --
  start       NUMBER     #IMPLIED  -- starting sequence number --
...
  >
```

ID — This keyword indicates that the attribute value is an *identifier* satisfying two requirements: First, it is a valid SGML name (as in the case of NAME), and second, it is unique across the document (that is, it cannot be assigned to any other attribute within the same document). This value type is specified for the ID attribute of the style sheets mechanism applicable to the majority of HTML elements.

Besides these keywords, you can specify the list of possible values directly using the group notation that you've already seen applied for model groups in this chapter. Thus, in the preceding ATTLIST declaration for the OL element, the COMPACT attribute may only take as value the character string "compact" or have no value at all, as in the example

```
<OL START=1 COMPACT>
```

which is equivalent to

```
<OL START=1 COMPACT=COMPACT>
```

Here's an example from the DTD with an attribute taking one of three possible values:

```
<!ATTLIST table
...
      align       (left|center|right)   #IMPLIED
...
>
```

Default Value Specification

Finally, for each attribute in an ATTLIST declaration, either a default value is provided or a keyword is specified indicating whether this attribute is changeable and whether it is required. In this position, character strings need not be enclosed in parentheses (although they should be put in quotes if they contain spaces or delimiters), but the keywords require using a # escape character as in the #PCDATA keyword mentioned earlier.

Here's a part of ATTLIST for TH and TD elements showing default values for ROWSPAN and COLSPAN attributes:

```
<!ATTLIST (th|td)                 -- header or data cell --
      ...
      rowspan NUMBER   1          -- number of rows spanned by cell --
      colspan NUMBER   1          -- number of cols spanned by cell --
      ...
>
```

More often, however, you'll see in place of the default value a keyword from the following list:

`#FIXED` This keyword must precede the actual default value and is used to specify that the value cannot be changed by the user. It is used by the DTD only once, in the declaration for `VERSION` attribute of the HTML element:

```
<!ATTLIST HTML
        VERSION CDATA #FIXED "%HTML.Version;"
...
>
```

This means that the only possible value of the `VERSION` attribute is the string substituted for the `%HTML.Version;` parameter entity. (See "Parameter Entities," earlier in this chapter).

`#IMPLIED` This keyword indicates that the attribute is optional.

`#REQUIRED` This keyword indicates that the attribute is obligatory. For example:

```
<!ATTLIST PARAM
        name    CDATA     #REQUIRED  -- property name --
        value   CDATA     #IMPLIED   -- property value --
...
>
```

Deprecated Features

Sometimes, a part of the DTD must be processed in a way different from the rest of it. For this, SGML offers the generic mechanism of *marked sections* that make it possible to isolate any markup statements and declarations in order to control their processing. HTML DTD uses this mechanism to mark its *deprecated features* that should be avoided in documents but are kept in the DTD for backwards compatibility. Here's what a marked section looks like:

```
<![ %HTML.Deprecated [
    <!ENTITY % preformatted "PRE | XMP | LISTING">
]]>
```

The `%HTML.Deprecated;` entity expands into the special keyword that tells the parser what to do with the contents of the section. The two keywords used in various HTML DTDs are `IGNORE` and `INCLUDE`. The `IGNORE` keyword allows you to ignore the marked section completely, and the `INCLUDE` keyword prescribes its contents should be processed on equal terms with the rest of DTD. So, to get a "strict" version of a DTD, all you need to do is to change the declaration

```
<!ENTITY % HTML.Deprecated "INCLUDE">
```

to

```
<!ENTITY % HTML.Deprecated "IGNORE">
```


Other DTDs and Related Resources

You are probably aware that, besides consecutive HTML versions (the latest being 4.0), there exists a number of HTML "flavors" deviating from the standard in the scope of supported features. The most notorious of these flavors is "Netscape HTML", a vague term used to circumscribe the suite of HTML extensions ("netscapisms") introduced by different versions of Netscape Navigator browser and now making their way to other browsers and HTML flavors.

Unfortunately, Netscape HTML extensions aren't officially documented in the form of a DTD. Other companies, as a rule, are more reliable in this respect; for example, the DTD for the version of HTML supported by Microsoft Internet Explorer can be found at `http://www.microsoft.com/workshop/author/ref/ie3dtd.htm`.

There were also independent attempts to provide DTDs for various HTML flavors, including Netscape extensions; one of the best collections can be found at `http://www.webtechs.com/html`. The HTML Pro project at `http://www.arbornet.org/~silmaril/dtds/html/htmlpro.html` attempts to combine in one gigantic DTD all HTML variants and extensions proposed by standard-setting organizations and browser manufacturers.

Certainly, the whole wizardry of DTD syntax would be pointless if there were no programs to automatically parse the DTD declarations. The most sensible purpose of such parsing (as well as of formally defined syntax in general) is to check HTML documents against the DTD to ensure they are valid SGML documents using only declared elements and attributes.

The confusion of HTML flavors notwithstanding, it is always a good idea to make sure that your document is formally correct from the viewpoint of at least one of the DTDs out there, preferably the DTD of the current official HTML version adopted by a standard-setting body. (To watch for latest developments in this area, visit the W3C page at `http://www.w3.org/pub/WWW/MarkUp/` or the home page of IETF at `ftp://www.ics.uci.edu/pub/ietf/html/index.html`.)

Such a validation also can be helpful by ensuring that the document contains no syntax errors such as unclosed tags or delimiters. Validation packages are sometimes combined with functions to check for broken links, estimate download time, examine images, check spelling, and so on.

Some of these validators are accessible over the Internet. The WebTechs validation service at `http://www.webtechs.com/html-val-svc/` is a pure SGML validator without any extras, but it offers a big collection of DTDs to choose from and can check not only HTTP-accessible documents but also HTML fragments entered interactively. The site also offers a handy hypertext version for each of the DTDs it uses. (See, for example, `http://www.webtechs.com/sgml/Wilbur/DTD-HOME.html`.)

Should HTML Continue Developing as an SGML Application?

It is true that one can create an HTML browser without it being an SGML parser (or even its proper subset). Equally true is that the overwhelming majority of HTML users are quite comfortable without the least notion of DTD intricacies. As you've seen in this chapter, a DTD is not very helpful in respect to the meaningful aspects of HTML, being limited to its formal syntax only. So what is the value that SGML adds to HTML development? Is it perhaps time to leave the SGML heritage behind?

Quite a lot can be said to advocate SGML importance. To begin with, SGML is an authoritative international standard that makes an SGML-supported argument especially strong in the modern world torn apart by browser wars and incompatible HTML extensions. After all, SGML has been proving its usefulness during more than a decade while HTML is much younger and, alas, significantly less stable.

SGML has a great potential outside of the HTML arena. Many important SGML applications have been and are being created for various documentation projects around the world. The SGML users community is strong and influential. HTML development can profit only by drawing from the mainstream of SGML philosophy and practice. Examples of promising SGML-inspired developments that may someday change the HTML world include Document Style Semantics and Specification Language (DSSSL), a versatile style language for use with SGML, and Extensible Markup Language (XML), a subset of SGML designed for use over the Internet. (See Chapter 21, "JavaScript Style Sheets and Other Alternatives to CSS," and Chapter 38, "The Emergence of Extensible Markup Language.")

The importance of a clear and unambiguous syntax specification should not be undervalued, either. Not only does a DTD, if present for a specific HTML flavor, give ultimate answers to many syntax-related questions, it also enables automatic checking of HTML documents in a robust and reliable way. I'd say that an HTML version without a DTD is like a language without a dictionary: Not everyone speaking the language needs to consult the dictionary, but those who really influence its advancement will hardly do without a good reference book.

The final, and probably the most important, argument is that it's SGML, not HTML, that was designed to ensure document portability and easy transformability. One of the main SGML missions is to guarantee that the content we create is accessible to everyone in spite of incompatible proprietary technologies. From this viewpoint, the HTML of nowadays with its flavors and browser feuds can hardly be named a deserved heir. However, by sticking to the SGML roots of the language, we can still considerably facilitate automatic handling of HTML files and using them in a diversity of environments, with the final effect of improving longevity of our information.

Summary

In this chapter, you've learned the foundations of SGML, a metalanguage of logical markup, and you've analyzed the HTML DTD, the definition of HTML syntax in terms of SGML. In particular, the following questions were answered:

- What is the difference between presentational and generic markup, and what is the importance of the latter?
- What was the purpose of creating SGML and how did the history of its development progress?
- What information on the properties of HTML can be elicited from the SGML declaration for HTML?
- How does HTML DTD define entities, elements, and elements' attributes, and how do you parse the content model specification for an element?
- Where on the Internet can you find additional information and DTD-related resources?
- What are the benefits of maintaining HTML as an SGML application?

CHAPTER 4

The Structure of an HTML Document

by Philip Stripling

IN THIS CHAPTER

In order to be displayed on the World Wide Web, files must be saved in text format, and they must contain the tags to inform browsers how to render the page on a reader's computer monitor. The insertion of those tags is called *marking up* the document, a carryover from the days of typesetting. When marking up a document, the author must consider the needs of the reader, just as did the typesetter of old, working with molten lead to form each letter of a printed document.

Unlike typesetters printing on paper, however, new authors never know for certain how their Web pages will appear. By using Hypertext Markup Language (HTML), which is platform independent, an author can ensure that the Web page will be rendered in a readable fashion, with no loss of content. In this chapter, I discuss the benefits of platform independence and how to put together a Web page that will be rendered correctly by any brand of browser running on any computer.

The discussion of markup elements, tags, and attributes goes into some detail, especially on those attributes that cause problems for many authors. Although the use of proprietary markup is discouraged, the information is available for you to make your own choice based on your needs. You are encouraged to know the rules of appropriate, platform-independent HTML and to use them to design simple, elegant, readable Web pages.

HTML Documents Are Platform Independent

Hypertext Markup Language is a data format that enables the exchange of information via the World Wide Web. To achieve this, the format must be recognized on all available computing platforms using available software. With platform independence, persons in different locations can exchange information, collaborate on authoring articles, publish data, and carry on a robust discourse without regard to time and space, and without having to pay for specific hardware and software with proprietary schemes.

With nonlinear information systems, it is important to provide users with a consistent view of the incredibly diverse data and data types available through the World Wide Web. Authors creating Web sites must satisfy the needs of users to have reliable, understandable access not only to the nodes of information, but also to the different protocols and applications that may be necessary to provide the data to the user. Authors who mark up text in a manner that limits usable access to their data may unwittingly exclude some persons from beneficial and useful information. In some circumstances, proprietary markup or design for specific window sizes may be perfectly acceptable—on a LAN or WAN where hardware and software are prescribed and set up for all users, for example, or where the purpose of the Web site is to present information to (or obtain information from) self-selecting patrons who will be using certain proprietary hardware or software and who are the target of the site. Authors who wish a wider market, however, should very carefully consider the use of proprietary extensions of HTML that will not degrade gracefully on other hardware or software than the proprietor's and the use of designs that presuppose the existence of a particular size of window for display.

HTML markup is the most hotly debated topic in the conferences of the World Wide Web Consortium (W3C). Because HTML is simple enough for all persons to learn and write, and because browsers that interpret HTML are going to be similar, the makers of browsers have sought to differentiate their programs by creating extensions to HTML that only their programs can interpret and display. This marketing ploy has aggravated many people. HTML is a simple method of presenting data to the world, and it is remarkably easy to learn and remarkably rich in its powers of expression.

By providing markup on a Web site using proprietary extensions or by designing to certain display sizes and resolutions, authors cease to serve the needs of their users and begin to serve the marketing interests of a particular software or hardware company. Nonlinear information systems have been proposed for some time, and several systems were implemented on proprietary hardware or software. They did not succeed because they did not transcend the platform for which they were written. The utility of HTML and the Hypertext Transfer Protocol is that they offer distributed data, wherein anyone can store and publish data on whatever system is at hand. Everyone can safely assume that any other person with any other hardware and software combination can have access to and provide links to other data anywhere in the world using the standard system of markup and transfer protocol. It is this utility that brings the World Wide Web to life, a life which no one person, corporation, or nation can give or take away. It is this utility that prevents the World Wide Web from being defined, limited, and bent to the will of any person, corporation, or nation.

At the moment, the open standards allow anyone to go online, search the World Wide Web, and find all the information that is necessary to mark up a file and publish it as a Web page. Many of the tools for doing this are freely available on the Internet. The standards for markup are public and freely available as well. HTML tutorials and guides are available at Web sites all over the world and as FAQs that are available for downloading as text files via FTP. As a result, the barriers to entry to the World Wide Web are remarkably low. No one has to be a "certified" professional to create a Web page and to make it available on the Web. All of us are free to write and publish Web sites on our own interests and make them available to others. The standards, the tools, and the lessons are free. We do not have to buy a particular brand of software to write the page, and we do not have to own a particular brand of hardware to see the page.

One problem of standards is that growth and evolution may be stunted even by the small bureaucracy of the W3C. Netscape and Microsoft, two companies offering Web browsers, have proposed, and in many cases unilaterally implemented, proprietary extensions to HTML that are useful, some say even necessary. Although not all agree with particular choices, examples of interesting and useful extensions might include color for backgrounds and text, width and height attributes for graphics, frames, and embedded sound files. By having commercial interests push for acceptance of otherwise proprietary extensions to HTML, the W3C may be pushed to consider more proposals and different issues than it otherwise would. On the whole, this tension between the W3C and its commercial members will work to the benefit of users of HTML, so long as the parties remain on roughly equal footing.

During the process of standardization, however, authors are torn between the *demands* of their customers for the latest and trendiest Web sites and the *needs* of their customers for reliable and valid HTML to preserve the value of the content. Prior to HTML 3.2, making Web sites conform to valid HTML was seen as condemning the site to staid, static pages that caused readers' eyes to glaze over, while using proprietary extensions allowed glitz, animation, and placement of text and images, drawing customers in droves. Although there may have been some truth to that, currently accepted HTML includes a great many ways of enhancing appearance and providing the glamour long relegated to the trendier sites that often sported a logo indicating that the site was best viewed with a particular brand of browser. Commercial software companies continue to push proprietary tags, however, and the temptation is for Web authors to comply with their customers' requests to keep up with the competitors using such extensions and, presumably, drawing more hits.

As a Web author, you should know which elements and attributes are "standard" HTML and which are proprietary. Furthermore, you should know the effect of using a proprietary tag and viewing the page with a browser that cannot interpret the tag. If information is lost or rendered in a manner that makes it unintelligible, you do your customer a disservice. If, for example, you use the BLINK element, you may be assured that those browsers that do not interpret that tag will ignore it, but that no information will be lost as a result. On the other hand, if you mark up a page using frames and do not use the NOFRAMES element, you may rest assured that those persons using browsers that do not implement frames will be presented with a blank page.

In addition to being aware of the results of using nonstandard markup, you should be aware that not only do different browsers render the same markup differently, but the same brand of browser on different computer operating systems will render the same markup differently. Thus, in addition to validating your work, you should view (or have your acquaintances view for you) your marked up pages on different platforms with different software. You will not be able to cover all the bases, but you should cover as many as possible, so that your customer is not faced with e-mails complaining that the pages are not readable on a certain computer with a certain browser.

HTML Markup Elements and Tags

This section presents a close look at the elements that make up an HTML document. You will learn the definition of element, how to distinguish an element from its tags, and how to specify the attributes that may be available in certain elements to affect their rendering. Part II, "Basic HTML," provides further information on the use of the elements available.

Elements

If you look at this book, you will find a title page, a table of contents, chapters, and an index; those are some of the elements of the book. With SGML, you define the presentation of those elements when they are rendered by the browser on a computer screen. In HTML, elements are delimited by the tags that are the symbols for beginning an element and for ending it. The

elements of an HTML page are described as containers delimited by those tags. An element is not the tag to which we all refer. It is the device used to present the concepts that you want the document to communicate when someone downloads your Web page and views it.

HTML elements are identified in a start tag that gives the element name and any attributes. (Element and attribute names are not case sensitive.) Tags themselves are delimited by the < and > symbols, as in <HTML>. You use elements to cause the browser to render the concept of a heading, a paragraph, a link, or an image. Although it might be convenient to use the words *tag* and *element* interchangeably, it is important to know the difference.

Some elements, such as anchors, cannot contain other elements. Some elements have only a start tag—such as BR and its tag,
—denoting a line break. Some elements have end tags that may be omitted—such as the paragraph and its tag, <P>. Chapter 3, "SGML and the HTML DTD," contains a full listing of the elements, and Chapter 7, "Structural Elements and Their Usage," contains a full explanation of structural elements and their use.

NOTE

The contents of an element are a string of characters and nested elements (where permitted). Keep in mind that the element is not the tag; it is the content.

I've classified elements into three groups for discussion: document type, comment, and structure. The elements for document type and comment are SGML statements and must begin with <! and end with >.

The DOCTYPE Element

Current HTML requires a document type (DOCTYPE) declaration before the use of any tags. This declaration provides an alert that the document conforms to SGML and specifies the DTD governing the document (which must conform to the DTD stated). A sample DOCTYPE declaration for HTML 2.0 would be

```
<!DOCTYPE  HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
```

This declaration informs the browser that the document to follow is HTML; that the document type definition is public (and, presumably, widely available; private DTDs exist, but the declaration would either have to give the URL of the private DTD where the user agent could access it or enclose it with the HTML document for proper parsing by the user agent); that the public DTD is maintained by the Internet Engineering Task Force; that the DTD is for HTML 2.0; and that the DTD is written in English.

The COMMENT Element

The COMMENT element is also a declaration, so it, too, is opened by <! and terminated by >. However, the COMMENT element itself begins with -- and includes all text up to and including the next --. This is confusing to many people, who have come to expect that a comment is

opened by the symbols <!-- and closed by -->. This belief is incorrect, and the incorrect use of dashes in comments may cause the comments to be displayed by browsers. The COMMENT element may be placed anywhere within the HTML element. Comments will not be displayed by the browser and, because nothing between (and including) two pairs of -- will be displayed, authors may incorporate comments that are several lines in length.

Structural Elements

The remaining elements are the heart of HTML. Used properly, the structural elements will create a page that is interesting, readable, and portable to all browsers, rendered with clarity, regardless of the software and hardware of the reader. *Success.* Used poorly, information meant to be hidden will be displayed, information meant to be seen will be hidden, and the page will be a jumbled mess with the rendering varying from browser to browser. *Disaster.*

I do not attempt to list and describe all the structural elements in this chapter; Part II goes into greater depth on the use of these elements. I do, however, mention briefly some of the elements that cause new authors problems and show how to put a Web page together while avoiding the traps new authors sometimes fall into.

Attributes

Attributes are the parts of elements that provide for author-specified values. Under the DTD specified in the DOCTYPE declaration, some elements may have no attributes. In a properly drafted DTD, those elements that have or can have attributes will have a *default* attribute for those instances when the author makes no declaration. Under the rules of SGML, attributes that have not been declared in the DTD noted in the DOCTYPE declaration should be ignored.

CAUTION

Commercial browsers with proprietary extensions to markup fail to follow this rule as to their own markup. As a result, the author who uses attributes that are proprietary to one browser may have different results in the display of that same markup by other brands of browsers.

Attributes enable an author to suggest characteristics and properties of an element that can be different from other instances of that element. Where they are allowed, attributes must appear in the opening tag of an element. An attribute is generally specified by an attribute name, an equal sign (=), and a value. In some instances, only the name of the attribute is required. The SGML declaration limits the length of an attribute value to 1,024 characters (including spaces).

It is possible to place an image in the body of an HTML document, so let's look at a sample of the markup for that placement to explore the definition of attribute. I don't discuss the specific attributes for the IMG element using this example—just the mechanics of specifying an attribute and its value.

This code line includes the element IMG and three of the available attributes (SRC, ALT, and ALIGN):

```
<IMG SRC="alfa.gif" ALT="This is the &quot;7% cereal&quot; model" ALIGN="left">
```

Each attribute has values. An equal sign follows the name of each attribute (whitespace on either side of the equal sign is allowed). Following the equal sign is the value of the attribute.

The method of stating the value of an attribute is the cause of some confusion. Authors may use a string of characters as the value, but may not use a >, as it will terminate the tag. If the string of characters contains only letters, numerals, periods (ASCII decimal 46), or hyphens (ASCII decimal 45), the string need not be surrounded by quotation marks (single or double quotation marks may be used). It is never incorrect to use quotation marks, however, and it is suggested that they be used at all times to prevent their inadvertent omission in those cases when they are necessary. In our example, it is not necessary to quotation the values of SRC and ALIGN. However, the ALT value contains spaces and a percent sign and so must be quoted. Furthermore, two quotation marks are included in the value for ALT. To prevent premature termination of the value, I have used entity references for those quotes (but I could have used single quotation marks).

Putting the Pieces Together

After this review of the elements, tags, and attributes, you may feel that you are writing in a restrictive environment and that much of your freedom of expression has been curtailed. In fact, the situation is the opposite. By giving yourself rules to follow, you are encouraged to be creative and thoughtful. You will be able to provide content and information to your audience, assured that it will reach the largest possible number of readers. By working within the rules, you will be encouraged to create a document that is not only easy to read, but easy to write and maintain. (If you have plans for permanence on the World Wide Web, maintenance will quickly become important.)

The HEAD

Using HTML 4.0, the HEAD section of your HTML file can contain the following elements:

- BASE
- FRAMESET
- ISINDEX
- META

- LINK
- SCRIPT
- STYLE
- TITLE

CAUTION

Although material within the HEAD section is not normally displayed, failing to enclose text within the elements allowed in the HEAD might confuse the browser; the presence of normal text not within tags or tags not allowed in the HEAD section might cause the browser to conclude that the HEAD has terminated and that the BODY has begun.

The HEAD element of an HTML file contains information about the file itself. The HEAD tags are optional and consist of opening and closing tags—<HEAD> and </HEAD>. The order of the elements contained in the HEAD is not prescribed, so they may be listed in the order the author finds useful. With the exception of TITLE, the elements of HEAD are optional. The information contained within the HEAD is used by browsers in various ways, and all the elements available for use in the HEAD element can cause problems, so they are discussed in some detail in the following sections.

BASE

The BASE element has only the attribute HREF, and it is used to set the base URL of the document. Relative URLs used within the document are resolved using the URL given in BASE. A sample BASE is

```
<BASE HREF="http://www.foo.com/~bar/index.html">
```

The HREF attribute is required, it must contain a fully specified URL, and the URL given must be where the document is located. If BASE is not used, the browser will resolve relative URLs using the URL it used to locate the document. BASE is not required, and it may be annoying in mirror sites where relative URLs then require a browser to go off site to locate images or even to retrieve the page being viewed to move to a named anchor within the page. This is particularly aggravating when the "mirror site" is the author's fixed drive and the computer is not connected to the Internet at the time the browser starts looking for www.foo.com.

FRAMESET

The FRAMESET element is used to divide the browser window into two or more document windows, which are then used to display different files or different parts of one file. These different windows are referred to as *frames*, and their size, shape, and relative locations are controlled by three elements: FRAMESET, FRAME, and NOFRAMES. Every page that uses frames must have a NOFRAMES element to provide content for browsers that do not support the FRAMESET tag. The use of frames is covered in detail in Chapter 18, "Creating Sophisticated Layouts with Frames and Layers."

ISINDEX

ISINDEX was proposed in the very early stages of HTML. Its purpose was to make a document searchable. An author would use <ISINDEX>, causing the user to be presented with a default prompt. The user would then type keywords that would be sent to the server. The server would execute the search and send the results back to the user. If the server did not have a search engine that could execute the search, it failed. The default prompt varied depending on the browser, and the attribute PROMPT was added to allow the author to suggest what was to be entered in the field. Because ISINDEX required a separate search engine, it never caught on. It has been replaced by the use of forms and the Common Gateway Interface.

LINK

The LINK element is a reference to a related document. It has four attributes, HREF, REL, REV, and TITLE. LINK has not been widely used, but with the advent of style sheets, it may prove more popular. Lynx and later versions of NCSA Mosaic used the attribute REV to allow users to send comments to the author of the file via e-mail using the following tag:

```
<LINK REV="made" HREF="mailto:foo@bar.com">
```

REL will likely have wider application in providing links to style sheets:

```
<LINK REL="stylesheet" HREF="style.css" TYPE="text/css">
```

This link refers to an external style sheet that may be used to suggest how the document should be displayed. For more information on style sheets, see Chapter 19, "Introducing Cascading Style Sheets," and Chapter 20, "Cascading Style Sheet Usage."

The attributes REL and REV are confusing. It was originally suggested that REL would be a reference to "forward" relationships and that REV would refer to "reverse" relationships. Suggested uses for REL were to indicate a page that had the relationship of "home," "toc" (table of contents), "index," or even "up." REV, on the other hand, was supposed to be the reverse relationship. For example, if home.html were a document containing

```
<LINK REL="toc" HREF="toc.html" TITLE="Table of Contents">
```

then toc.html could contain the following LINK:

```
<LINK REV="previous" HREF="home.html" TITLE="Home Page">
```

At the time LINK was proposed, the thinking was that browsers could build navigation bars with the information from several <LINK> tags in a document, with users able to select links to navigate a site. (Although in our example relative URLs are given as the values in the examples, the URLs could be absolute and refer to documents located on another computer.) Later versions of NCSA Mosaic provided for the creation of a separate, nonscrolling pane in its browser window where an e-mail icon is located; this pane is created when the page being viewed has a LINK with a mailto: address. Presumably, if development of Mosaic had not been discontinued, other navigation aids would have been available when appropriate <LINK> tags were used.

META

A META tag is referred to in RFC 1866 as "an extensible container for use in identifying specialized document meta-information." What else needs to be said?

Well, perhaps this could be expanded a little. The problem is that META has no strictly defined use and no strictly defined method of being accessed. In fact, access to and use of META information is not required. The META element has no closing tag, and it contains a name/value pair. Among the attributes of the META tag are HTTP-EQUIV, NAME, CONTENT, and SCHEME. CONTENT gives the value of the name/value pair, and it is required when HTTP-EQUIV or NAME is used. HTTP-EQUIV is a problem. Its use is supposed to be directed at the HTTP server containing the page, as in this example:

```
<META HTTP-EQUIV="Expires" CONTENT="Fri, 31 December 1999 23:59:59 GMT">
```

When a page is requested by a browser, the HTTP server can send the contents of the <META> tags as part of the HTTP protocol headers. (See Chapter 5, "Behind the Scenes: HTTP and URIs.") In that event, one of the protocol headers sent for this example would be

```
Expires: Fri, 31 December 1999 23:59:59 GMT
```

The use to be made of this header information is undefined at this time. If a page with an expiration date is cached by a browser, and the page is requested after the date in the expiry, the browser *should* request the page again from its source rather than display the cached version. Generally, however, it is ignored by browsers, leading to some irritation from authors who want to "expire" their pages so that they are never cached.

Several search engines have published information stating that their indexing algorithm uses META information to locate Web pages relevant to queries of their database. By following the instructions given at the search engine Web site, authors can have some assurance that their pages will be returned to people doing searches. Because there is no limit to the number of META elements you can insert in your HEAD, you might do well to read the help pages of several search engines and draft separate META tags that conform to each engine's requirements.

In an effort to assist robots in the indexing of pages, HMTL 4.0 has added several attributes to the META tag. The LANG attribute has been added, both to aid indexers in determining the appropriate language to present the search results and to facilitate the application of pronunciation rules for speech synthesizers. A SCHEME attribute should be used to indicate a method of describing a value. For example, if a book is referenced in a META tag by using the International Serial Book Number, that scheme would be denoted as follows:

```
<META NAME=identifier SCHEME=ISBN CONTENT="0-8230-2355-9">
```

Some of the elements that are used, always with NAME/CONTENT pairs, are the following:

- <META NAME="*author*" CONTENT="*Your Name*"> to give the author's name.
- <META NAME="*keywords*" CONTENT="*A list of keywords for the search engine*">, but be sure to read the help file for each search engine, as there are limits on number of words and on number of repetitions.
- <META NAME="*description*" CONTENT="*Describe your web page*">, as some engines will use that description when your site is listed in response to a query.

Another problematic use of HTTP-EQUIV was promulgated by Netscape. This is the use of "REFRESH", as shown in the following example:

```
<META HTTP-EQUIV="REFRESH" CONTENT="5; URL=http://www.foo.com/">
```

For those using the Netscape browser, this tag will cause Netscape to load the page from the URL listed in the value of CONTENT 5 seconds after some point in time. The "meta refresh" is often used when an author has moved to a new site. The old URL containing a page with a meta refresh tag directs the reader to the new location. A text explanation should be furnished and a link should be provided for those who do not use a browser that will interpret and execute a meta refresh. Others have used meta refresh for "slide shows." A series of pages are displayed on the user's browser with no action required by the user. The problem for authors is that they cannot finally fix the amount of time the transfer will require, and an incoming page called by the meta refresh may interrupt the download of each page in the series.

CAUTION

The point at which the new page begins to load is unknown because of variations in the speed of the server, the speed of transfers over the Internet, and the speed of the user's modem. Thus, meta refresh will begin counting, presumably when the header is received, but the rest of the page being transmitted for display may be delayed in reaching the browser. If the page contains an image that takes longer to load than the allotted time, the new page will arrive and begin loading prior to the browser having fully rendered the page containing the meta refresh.

Meta refresh should be used with restraint and with caution. In addition to the problem of full transfer of the pages containing the meta refresh, users may have trouble going back to the page. If that is the introductory page to a series, and the meta refresh is very short, your readers may not have time to bookmark it. Meta refresh should be used, if at all, only with some thought as to the benefit of refresh. If your page has moved, leaving the user with a meta refresh does not encourage bookmarking the new page and deleting the old one.

As an aid in indexing its pages, resulting in wider access, the W3C recommends specifying the language of the document and providing keywords and a description of the contents. If different versions of the document are available in different languages, they can be referenced using LINK with the REL attribute.

SCRIPT

The SCRIPT element was incorporated into HTML to accommodate scripting languages promulgated by commercial browser developers. There must be an opening <SCRIPT> tag and a closing </SCRIPT> tag, with the script placed between them. (Scripting is discussed in Part V, "Associated Technologies and Programming Languages.")

Scripting allows for considerable interactive content in a Web page. An advantage of scripting over CGI is that the processing of the script is done on the user's computer, not on the computer serving the page. In addition to conserving server processing, it enables persons to have dynamic pages posted to servers that do not allow Common Gateway Interface activities on the server.

With this power comes problems. Scripting languages are not universally understood. Some browsers do not recognize *any* scripts and will not execute them. The SCRIPT element is to ensure that *current* browsers hide the contents between the tags. Older browsers will display the contents if the script is not also enclosed in a comment. Yet another problem arises, however, if the script contains > or --. Some browsers interpret these characters as the end of the comment, resulting in the display of the parts of the script that follow.

The attributes of SCRIPT are as follows: TYPE (giving the media type specifying the scripting language: TYPE="text/tcl"), LANGUAGE (giving the scripting language; TYPE is preferred), and SRC (giving the URL for an external script; when SRC is used, the contents of the SCRIPT element should be ignored).

Web pages can contain more than one SCRIPT element, and they can be located in the BODY as well as in the HEAD. Scripts can be written to modify the document as it is being parsed—generating a table of contents, for example. Such scripts are, in effect, CGI scripts without server processing. Great care in crafting such scripts will be required, as will careful testing.

The use of the SCRIPT element might require the NOSCRIPT element in certain cases. NOSCRIPT has opening and closing tags, and its content will be rendered by browsers that do not support scripting or the language used by the SCRIPT element. Authors should never use the SCRIPT element as the sole element of a page without also using NOSCRIPT to provide content to users of browsers that do not support scripts or that have script support turned off for security reasons.

CAUTION

Browsers that do not recognize the SCRIPT element will ignore the tag. The result is that the scripts placed between the tags will be displayed as plain text in those browsers, and the script will not be executed. Remember also that upon finding plain text within a header, some browsers assume that the HEAD element has been terminated. Avoid unexpected results by placing scripts within a comment to prevent older browsers from displaying them.

STYLE

The STYLE element is another method of using a style sheet in an HTML document. STYLE has opening and closing tags, and it has the attribute TYPE. The LINK element discussed earlier will import style information from a separate style sheet, while the STYLE element allows the author to suggest presentation within the document itself. This has the advantage of requiring the download of only one document, and it has the further advantage of working when the server on which the page resides has not yet been configured to recognize the style sheet MIME type and cannot properly transfer an external style sheet. The syntax for a document-level style sheet is the same as for external style sheets, and the language used must be given in the TYPE attribute. (See Chapters 19 and 20.)

CAUTION

Browsers that do not recognize the STYLE element will ignore the tag. The result is that the style information entered between the tags will be displayed as plain text in those browsers, and the styles suggested will not be implemented. Remember also that upon finding plain text within a header, some browsers will assume that the HEAD element has been terminated. Avoid unexpected results by placing the style information within a comment to prevent older browsers from displaying them.

TITLE

Although the <HEAD> tags are optional, current standards of HTML require a TITLE for your file. Although the contents of the HEAD section are not to be displayed, the TITLE element is often picked up by browsers and displayed in the title bar of the window. Some browsers use the TITLE for their list of bookmarks, and search engines often display the TITLE in their results, making a useful and informative title helpful for your readership. The TITLE element may contain text and escaped entities, but not markup. The <TITLE> tag requires a closing tag; it has no attributes, and it is to be used only once in the HEAD of the document.

The BODY

The BODY element contains the text and other information that (it is hoped) will be displayed to the reader. BODY has opening and closing tags, but they may be omitted. If you have used the HEAD element and closed it properly, browsers should assume that termination of the HEAD element implies the opening of the BODY element. Do not rely on this assumption, however, as not all browsers properly follow the DTD.

The structure and appearance of the text within BODY are suggested by the elements that may be nested within the BODY element. The elements that may be contained within the BODY can be divided into block formatting elements and character formatting elements.

Block-Level Formatting Elements

The block-level formatting elements include the following:

Element	*Function*
ADDRESS	An address for further information.
BLOCKQUOTE	A quotation of several lines from another source.
CENTER	A centered division.
DIV	A logical division that can be rendered in a certain manner.
FORM	An area providing for interaction with the user.
H*n*	Headings, where *n* is an integer from 1 through 6. Headings are supposed to be used to indicate comparative importance of information through a hierarchy of prominence in the display of the heading, with H1 being the most prominent and H6 the least.
HR	A horizontal rule; that is, a line across the width of the page. (There are some attributes available to suggest different sizes and alignments of the line.)
P	A paragraph container.
PRE	For the display of text in a given format.
SPAN	For the application of a certain style to text that has no structural role or established rendering. For example, assigning <BIG> and <I> to the first three words of each paragraph could be done by defining such a style and then wrapping the words within <SPAN> and </SPAN>.
TABLE	For the display of text in a grid pattern.
Lists	For the display of items in lists, there is DIR, DL, MENU, OL, and UL; some of the list elements can contain subelements for list items (LI), definition terms (DT), and definitions (DD).

These elements imply the closing of the preceding paragraph, although DIV does not imply the whitespace usually associated with paragraph breaks. With the exception of HR, block-level elements should be considered containers, requiring opening and closing tags. The proposed use of the horizontal rule is to indicate a change in topic. With the exception again of HR, block-level elements can contain character formatting elements.

Character Formatting Elements

These elements are used to mark up text and should be contained within block formatting elements. Character formatting elements may be subdivided into logical markup, physical markup, and other markup, a category to pick up the remaining text-level elements.

The next sections distinguish between logical markup and physical markup, a distinction that may be dismissed superficially as mere semantics. The use of logical elements where possible, however, has advantages for authors wishing to assure hardware and software independence, and it allows the readers to define the method of display so that it is suitable for their needs.

Logical Markup

Logical markup is the preferred method of defining the characteristics of text where possible. Logical markup elements include

- CITE to indicate a citation
- CODE to render computer language source code
- DFN to define a term
- EM to emphasize text
- KBD to mark text that the user is to enter on a keyboard
- Q to indicate inline quotes
- SAMP to indicate a sequence of literal characters
- STRONG to indicate strong emphasis
- VAR to indicate a variable used in computer code

The advantage of logical markup is that it can be used to define the meaning of the marked text. By defining certain characters as CODE, KBD, or VAR, you can give a reader information about the content of the text. Readers can configure their browsers to display those semantic elements so that they are in a particular font or a particular color and quickly scan the text to locate items of particular interest or to know which elements may be ignored. A similar situation often arises when authors use a program to write HTML and that program displays the elements in a different color and font style from the text portions of the document. Such devices are very helpful in separating markup from explanatory text, giving a quicker understanding of the meaning of the text and a quicker way to locate and resolve problems. Using logical markup enables your readers to have your text rendered with the same helpful clarity.

Physical Markup

Writers and publishers in the print world have long used italics and bold to make text stand out. The appearance of such text can add meaning within the context of the document. Using markup that renders text in italics, for example, is not a contradiction of the SGML theory of marking for structure and content when that tag is used to indicate foreign words or magazine titles. The use of such physical markup to add emphasis, however, is discouraged.

The following elements are used for physical markup:

- B to display text as bold
- BIG to display text in a larger size than the default
- I to display text in italics
- SMALL to display text in a smaller size than the default
- STRIKE to display text with a horizontal line through the characters
- SUB to display text as a subscript
- SUP to display text as a superscript
- TT to display text in a monospaced font
- U to display text as underlined

Remember that readers may be using equipment that is incapable of rendering physical markup; the monitor might not be capable of showing superscripts and subscripts or different font sizes, for example. Because markup that cannot be rendered will be ignored, physical markup should be used only where its absence will not change the meaning of the text.

Other Formatting Elements

Although this section is labeled "Other Formatting Elements," some may consider the following as the most interesting and useful elements. Here are the "other" elements:

- A is the anchor for links.
- APPLET is the container for Java applets.
- BASEFONT is the tag to set default font sizes.
- BR indicates line break.
- FONT is the tag to modify font display.
- FORM is the tag to allow user input for a variety of purposes.
- IMG is the element to display images and animation.
- MAP is the element to display client-side image maps.
- OBJECT is the new, more powerful element to display images, animation, and applets.

Several of these elements cause problems for beginning authors.

BASEFONT

This tag is used to suggest a default font size. Its only attribute is SIZE, where the value must be an Integer from 1 through 7; the default size is 3. Beginners often use BASEFONT to set a larger-than-usual size, assuming that the large text will seem more important or exciting.

BASEFONT has only its opening tag, and there is no closing tag. BASEFONT affects normal and preformatted text. BASEFONT cannot be closed; therefore, it then affects all subsequent normal and preformatted text. Different browsers interpret what is "normal text" differently, however,

and authors may find that the tag is ignored in certain nested elements that vary from browser to browser. Proprietary attributes other than SIZE are rendered by some browsers, but not others. Because of the differences in the treatment of the element and its attribute, its use is not recommended. To have text rendered in different sizes, the elements BIG and SMALL are recommended. To cause text to be emphasized, consider EM and STRONG instead. The heading level elements of H*n* should be considered where appropriate to indicate comparative importance of information. Remember also that some search engines index Web pages according to the six heading elements. BASEFONT is ignored in determining the index of your document.

BR

Under normal circumstances, the user's browser will wrap text to fit within the user's window. On some occasions, an author may wish to have text forced into shorter lines than the window, but without the extra space between lines usually created by the paragraph element. A common example is an address:

Joan Smith
123 Boulevard
Orleans

The BR element has been described as the specification of a line break between words. Text with a BR inserted should end at the
 and continue at the beginning of the next line, with no whitespace between the two lines. Under that description, more than one
 in a series is undefined, as the succeeding line breaks would not be "between words." Undefined tags are to be ignored, so only the first
 should be given effect, and some browsers render it so. Because the use of multiple
 tags will not create whitespace for all browsers, BR's use for that effect should be discouraged. The BR element has no closing tag.

FONT

The FONT element is a container requiring opening and closing tags. The attributes are SIZE and COLOR. SIZE sets the font's size for the text delimited by the tags. SIZE can be set to absolute sizes using the integers from 1 through 7, or it can be set to relative sizes with signed integer values from –7 through –1 and from +1 through +7; signed integer values must be quoted. FONT will override BASEFONT settings if the absolute size has been set with FONT, or FONT will use the BASEFONT SIZE value to determine the relative size if a signed integer has been used.

COLOR will set the color of the text. Colors may be set in RGB hexadecimal notation (#RRGGBB) or as a color name, using one of the 16 named values available. The named value is not as widely supported as RGB notation, and neither is broadly supported at this time. In addition, text may be inadvertently set to the color of the user's background, causing the text to disappear.

It is possible for users to override the background color set by an author, but not the FONT color; even though you have set the background, you cannot depend on the text color being readable. FONT size cannot be overridden, and the size you have chosen may be unreadable to the user for

a number of reasons—it did not scale well in the user's font, it is too small, it is too large and causes wrapping problems. The use of FONT is discouraged for the same reasons as the use of BASEFONT.

Summary

If you are creating a page for your personal pleasure, go wild. Make the background a pattern, make the text the same shade as the background, use huge animation files that take forever to download, and make it all <BLINK>. You have wonderful choices to make, and you should try them all. This book explains how to create Web pages using the incredibly powerful tools not only of HTML, but style sheets, CGI, JavaScript, Java, ActiveX, and VBScript. You will learn how to create forms, do tables, and incorporate images, video clips, animation, and sound. You will be able to create background images, background sounds, and colored text of different fonts and sizes. Go for it.

However, if you are creating a page or Web site for a customer or for more serious purposes, and you want to incorporate all the trendy alternatives available on the World Wide Web, I have two words for you: Please don't.

When you are creating a page for the world to see and read, know the rules of standard HTML. Follow them to allow full accessibility to your page and the information it contains. Know the rules of standard HTML, but break them when you know what you are doing and what the results are likely to be. Let your failure to follow the standards be purposeful, not ignorant.

Create a page and a Web site with the simple elegance enforced by rules. You have been given a standard with many elements, and you may create a Web page in which the elements are unified and interrelated within the page and within a larger Web site. You may use the elements to create a page that is logically organized and logically displayed by readers who may come to your page from anywhere in the world and who may see your page as written in a foreign language. Use HTML to make your page easy to read.

As you read this book and apply its lessons, be constantly aware that your work will be published on the World Wide Web. It will be read by people who will not have particular hardware and software available to them. Your work may have valuable or entertaining information, and you should not restrict the access to your work. The World Wide Web rests on the notion that data should be freely available to all, regardless of platform. If you want to use proprietary markup, you are free to do so, but you should be aware of the consequences.

CHAPTER 5

Behind the Scenes: HTTP and URIs

by Philip Stripling

IN THIS CHAPTER

The World Wide Web is a nonlinear information system providing users with the facilities to search for and retrieve information sources anywhere in the world. With a uniform naming scheme for locating resources, recognized protocols for retrieving them, and a markup language supporting links within the documents, users have access not only to text-based documents, but also to still images, moving images, PostScript files, sound files, and more data formats than those that currently exist. The World Wide Web has no bounds; it has grown far beyond being the answer to the problem at CERN that it was intended to address.

By implementing the World Wide Web over the Internet as a public standard, Tim Berners-Lee provided for minimization of user effort through the use of URIs, embedded links, and consistent naming conventions. Both HTTP and URIs provided for platform independence to a remarkable degree, in part through the use of TCP/IP and in part through the use of hexadecimal encoding so that URIs can pass through all operating systems and still be read by the client and the server.

Uniform (sometimes called *universal*) *resource identifier* (URI) is the term used generically to identify *uniform resource locators* (URLs), *uniform resource names* (URNs) and *uniform resource characteristics* or *citations* (URCs). URIs are used to name, find, and retrieve information across the far reaches of the World Wide Web. A URL is just an extended filename; it tells your Web browser not only the name of the file you want to view, but also where it is and how to get it. For example, the URL for a hypertext file on another computer could be specified as

```
http://www.foo.com/-bar/file.html
```

The URI can specify any recognized protocol.

This chapter closely examines HTTP and URIs, and it shows how the two work together to enable cross-platform multimedia information exchange. You also learn about some of the problems in the implementation of HTTP and about proposals for the next generation.

Hypertext Transfer Protocol

HTTP is not just for the transfer of hypertext documents. It is a protocol for the transfer of data formats between the server and the client (usually a browser). The data formats include plain text, hypertext, images, sound, and any other data formats that have been specified as MIME types, including proprietary formats such as Adobe Acrobat files. In addition to the data itself, HTTP transfers *meta-information* about the data.

HTTP operates over the Internet, where communication occurs between computers via TCP/IP. *TCP/IP* stands for *Transfer Control Protocol/Internet Protocol*. Understanding HTTP requires some understanding of how TCP/IP works, so this section covers how transfers take place over the Internet.

In the Beginning...

Legend has it that the origins of the Internet lie in the need of the United States Department of Defense for a command-and-control system capable of surviving global thermonuclear war. This need was difficult to satisfy because different branches of the DOD had different computer systems and different network protocols, and no way existed to communicate between the networks. TCP/IP was introduced to allow a network of the networks with full communication between the nets, and with no set path in the event that global thermonuclear war destroyed parts of the network.

NOTE

Some people might not remember the debates on whether the United States could win a nuclear war, whether it could survive a nuclear war, and whether it was moral to think about winning or surviving a nuclear war. I recommend the novels *Dr. Strangelove, or How I Learned to Stop Worrying and Love the Bomb* (by Peter Bryan George), and *Fail-safe* (by Eugene Burdick and Harvey Wheeler). If you don't want to read about that time, you could rent the videos of the movies made of these two novels. (Interested persons might also want to research the suit and countersuit filed by each party alleging plagiarism—but that is another story. The time of the birth of the Internet was an interesting time in which to live.)

In those olden days, computers were mainframes, and they were always turned on and always on the Net (unless, of course, they crashed or were down for maintenance). The protocols developed for communication presumed the presence of many large computers that were completely self-sufficient. Among the early services run in the family of TCP/IP protocols were *ftp*, *rlogin*, and *mail*. With ftp, a user at a terminal to a mainframe started the program called `ftp` and used it to log on to a distant computer. The ftp session maintained its connection while the user transferred files to (using `put`) or from (using `get`) the distant computer. ftp required logging on with a known username and password, and the protocol handled any problems arising from the receiving computer having character sets and line endings different from the sending computer. The ftp session was terminated by logging off (the preferred method) or by inactivity for a certain period of time.

rlogin, now usually known as *telnet*, allows a person at one computer to log on to a distant computer and have the same access that would be allowed to a person logged in to a terminal directly connected to the distant computer. A username and password are generally required, and termination occurs by the same two methods used for ftp.

Computer mail allows a user to send a message to another computer for storage in a second person's mail file. Logging on is not required, and the second person need not be logged on to have the message received and stored.

The Internet now has personal computers, workstations, minicomputers, and dial-up Internet service providers (ISPs) connected. Not all the devices are turned on and functioning at all times, and in the case of ISPs, a user might not even be connected to the ISP when e-mail is received, for example. In these cases, the user calls on another computer for the provision of services in a client/server model of operation. The server provides Internet services for the computers that have access to the server. The client/server model still relies on the protocols of TCP/IP for the transfer of data across the network of networks with all the different operating systems that might be attached at any given point.

TCP/IP

NOTE

Under the Internet Protocol, all computers with full-time access to the Internet are assigned a four-digit number. This IP number is 4 bytes, but, by convention, each byte is translated into a number between 0 and 255, and the bytes are separated by periods. The IP number is also referred to as a *dotted quad*.

The Internet is made up of hundreds of thousands of computers connected to networks, which in turn are connected to each other. The computers and connecting cables are the hardware, and the Internet Protocol is the method by which communications over the hardware are effected. It is the Internet Protocol that gives us dotted quad addresses: `123.456.789.123`. The domain name service provides a translation service so that we have to remember only `www.foo.com` instead of the four sets of digits with dots between them. Under the Internet Protocol, the route of delivery of data is not predetermined. Assume that a person at the Stanford Linear Accelerator starts her Web browser and connects to the Web page at CERN. When data is transferred from CERN in Switzerland to the Stanford Linear Accelerator in California, the packets of data do not travel in a straight connection, nor do all the packets travel along the same path. There are many networks, switches, and routers along the way, and any of these could be fully occupied or inoperative. IP ensures, however, that the packets will find alternative paths to the final destination. IP is called a *connectionless protocol* because it does not require a straight connection between sender and receiver; indeed, IP presumes that no particular path will exist during the transfer and that the data will pass through a dozen different networks on its way to the destination. (Remember that this process is the result of the Cold War and U.S. fears of a Soviet preemptive nuclear strike.)

Note that the user in Stanford does not log on to the computer at CERN. Her browser sends out a request for the page over the Internet, and the request is answered by the CERN computer sending the file back to her computer in packets. Each packet sent by CERN is addressed to her computer, but because the network does not know of the connection between any two packets, the order in which they arrive will be random, and they must be reassembled in order. This is handled by TCP.

Transport Control Protocol is the means by which a connection is established between the client and the server. When the server is ready to send data to the client, TCP breaks the file up into packets, each with an address and a number to allow reassembly in the correct order. TCP hands the packets to IP, which sends them out. The recipient uses TCP to reassemble the packets in order and to request retransmission of any missing packets.

TCP at the receiving end keeps track of packets by *demultiplexing* each packet. The information TCP needs in order to do this is contained in headers sent along with each packet and put there by the sending computer's version of TCP. Each computer maintains a conversation during the transfer, acknowledging receipt of packets, requesting retransfers of packets not received after a certain lapse of time, and maintaining flow. The combination of TCP/IP ensures, for the most part, that packets can be routed around congested nodes and that lost packets can be retransmitted; data loss is remarkably small.

As with HTML and HTTP, the TCP/IP is *open*. They are public standards supported by the Internet Engineering Task Force; the standards are published and freely available online. No commercial vendor owns the standards, and no payment or other fee is exacted for their use.

HTTP—A Brief Overview

HTTP uses TCP/IP to transfer files over the Internet. One of the results of the use of TCP/IP is that HTTP is *stateless*. When a user requests a file by, for example, selecting a link, HTTP sets up a connection between the requesting computer and the sending computer via TCP/IP, it requests the file, the sending computer sends it, and the connection is then dropped. This has the advantage of not tying up Internet resources and the sending computer while the user reads a lengthy file and admires the tasteful rendering of the Web page. There is no "memory" of the transaction for either computer; if the user requests the page again (and it has not been cached), the sender will send it again, along with all its contents.

The basic transactions supported by HTTP are the connection, the request, the response, and the termination of the connection. The connection is the establishment of the TCP/IP connection over the Internet. This is a client/server mode; in some instances, such as a `POST` (which is covered in the "`POST`" section of this chapter), the roles of client and server might switch. HTTP specifically allows and provides for both sides of the connection to assume whichever role is necessary during a connection. Assuming that a person at Stanford has started her browser and selected her bookmark for a Web page at CERN, the browser will cause TCP/IP to establish a connection with the computer at CERN. In this example, the computer at Stanford is the client, and the computer at CERN is the server. It does not matter what brand of computer or what operating system is in use under the protocols of TCP/IP and HTTP. When it is notified that the connection has been established, the client (browser) sends a request to the server. This request includes certain information; the minimum request identifies the HTTP version as HTTP/0.9 and requests the contents of the Web page as text/html. The section "The Request" covers the additional information that can be included in a request. The response from the server will consist of the file requested if the request was a minimum request, or it will

consist of a status line, header, and the response data if the request was under HTTP/1.0. When the response has been sent and acknowledged, the connection is terminated, generally by the server.

When our user at Stanford finishes rendering the Web page, she can then follow links on the page, and this entire set of transactions is repeated. If the followed link is on another computer, the transaction is initiated with this other computer as the server. The differences between HTTP/0.9 and HTTP/1.0 are outlined in more detail later, but I should mention here that under HTTP/0.9, no information is required to be transferred other than the request (which contains GET and the URL) and the response (which contains the Web page). The client transfers no information about itself to the server, and the server retains no trace of the transaction. This stateless protocol is in accordance with TCP/IP and was an intentional design of the drafters of HTTP. It has, however, led to some dissatisfaction among users, browser makers, and the Internet Engineering Task Force. You'll learn about the reasons for this dissatisfaction and proposed solutions in the section "HTTP/1.1 and HTTP-NG," later in this chapter.

HTTP—A Detailed Examination

For those who want to use the Common Gateway Interface or scripting languages to extend HTML and provide interactivity on their Web pages, a more detailed knowledge of the rhythm of connect, request, respond, and terminate will be helpful. This section examines what goes on in this rhythm of HTTP in greater detail than the hobbyist requires, but it will be valuable for those of you who want to generate pages on the fly. As I discuss the details of the protocol, bear in mind that the identity of the client and the identity of the server may vary at any time. When I use the word *client*, I mean the application that establishes the connection and sends the request. When I use the word *server*, I mean the application that accepts the connection in order to respond to the request. Many applications are capable of playing both roles, but this discussion is limited to the particular example and takes no account of an application's other capabilities.

The Connection

Let's assume that the HTTP communication is initiated by a Web browser and is a request for the service of a certain file on a server. The request might travel directly between the programs or, more likely, travel over several networks, through routers, switches, nodes, and gateways. Generally, the connection will be made through TCP/IP connections on the Internet. Currently, the protocol requires that the connection be established by the client before sending the request and that the connection be terminated by the server after sending the response. In the real world, the connection can be closed by either party due to failures of either program, user action, or automatic timing out of the connection due to inactivity. CGI interactions should take into account the possibility of premature termination.

If our person at Stanford has a connection to the Internet and has selected a bookmark in her browser with the URL `http://www.cern.org/pub/highenergy.html`, a connection is made via TCP/IP to the computer known as `www.cern.org`; when that connection is established, her application will send a request.

The Request

The nature of the request depends on the version of HTTP in use. The version must be stated in the request as `HTTP/1.0` or `HTTP/0.9`. If no version is stated, version 0.9 is assumed. If you are writing a CGI program that expects to send a Full-Request or to receive a Full-Response, `HTTP/1.0` must be included or received in the request.

HTTP/0.9

HTTP/0.9 understands only a Simple-Request, which is

```
GET [Request-URI] CRLF
```

where `Request-URI` is the relative path to the entity requested and `CRLF` is a carriage return/line feed, which is the HTTP end-of-line marker. There must be a space between `GET` and the `Request-URI`. In our example, the request will be

```
GET /pub/highenergy.html
```

with the CRLF not shown but present. Note that `GET` is issued to the path relative to the computer to which the connection has been made.

If an HTTP/1.0 server receives a Simple-Request, it *must* respond with a Simple-Response (which is described later in this chapter, in the section called "The Response").

HTTP/1.0

An HTTP/1.0 application must generate a Full-Request; an HTTP/1.0 server receiving a Full-Request will then generate a Full-Response. Instead of a `GET`, the HTTP/1.0 Full-Request begins with a Request-Line such as this:

```
Method [Request-URI] HTTP/1.0 CRLF
```

Following the Request-Line are the request header fields. Those fields are `Authorization`, `From`, `If-Modified-Since`, `Referer`, and `User-Agent`. The following sections cover these in greater detail.

The Request-Line

HTTP/1.0 offers a great deal more flexibility than HTTP/0.9, and one of the reasons is the availability of methods other than `GET`. The method specified can be `GET`, `HEAD`, `POST`, or an extension. Note that HTTP/1.0 must be specified in the Request-Line. There must be spaces between the method, the Request-URI, and the specification of the HTTP version. The Request-Line must end with a carriage return/line feed.

Allowable methods are established by the target resource. The client should be notified in the response if the method is not allowed or not recognized.

GET

The GET method retrieves the data identified in the Request-URI. If the data is a Web page, the file constituting the page will be returned to the client, generally as `text/html`. If the GET points to a script or other process (rather than data), the result of the process will be returned. CGI programmers must be sure to include the necessary headers as part of the response.

If the request includes an `If-Modified-Since` header, GET requests that the file be transferred only if the date given in the `If-Modified-Since` field is earlier than the modified date of the file. This is called a *conditional* GET.

HEAD

The HEAD method requests that the server should respond only with the information contained in the HTTP headers, generated by the META elements that may be contained in the file. The server is not to transfer any data contained in the BODY element of the file. This method is suggested for obtaining the meta-information from the file for use by search engines for indexing purposes and for use in testing links.

POST

The POST method requests that the server accept data enclosed in the request as new data to be included in the file identified in the Request-URI. POST was designed to allow for adding data to existing resources; posting messages to newsgroups, mailing lists, or other similar groups; and transferring a block of data as the result of submitting a form to a script or other data process.

POST does not require that a file be created on the server. It often is merely the transfer of data from a form and can result in the search for and display of existing data. The response to such a POST should be either `200` (OK) or `204` (No Content), depending on whether the response will include any resulting data. If POST creates a file on the server, the response should be `201` (Created) and contain an HTML file noting the successful creation of the file and containing a link.

The data to be transferred in the POST is considered an Entity-Body. Therefore, POST requests must contain a valid Content-Length header; if the server cannot determine the length of the request, it should generate a `400` (Bad Request) error message. The Entity-Body is covered in the section "The Entity," later in this chapter.

Request-URI

The Request-URI is a uniform resource identifier that identifies the file or data to which the request refers. An absolute URI is required when the request passes through a proxy. In all other cases, the URI must give the path to the URI relative to the server. The path of the URI cannot

be empty. If the bookmark of our Stanford user were to `http://www.cern.org`, the path is the server root, denoted as / in UNIX parlance but *not* required for the HTTP protocol. (See the section "The File Path" for more information on the slash.)

CGI programmers should keep in mind that the Request-URI is transmitted as an encoded string, and arrange for the encoding or decoding of the URL as appropriate. See Chapter 27, "CGI Programming."

The Request Header Fields

The client can include information in its request in addition to the method.

Authorization

Occasionally, you might receive an error code `401` (Unauthorized) response when you try to access a Web page. The `Authorization` field is the companion to that response. A browser might authenticate itself by including the `Authorization` request header field with its request. The authorization process is determined by the server, and failure to provide the appropriate credentials in the `Authorization` field of the request will result in another `401` response. The authentication method is determined by the users. Note also that this is an instance where the roles of client and server are temporarily reversed.

The authentication scheme must be arranged before it is used. A server that receives a request for a file that has been identified as protected must send back a response with a WWW-Authenticate header (which is a challenge). This header must contain certain information in an arranged format. When the user agent receives the response with this header, it must generate a new request with an Authentication header containing the agreed upon response to the challenge issued by the server (the credentials). If the credentials are not within the defined protection scheme, the server must issue a `403` (Forbidden). The means for implementing authorization are not specified in the protocol. If you are interested, read RFC 1945, section 11, for additional information.

NOTE

`Authorization` is not secure and should not be relied on to prevent access to any information that should be kept private.

From

The `From` request-header field contains the Internet e-mail address of the person running the user agent. This information can be given by a person when setting up an Internet browser, along with other information requested by the software to establish the preferences of the person using the program. I use the term *user agent* at the beginning of this paragraph because software other than browsers roams the Internet. Robot agents should include this field so that problems can be reported to the person responsible for the robot's actions.

If-Modified-Since

I have already mentioned the use for this field—the conditional GET. If the file being requested has not been modified since the time specified in this field, a cached copy of the file might be used, and the file itself will not be transferred to the client. The proper response in that event is 304 (Not Modified). The field should be constructed as follows:

```
If-Modified-Since: Mon, 31 Dec 1999 23:59:59 GMT
```

When constructing a response to the conditional GET, the CGI programmer should bear in mind that a date later than the current time according to the server is invalid, and the If-Modified-Since field will be ignored. If the requested file has been modified since the If-Modified-Since date, the response is as if the request were a normal GET. If the resource has not been modified since the If-Modified-Since date, the response should be 304 (Not Modified).

Determining the date of modification is sometimes a problem. On some servers, it might be the file system last-modified time. If the file contains dynamically included parts, as is likely with CGI scripting, the date of modification might be the most recent of the last-modified times for the included parts. The format required for the date is discussed in the "Date" section, later in this chapter.

Referer

The Referer is the URI from which the Request-URI was obtained; that is, if our person at Stanford clicked on the link to CERN while viewing a page located at http://www.leland.stanford.edu/slac/lowenergy.html, the Referer field will contain that URI. This field was intended to allow for optimized caching and tracing of mistyped or invalid links, but its use has raised concerns about revealing private information about personal interests.

If the Referer is on the same client as the Request-URI, a relative URI should be given; if the Referer is from another IP address, the absolute URI should be given. The field is not sent if the Request-URI was obtained from a source not having its own URI, such as keyboard input or bookmark.

User-Agent

The User-Agent field usually contains the brand of the browser. This field is useful for determining protocol violations, but it is usually used to determine the popularity of particular brands of browsers. The User-Agent field can also be used to tailor a dynamic response to particular browsers, such as presenting text pages to Lynx users and animated graphics and scripts to Netscape and Internet Explorer browsers.

The Response

Now that you have learned about the request side of the rhythm of HTTP, let's look at the partner's response to the request. After receiving the request message, the server must determine the version of HTTP being used by the client. If it is HTTP/0.9, the only response allowed is the Simple-Response. The client sends the BODY element of the page. The Simple-Response is terminated by the closing of the connection by the server. If the client has sent an HTTP/1.0 request, but receives the BODY and not the Status Line, the client should presume the response to be a Simple-Response and treat the contents as an Entity-Body.

Full-Response

A Full-Response begins with a status line and one or more of the following: General-Header, Response-Header, and Entity-Header. The headers will be followed by an empty carriage return/line feed and then the BODY of the page.

Status-Line

The Status-Line of a Full-Response will contain the protocol version, a numeric status code, and a textual phrase associated with the status code. Each element of the Status-Line must be separated by a space; no carriage return or line feed is allowed until the end of the sequence:

```
HTTP/1.0 200 OK
```

The carriage return/line feed is present but not shown. This line alerts the browser that the response is a Full-Response, not a Simple-Response.

Status Codes and Reason Phrases

The Status-Code element is a three-digit number sent in response to the request, alerting the browser to the success or failure of the request. The Reason-Phrase is a simple text explanation, which can provide some information for a person trying to make sense of what went wrong.

The 100 series of numbers is not used. The remaining series are as follows:

Status Code	*Reason Phrase*
200	OK
201	Created
202	Accepted
204	No Content
301	Moved Permanently
302	Moved Temporarily
304	Not Modified
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable

General Header Fields

These headers are not restricted to either a request or a response, and they can be generated by the client, the server, or both, depending on circumstances at the time of the connection.

Date

`Date` is a general header field, and its contents should give the date and time of the origination of the message. The date format for HTTP is specific; if the format is incorrectly stated, the `Expires` field should be considered to have passed its date and time of expiry. The allowable date formats are as follows:

Sat, 01 Jan 1999 00:01:01 GMT

Saturday, 01-Jan-99 00:01:01 GMT

Sat Jan 1 00:01:01 1999

The first format is preferred. All HTTP/1.0 date and time indicators used within the protocol stream must be in Universal Time, formerly known as Greenwich Mean Time.

It is recommended that servers always send a `Date` header for use by the client in evaluating expiries of cached material. It is suggested that clients not include a `Date` header unless the message includes an Entity-Body (as in the case of a `POST` request). Messages without a `Date` header should be assigned one by the receiving application if the message will be stored in a cache or sent through a gateway using a protocol requiring a `Date` header.

Pragma

`Pragma` is a general header field, and its contents should give directions for optional behavior within the protocol. The instructions in the `Pragma` header are implementation specific, and they can apply to any recipient within the request-response flow of traffic. The format for a `Pragma` header is

```
Pragma: [One or more pragma-directives] CRLF
```

A `pragma-directive` can be `no-cache` and can also include other extensions that are specific to a protocol. If the `no-cache` directive is used, the receiving application should send the request to the server even if the receiving application has a cached copy of the file containing the `pragma-directive`.

Because `pragma-directives` can apply to any or all recipients in the flow of traffic between the client and server, they should be passed through proxies and gateway applications. It is not possible to address a pragma-directive to a specific recipient, and it should be ignored by a recipient if the directive is not relevant to that recipient.

Response Header Fields

The response header fields provide additional information about the response. The fields include `Location`, `Server`, and `WWW-Authenticate`. The `WWW-Authenticate` field was discussed earlier in this chapter in the section "`Authorization`."

Location

The `Location` field specifies the exact location of the file identified in the Request-URI. For response codes in the 300 series, the location must give the URL for automatic redirection; only one absolute URL can be specified. Here is an example:

```
Location: http://www.foo.com/~bar/here.html
```

Server

The `Server` field provides information about the software used by the server to handle the request. Here is an example:

```
Server: CERN/3.0 libwww/2.17
```

The Entity

The file being transferred in response to the request is referred to as the Entity. It comprises Entity-Header fields and an Entity-Body.

The Entity-Header Fields

The Entity-Header fields contain optional information about the Entity-Body or other resource identified in the request. The Entity-Header fields are `Allow`, `Content-Encoding`, `Content-Length`, `Content-Type`, `Expires`, and `Last-Modified`.

Allow

The `Allow` field lists the methods supported by the resource identified by the Request-URI. If the request generating the response uses the `POST` method, the `Allow` header is not allowed; it should be ignored. Thus, the methods allowed by `Allow` are `GET` and `HEAD`. The client can attempt other methods, but a properly crafted program will follow the suggestion made by `Allow`.

Content-Encoding

The `Content-Encoding` field indicates whether additional encoding (other than media-type encoding) has been applied to the file. `Content-Encoding`, for example, alerts the browser to the fact that a document has been compressed; the browser might then be able to handle the decompression itself or call upon a helper application to extract the file, rather than merely saving the contents to the would-be reader's disk.

Content-Length

The `Content-Length` field indicates the size of the Entity-Body in a decimal number of octets. Here is an example:

```
Content-Length: 1066
```

NOTE

Octet is a byte made up of 8 bits. Using *byte* was considered, but some operating systems define bytes as being composed of other than 8 bits.

Content-Type

The `Content-Type` field indicates the media type of the Entity-Body. Media types are open: There is no set number, limit, or definition of media types; there is only a method of describing them. This allows you to include media types as they are conceived; the number is boundless.

Media types are described as *type/subtype* (with no spaces allowed) and can have parameters. An example is `text/html`. Type, subtype, and parameter names are not case sensitive; however, the values of parameters can be. Media types are registered with the Internet Assigned Number Authority, and the use of nonregistered media types is discouraged.

Expires

The `Expires` field indicates the date and time after which the file should be invalid. It is suggested that such files should not be cached; if a cached file is found in response to a request after the date in the `Expires` field, a fresh copy should be requested. Many Web authors know, however, that this field is frequently ignored by caching applications. See the "`Date`" section, earlier in this chapter, for formatting requirements.

Last-Modified

The `Last-Modified` field indicates the date and time when the file appears to have been last modified. The problems with determining when a file was last modified, especially if parts of the entity have been created dynamically, have already been discussed.

The Entity-Body

The Entity-Body is the file that is being requested. It is referred to as the Entity-Body because it can be a text file, a video clip, a PostScript file, an archive of compressed files, or an executable program; its exact format is unknown at the time the request is made. The `Entity-Header` fields give the format in the response. An Entity-Body is sent with the request message only if the request method requires it, and a `Content-Length` header field must be included in the request headers.

An Entity-Body must not be returned if the response is `204` (No Content) and `304` (Not Modified). Responses to the `HEAD` method must not include an Entity-Body. All other responses must include either the Entity-Body or a `Content-Length` header field of zero.

Content-Type

The Entity-Body must carry with it a `Content-Type` field, which identifies the media type of the body. Designation of media type has been discussed in the section "The Entity Header Fields."

NOTE

If the media type is not given, the receiving program should examine the contents of the file and any name extension in an attempt to determine the media type. If the media type cannot be determined, the receiving application should treat the Entity-Body as "application/octet-stream." The person using a browser should have the program configured either to alert the user of that media type and ask for further instruction or to deal with that media type in a specified manner.

Content-Encoding

If the Entity-Body has been encoded, the `Content-Encoding` field should be used to indicate the type of additional encoding applied. Examples of additional encoding include compression of a file and translation of an 8-bit binary file to 7 bits for transfer over the Internet.

Content-Length

The length of the Entity-Body must be included in the `Content-Length` header field. If this field is not sent through error or because of a Simple-Request, the receiving application will determine body length as the total received at the time of termination of the connection by the server.

If an Entity-Body is included in a request message, the message *must* contain a `Content-Length` header field; brute force determination of body length by termination of the connection is not possible with a request, because the server would not be able to respond. If a request is sent with an Entity-Body and no (or an invalid) `Content-Length` header, the response should be `400` (Bad Request).

HTTP/1.1 and HTTP-NG

To address some of the issues raised by statelessness, the World Wide Web Consortium (W3C) and others have been looking at the Hypertext Transfer Protocol and its weaknesses. HTTP opens only one connection per request and returns only one resource per request. Typically, a link is selected on a Web browser, the connection is established, and the server sends the Web page selected; the page, however, is likely to have a few icons, an image, an animation, a sound file, and so forth. The browser requests each object in turn, with a single connection, a single request, a single response, and termination. Although more than one request can be made at a time, a new connection must be established for each request, and the rhythm of request, response, and termination begins.

HTTP/1.1

HTTP/1.1 proposes a persistent connection, where the client and server maintain their connection until all objects have been transferred. It is estimated that half the packet traffic under HTTP/1.0 is opening and closing connections. HTTP/1.1 also introduces pipelining, where

the requests are issued in batches rather than serially (and waiting for the response/termination cycle). With multiple requests being sent in the same TCP packet, the number of packets sent is significantly reduced. Pipelining is the major factor in increasing the efficiency of HTTP/1.1 over HTTP/1.0.

HTTP-Next Generation

The authors of HTTP-Next Generation (HTTP-NG) have the opinion that the constraints of HTTP and MIME are too great to be repaired. They propose to replace HTTP with a new generation of Web transport. HTTP-NG allows for multiple requests to be sent over a single connection, without the requirement of waiting for a response before sending a new request. As is the way on the Internet, the server need not respond to requests in the order in which they are received, and the responses can be interleaved.

HTTP-NG divides the connection into channels and sends control messages over a control channel and each object over its own object channel. This allows mixing protocols, by the way. After a connection is established, objects can be transferred over the connection in a channel using a protocol specifically suited to the transfer of that object. A test implementation has been set up, and early reports are that HTTP 1.0 used only a tenth of the bandwidth available for the test, but that HTTP-NG used the entire path bandwidth using multiple requests over one connection. Multiple requests over individual connections using HTTP 1.0 congested the available bandwidth and performance was degraded badly.

HTTP-NG is not compatible with HTTP/1.*x*. Its proponents are confident, however, that it can be implemented without loss of access to data residing on servers still operating under HTTP/1.*x* and without cutting those servers out of the part of the Web using the new generation of the protocol.

MUX

There is also an experimental protocol for multiplexing transport over the Web (although it is generic and not restricted to the Web in any of its workings). MUX proposes to allow multiple protocols to be multiplexed over the same connection. This allows the server to send text, video, sound, and other files during the same connection, using the protocol best suited for each object. MUX also allows for files to be sent compressed, with an alert to the client of the method of compression. Because MUX works under HTTP, it does not obsolete current standards.

Uniform Resource Identifiers—An Overview

Uniform (or sometimes *universal*) resource identifiers are strings of characters that identify the precise location of a resource available on the World Wide Web. URIs encompass not only the familiar uniform resource locators (URLs), but also uniform resource names (URNs) and uniform resource citations (URCs). URNs and URCs are discussed further in the "A Few Problems with URLs, and Some Proposals" section of this chapter. The syntax of a URI must not

only encompass the currently available protocols for accessing data on the Internet (ftp, http, and gopher, for example), but it should also be capable of being extended to recognize new protocols as they come into general use.

The most common use of a uniform resource identifier these days is to locate a Web page using a URL. URLs are commonly thought of as pointers to a filename in a directory on a Web server. Most people think URLs look like this:

```
http://www.foo.com/~bar/file.html
```

To a large extent that is correct; however, URLs can also point to ftp sites, gopher space, newsgroups, and other resources using protocols other than HTTP. The preceding representation of a URL is called an absolute URL because it specifies the protocol to be used to make the connection, it states that the request for the resource complies with the Internet Protocol, and it states the path to be followed to locate and retrieve the resource. However, URLs can be *relative*, as shown in this example:

```
file2.html
```

In this case, the browser will look for `file2.html` in the same directory as the resource located at `~bar/file.html`.

The preceding section covers HTTP, but URIs can access files using other protocols as well. Generically, a URI consists of four parts: the protocol, the Internet node, the file path, and optional arguments. A URL looks something like the following:

```
[protocol]://[internet node]:[port][/file path/file path]?[optional arguments]
```

Protocols

The protocol is the mechanism by which access to the Internet node is obtained. More than a dozen protocols are available, and a more detailed examination of a few of the more common ones can be found later in this chapter, in the section "The Syntax of Schemes."

The Internet Node

The Internet node given for the connection is the fully qualified domain name (or IP number) of a computer with a permanent connection to the Internet.

The File Path

The file path is optional. According to RFC 1738, the authority on the subject, if the file path is omitted, the slash is optional. Therefore, if you want to obtain a file from `www.foo.com`, the full Request-URI is blank. You would direct your browser to the following URI:

```
http://www.foo.com
```

TIP

Much discussion is waged on certain Usenet newsgroups concerning the need for a "trailing slash." The world of the World Wide Web exists mainly on the UNIX operating system. In UNIX, the slash has two functions. In one function, it acts as a separator between directories and files. The top directory, which contains all other directories, is called the *root directory*, or just *root*. The second function of the slash is to represent the *root directory*. The path to a file in a subdirectory could be represented as `/root/foo/bar/ralph.txt`. This is an *absolute pathname*, providing a complete and unambiguous name for the file `ralph.txt`. The first slash refers to root, and all the other slashes separate the names of directories, except for the last slash, which separates the filename from its directory.

Although URIs follow the UNIX convention of using a slash as a separator, a URI is not a UNIX filename. Using a slash as a URI separator works on all operating systems, even DOS and Macintosh. The slash in a URI is considered a separator of components of a hierarchical name known as the *url-path*. The slash preceding the url-path is not a part of the url-path. It is the indicator of the termination of the `<host>:<port>` part of the `<scheme-specific-part>`, and indicates the beginning of the url-path. The use of slashes in the url-path does not mean that the URI is a UNIX filename.

Optional Arguments

The optional arguments must be preceded by a question mark. The optional arguments are often passed to a CGI script and contain, for example, an encoded string for which the CGI script will search.

Uniform Resource Locators—A Detailed Examination

For most hobbyists, this brief overview will be more than sufficient. However, CGI programmers and others need to resolve a number of issues in order to create Web pages on the fly, create searchable databases, or receive and return information from forms on a Web page.

The Parts of a URL

A generic URL can be represented as `<scheme>:<scheme-specific-part>`. The URL must contain the name of the scheme being used (which is what I called a *protocol* earlier in this section). The name of the scheme must consist of lowercase letters, numerals, the plus sign, the period, or the hyphen. Programs that receive scheme names in uppercase letters should treat them as lowercase.

Character Encoding

Characters other than those listed in the preceding paragraph must be encoded. As I mentioned earlier, an octet is an 8-bit byte; the characters in a URL represent the octets used in Internet protocols. The US-ASCII coded character set defines the relationship between characters and octets representing those characters. Each octet can be encoded in hexadecimal using the percent sign and two hexadecimal digits. (The hexadecimal digits are `0123456789ABCDEF`; lowercase letters are recognized.)

It is required that octets be encoded if they have no displayable character in the US-ASCII coded character set or if the character represented by the octet is "unsafe" or "reserved."

Characters that cannot be displayed range from hexadecimal `00` through `1F`, `7F`, and `80` through `FF`. Characters that are unsafe are those that have different uses in different situations. These characters include the space, <, >, the quotation mark, #, %, {, }, |, \, ^, ~, [,], and '. All unsafe characters must be encoded within a URL by representing them as two hexadecimal characters preceded by the percent sign. Characters are reserved when their characters have specified meanings. If the character is reserved in a scheme, its octet must be encoded. Reserved characters are ;, /, ?, :, @, =, and &.

The Syntax of Scheme-Specific Parts

Although the scheme comes first, it is inextricably bound to the *scheme-specific part*, which I will cover first. The scheme-specific part is presumed to use the Internet Protocol to access a specified host on the Internet to locate and retrieve specific data. The scheme-specific part must follow this syntax:

```
//<user>:<password>@<host>:<port>/<url-path>
```

The scheme-specific part must begin with a double slash, which indicates that it complies with the Internet scheme syntax. The following parts are optional and can be omitted: `<user>:<password>@`, `:<password>@`, `:<port>`, and `<url-path>`. The scheme-specific part must end with a carriage return/line feed not shown here.

The `<host>` portion must be a fully qualified domain name or its IP address (a dotted quad, which is four decimal digit groups separated by periods). `<port>` is the software port number to connect to. Most schemes have a default port specified; if no port number is given, the default is used. Omit the colon if omitting the port number. `<url-path>` is the location of the specified file to be retrieved.

CAUTION

The slash following the `<host>:<port>` is *not* part of `<url-path>`. See the discussion in the section "The File Path," earlier in this chapter. If only `<host>` or `<host>:<port>` is used, the slash can be omitted.

url-path is dependent on the scheme and is discussed with each scheme section.

The Syntax of Schemes

Schemes (or protocols) are registered with the Internet Assigned Numbers Authority. A new scheme can be submitted for registry with a definition of the algorithm for accessing a resource and the syntax for representing the scheme. Rather than provide a full description of all the schemes, I've limited the discussion to the most common schemes.

file

`file` is the method used to designate a file available on a local fixed disk; it cannot be used to identify a file accessible over the Internet. A typical `file` request looks like the following:

```
file://<hostname>/<path>
```

In this request, *<hostname>* is the fully qualified domain name of the system, and *<path>* is the hierarchical directory path residing on the host. *<hostname>* can be omitted, in which case *<path>* is resolved relative to the computer from which the URL is being resolved. `file` is unique in that it specifies neither an Internet protocol (such as `http`) nor an access method (such as `news`) for a resource.

File Transfer Protocol

The `ftp` scheme is used to retrieve files from ftp servers. A username and password can be supplied to be passed to the server after the connection is made. If they are not supplied and one is requested, the username `anonymous` should be sent in response to the server's request for `USER`, and the Internet e-mail address should be sent in response to the server's request for `PASS`. A typical request for anonymous ftp would look like this:

```
ftp://ftp.foo.com/bar/MyPrize.gzip
```

The *url-path* for `ftp` is `<cwd1>/<cwd2>/<name>;type=<typecode>`. *<cwdN>* represents the directories to be entered as arguments to the change working directory command. *<name>* is the file to be retrieved, and *<typecode>* is one of `a`, `i`, or `d`, representing the type of transfer (ASCII, binary, or list). If a slash must be used in a *<cwd>* or *<name>* component, it must be encoded. For example, if the path sought is `/pub/myfile.text`, located on the server known as `ftp.foo.com`, the properly encoded URL is

```
ftp://myname:mypword@ftp.foo.com/%2Fpub/myfile.txt;type=a
```

This URL establishes a connection with the server, transmits the name `myname` and the password `mypword` in response to the requests, changes the directory to `/pub`, and retrieves `myfile.txt` in ASCII mode. If a username is supplied without a password, the user agent should prompt the user for a password. Unlike a true ftp connection, the connection will be terminated following the file transfer.

Gopher

Gopher is a protocol that is used to provide indexing of information on the Internet, and it is text based. A gopher URL looks like this:

```
gopher://<host>:<port>/<gopher-path>
```

The default port is `70`. *<gopher-path>* is one of the following:

```
<gophertype><selector>
<gophertype><selector>%09<search>
<gophertype><selector>%09<search>%09<gopher+_string>
```

The entire *<gopher-path>* may be empty; in that event, the slash is optional, and *<gophertype>* defaults to `1`. *<selector>* is the gopher selector string. For specific details on gopher types and selector strings, see RFC 1436.

Hypertext Transfer Protocol

HTTP is discussed in the first part of this chapter. The syntax of an HTTP URL is

```
http://<host>:<port>/<path>?<searchpart>
```

The default port is `80`. No username or password is allowed. *<path>* and *?<searchpart>* are optional; if they are omitted, the slash can also be omitted. (See the discussion in the section, "The File Path," earlier in this chapter.) Within *<path>* and *<searchpart>*, the slash, semicolon, and question mark are reserved. The slash can be used to act as a separator in a hierarchical structure.

mailto

The `mailto` scheme is used to designate an Internet e-mail address. The form is

```
mailto:<rfc822-address-specification>
```

There are no reserved characters, but RFC 822 addresses often contain a percent sign that must be encoded. Notice that the URL does not contain the double slash indicating compliance with the Internet Protocol. The `mailto` scheme does not request the retrieval of a file. The URL cannot contain any information other than an Internet e-mail address.

news

The `news` scheme is used to access either a newsgroup or an article of news in Usenet form. The URL form required depends on whether a newsgroup or an article is to be accessed:

```
news:<newsgroup-name>
news:<message-id>
```

<newsgroup-name> is a period-delimited name such as *comp.infosystems.www.authoring.html*. *<message-id>* refers to the Message-ID header of a posted article without the enclosing angle brackets, along with the fully specified domain name. Notice that the URL does not contain the double slash indicating compliance with the Internet Protocol. The URL will refer to the newsgroup on the host of the user.

telnet

The `telnet` scheme is used to institute a `telnet` session through the browser. The URL is

`telnet://<user>:<password>@<host>:<port>/`

The default port is 23. The `<user>:<password>`, `:<password>`, and slash portions are optional. The `telnet` scheme does not specify a path to a file. Remote logins vary in implementation and allowed activity.

A Few Problems with URLs, and Some Proposals

Web browsers encounter problems with URLs. They reach dead links and have transfer times that stretch into infinity on overseas links and overloaded sites. Let's consider a frequent situation: A person has moved from one Internet service provider to a new one, and the original account has been terminated. The resource that person maintained on the first ISP has been moved, but there is no method of determining the new location until search engines have been updated. (This is sometimes referred to as a "lack of persistence." The URL can disappear without a trace just because it was relocated.) Uniform resource names have been proposed to solve this common problem. URNs would be permanently affixed to an object and resolved to the URL wherever the resource may happen to be. URNs can also be used to locate a copy of the resource nearest the requesting user agent, in the event of multiple copies. The goal is to provide persistence, cutting down on wasted requests for resources that have been relocated.

Another proposal is the uniform resource characteristic (or citation), where a resource would not only be described, but would also provide the author, title, and additional locations if available. A URC can also indicate data type, which is helpful in determining how to download a resource and how to handle it after it is obtained; or it can indicate copyright status, which is helpful in alerting users that the resource is not to be shared promiscuously (or alerting users that it can be so shared). A URC, when implemented, is likely to be a series of `<fields>=<values>` similar to the META element in HTML.

Knowledge Representation

People with Web browsers can obtain information from databases located anywhere in the world. Overnight couriers have placed their status reports online, where anyone with the appropriate transit number of an overnight shipment can access the couriers' databases and determine the status of the delivery. Online services allow the order of automobiles without having to drive to a dealership. Bookstores are online. Large corporations represented by several law firms in different parts of the world now require the firms to make contract language, research, and legal memoranda available for other lawyers, reducing duplicate effort and duplicate billing on as many issues as possible. The amount and timeliness of business information available on the World Wide Web boggles the imagination, and much of the information is provided at no charge in the hopes that the user will buy other more timely or more proprietary information.

The use of HTTP and URIs on the World Wide Web enables an unbounded nonlinear information system. The protocols used by URIs provide us with access to resources that might be located anywhere in the world. By using standardized syntax in accordance with the registry maintained by the IANA, URIs are infinitely extensible. As new resources are invented, the method of locating and retrieving them can be specified and registered for worldwide use by all. By using the public protocols of TCP/IP and publishing the protocols of HTTP and URI, the W3C has attempted to ensure that the Web will remain accessible to all with no requirements of proprietary hardware or software.

In 1945, Vannevar Bush, President Roosevelt's Director of the Office of Scientific Research and Development, wrote an article in *The Atlantic Monthly* entitled "As We May Think." He proposed a mechanical contraption built into a desk, with information stored on microfilm and several projectors and screens available in the desk for viewing more than one resource at a time in a random fashion. He called this contraption a *memex*, which was short for *memory expander*. As Vannevar Bush imagined in 1945, we can access information on the Web in a random fashion, without having to wade through a serial retrieval process. We can add to the data by publishing our own Web pages, and have our information become a part of the rhythm of the Web. Unlike Bush's *memex*, however, our nonlinear information system is not limited in its resources to the material on our desktop.

As Ted Nelson foretold, we have available to us *hyperspace*. With links not only to text but to images, maps, sound, and whatever else can be imagined and implemented, our resources have become unbounded. Unlike early attempts at nonlinear information systems, with the World Wide Web we are not required to buy specific hardware or software for access to this new world of data, and we are not limited to resources that must be purchased and reside within our computers.

The steps leading to our present riches were certainly not inexorable. Vannevar Bush recognized that the explosion of new knowledge resulting from World War II would be impossible to keep track of. Bush's article suggesting a solution inspired Doug Engelbart, who happened to see the magazine half a world away in the Philippines and remember it until the 1960s. Engelbart's work on the mouse and collaborative authoring made it possible even to consider a computer as a support tool for just one person, something unheard of in the days of mainframes. Engelbart was awarded the Lemelson-MIT prize in 1997 in recognition of the advances based on his fundamental concepts. Ted Nelson and his Xanadu became a rich source of inspiration for supporters of nonlinear information, but it took the United States Department of Defense and its determination to survive—if not win—global thermonuclear war and all the destruction wrought by atoms to create a survivable network of networks on which such a system could run. And finally, it took high employee turnover at CERN to bring us to the beginning chapter of the World Wide Web: In the face of high employee turnover, how could CERN preserve the knowledge wrung at great cost from the destruction of atoms?

Today, we move information in myriad ways over the Internet and the World Wide Web. Our access is independent of any platform or software requirements, thanks to the public standards implemented by Tim Berners-Lee. We download text, images, video and sound clips, and executable programs that will run on any computer. None of this was possible with proprietary programs or proprietary hardware, and it is unlikely it would be possible today had the standards for doing so been proprietary. As you work through the chapters of this book, learning to create pages with HTML, to create scripts and programs for interactive pages, and to suggest design and layout, remember that accessibility for all users is fundamental to the existence of the Internet and is the foundation on which the World Wide Web is built.

Summary

Hypertext Transfer Protocol and uniform resource identifiers work hand in hand to enable the transfer of data over the Internet through the World Wide Web. HTTP interfaces with TCP/IP to establish connections between computers and send packets of data back and forth, bringing us text, animated images, sound, and programs that run on virtual machines in our computers' memory. Uniform resource identifiers enable our computers to locate these resources for retrieval. Because these standards are publicly available, there are no restrictions on who can use them, and there are no restrictions on their use. Indeed, it is expected that they will be extended and improved as new ideas for data formats evolve.

Beginners in any subject always ask questions that begin with *how*. After they have learned how, they ask the interesting questions that begin with *why*. In this chapter, you have learned how HTTP and URIs work. As you read through the remaining chapters of this book and learn how to write CGI programs and style sheets, for example, you will already have the background to understand why CGI scripts must be structured to return certain information in response to a query, and why style sheets include a TYPE declaration. By having some of the why questions already answered, it will be easier to apply the lessons in learning how.

CHAPTER 6

Web Browsers and Platforms

by Mike Sessums

IN THIS CHAPTER

The user's platform and browser can affect how your page is viewed. It also determines what can be accessed on your Web site.

This chapter introduces you to HTML standards and browser extensions. It also helps you become familiar with the various platforms and browsers people are using to access the Web. Armed with this information, Web site developers can better plan their sites.

Extensions Versus HTML Standards

Some people believe that it is all Microsoft's idea, but the W3 Consortium (W3C) sets the HTML standards for the World Wide Web (W3). Any browser that wants to survive must conform to the standards set by the W3C, not the other way around.

The W3C (`www.w3.org`) is composed of members from universities and leading-edge technology corporations around the world. Even with Microsoft and Netscape as members, the W3C tries to stay platform- and browser-neutral. This helps avoid a conflict of interest.

The consortium was created in 1994 in conjunction with CERN, the European Laboratory for Particle Physics in Switzerland. CERN (`www.cern.ch`) was the group that gave birth to the Web in 1989. Some of the W3C's goals include helping others realize the Web's potential by developing new standards, and making software to test those standards publicly available. (See Chapter 2, "The History of HTML," for more information on the roles that the W3C, Microsoft, and Netscape play in HTML development.)

Extensions are very hard to differentiate from standards. An example of an HTML standard is `<HR>`, meaning horizontal rule. A Netscape extension to this standard is `SIZE=`*n*. The latter allows you to specify how thick you want to make the horizontal rule.

Take a look at Listing 6.1 to see how these two tags are used.

Listing 6.1. Horizontal rules: extensions vs. standards.

```
<HTML>
<BODY bgcolor="ffffff">
<BR>
<HR>
<BR>
<HR SIZE=8>
</BODY>
</HTML>
```

The fourth line in Listing 6.1 shows the HTML standard, and the sixth line shows the Netscape extension. The `<BR>` gives your sample extra padding by using line breaks. The `bgcolor="ffffff"` attribute sets the background color to white, so that you can see the shaded horizontal rules better.

Figure 6.1 shows the difference that the extension makes when viewed with Netscape Navigator 4.0.

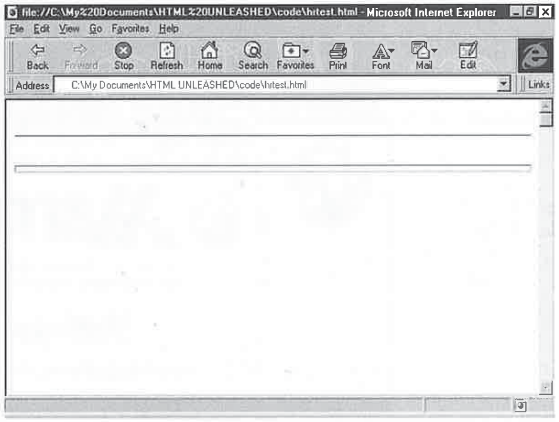


FIGURE 6.1.
The results of HTML standard code for horizontal rule (top line) compared to the Netscape `SIZE` *extension to that code (bottom line).*

As you can see, browser extensions are coding additions that the browser manufacturers would like to see added to the current HTML standards. That is why some former browser extensions are now part of the HTML standards. Extensions are also a good way for one browser to try to outshine the others. This can be a real boon for Web site developers.

HTML and Platform Idiosyncrasies

Not all platforms and their software support the latest HTML standards and browser extensions. Some plug-ins are available only on certain platforms.

The major platforms are either Windows-, Mac-, or UNIX-based machines. However, many other players are involved in the hardware and browser war, and with each come their own bonuses and drawbacks for Web site development.

Pixel size is a very confusing measurement when compared to physical display size, and it can be very deceptive. A 640×480 image fills the screen when display devices of different physical sizes are set to 640×480 pixel resolution. As an example of this, you could view the same page on a 14-inch monitor and a 20-inch monitor set at 640×480 pixels. They both fill the screen, but one physically displays six inches larger than the other (as shown in Figure 6.2).

A platform pixel comparison chart is located in Table 6.1.

FIGURE 6.2.
Comparison of physical size to pixel size.

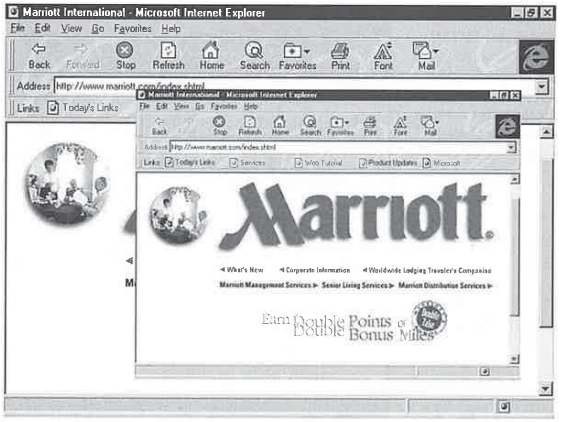


Table 6.1. Platform and display size comparison in pixels.

Platform	*Display*
PC	640×480/1024×768
NC	1280×1024
Notebook	640×480
WebTV	560×420
Palmtop	480×240

One thing to note on this chart is that even though the WebTV has a 560×420 display, the actual Web page design/display area is 544×378 with no horizontal scrolling.

Something else to consider when designing your site is what type of connection your target audience is using. If it is a T1 line typical of a corporate environment, you might be able to easily add plenty of graphics, background music, and VRML environments without worrying about load time. A link to a 10MB video is not a major problem and will probably load in just a couple of minutes.

However, if your target audience is accessing your site over a telephone line, you should think twice about overloading that site with frills. Even with a high-speed modem, a Web surfer will probably move on to another site long before your page gets a chance to load.

TIP

Although 300 dpi or higher resolution images might look great on paper, most computers can display only 90 dpi or less. Images much larger than this on your Web page are simply a waste of bandwidth. Saving your images at 90 dpi or 100 dpi will decrease load time and still deliver a quality image to your user's screen.

Now, let's take a closer look at the platforms people are using to access the Web, including personal computers (PCs), network computers (NCs), notebooks, laptops, Personal Digital Assistants (PDAs), and handheld PCs (HPCs). Each of these has its own operating system (OS), display, connection, and other features that can make Web site planning a challenge.

Personal Computers

The biggest user base for most Web developers can be found in the *personal computer* (*PC*), which is popular in both corporate and home markets. The PC market is dominated by IBM clones and followed by PowerMacs, and some of the most innovative developments in Web software and support can be found here.

PCs in the U.S. have sported a variety of operating systems, including DOS, Windows, and OS/2 for the IBM and clones, and various System X incarnations for the Apple. In Japan, there were at least seven popular PCs and operating systems at last count.

Other PCs such as Atari, Amiga, and Commodore 64 and 128 are also being used to access the Internet, but in considerably fewer instances. Although their user base is not as large as the PC user base, computers such as the Commodores are still quite popular in several countries, mostly because of their price.

Luckily for Web developers, Microsoft has been able to use its Windows environment (in a variety of flavors, including 3.1, 95, and NT) to narrow the OS choices on IBM PCs and clones for a large portion of the world. Intense marketing by Microsoft and the waning popularity of the Apple PCs have also given Microsoft a strong hold on the PC market. Of course, this makes Web site planning and development easier, because it moves everyone closer to a standard.

However, Apple might not be down for the count just yet. Apple's Quicktime and QuicktimeVR formats have left their mark on many Web sites. The company's plans to team BeOS with its PowerMacs is one strategy it is using to get established in the consumer PC market again.

PC monitors are typically in the 14- to 20-inch range at 640×480 to 1024×768 pixels. The normal color range can be from 256 to 16 million colors. However, you might want to stay away from a 16-million color splash image because of the image file's considerable size.

PC users in the private sector usually connect to the Internet over residential phone lines. A much smaller number of users have high-speed ISDN connections. Corporate PC users, on the other hand, are usually connected to an intranet, just like their network computer cousins.

Network Computers

Besides the PC, Web developers in a university or corporate engineering environment may also be catering to network computer (NC) users. The NCs can't match the popularity of the PCs for the general public. One reason is price, and the other is the use of a different marketing strategy than the PCs.

UNIX is the biggest OS player in this field, with Sun and Silicon Graphics leading the pack. Hewlett-Packard also produces UNIX workstations, but it has moved more toward the network PC in its partnership with Microsoft and Windows NT.

At one time, Digital's VAX VMS was the king of the NCs, but Web software development is almost nonexistent now. Many VAX users are still strapped with an archaic version of the Mosaic browser that doesn't support tables, forms, frames, Java, ActiveX, or the new HTML 3.2 standards. Web developers will find their options very limited when working with this target group.

NC monitors are typically in the 17- to 20-inch range at 1280×1024 pixels in both color and grayscale.

Most NC users are permanently connected to their school's or company's network (hence the name) or intranet.

Laptops and Notebooks

Some of the new laptop and notebook computers sport awesome features that take full advantage of your Web site. Their performance typically rivals that of Pentium-based desktop computers.

One leading notebook is a true multimedia computer with a 12-inch SVGA screen, 16-bit sound system (with bass and treble controls), full-screen MPEG capability, and 128-bit graphics.

Color text and graphics are great on some of the SVGA screens but are difficult to see or totally washed out on others. Animation and other moving objects on Web pages have a tendency to blur or leave ghost image trails, especially when light-colored objects are used. Another drawback is that many users with 16-bit sound have only the tiny built-in speaker, which could make your site's sounds come out rather tinny.

The most common OS you will find in the notebook computer market is Windows. Notebook users can use the same browser software, browser plug-ins, and Web applications used by their desk-bound counterparts.

Notebook users generally connect to the Web through cellular phone hookups or anywhere they can find a place to "jack in."

Micro Minis (Palmtops and PDAs)

Many of the Palmtops and PDAs (Personal Digital Assistants) that have cropped up over the years have been nothing more than glorified (and expensive) address books. However, newer models that have hit the shelves have been tagged with the acronym *HPC* (Handheld Personal Computer).

Although these little computers can't compare with the power of the standard desktop PC, their portability and compatibility with their bigger brothers make them an alternative to the notebook computer.

When PDA users were surveyed, one of the top items on almost everyone's wish list was a Web browser. Several of the PDAs do support some form of access to the Internet, thanks to prepackaged, third-party, and home-grown software. Some of the manufacturers even include a programming language so that users can create their own software.

Currently, each HPC has its own proprietary operating system, but Microsoft might bring everyone back to standardization again. Based on a Windows 95 look and feel, Windows CE offers "pocket" versions of popular software such as Word, Excel, and Internet Explorer with Windows 95 file-sharing compatibility. A new version of Windows CE reportedly will support Java and DVD (Digital Video Disk). Many hardware and software developers have already jumped on the Windows CE bandwagon and pledged their support. Casio, alone, plans to ship 500,000 of its Windows CE–based Cassiopeias to consumers by next year.

These palmtop marvels sport only 2MB or 4MB of RAM and 4MB or 8MB of ROM ("the virtual hard drive"), making most graphic-laden Web sites a real burden, if not a downright impossibility to view. Rectangular 480×240 pixel backlit grayscale displays are typical of the HPCs.

Palmtop users make their connection to the Internet in a variety of ways, due to the portability of these little workhorses. They can be connected using cellular phones, wireless modems, and standard telephone lines. One company even offers a Windows CE-to-NT network connection. But here's the rub: Modems available for some of these units run as slow as 2400 baud, which makes text browsing the only option. Luckily, many users have access to 19.2Kbps or faster modems.

Browser Heritage

Before the graphical environment of the Web came along, users happily surfed through text on the Internet with computers that emulated standard terminals such as the VT100.

This greatly limited the information that could be put on the Net. Content developers were restricted to a simple hyperlinked format for their documents and information. There were no sounds, animations, or virtual environments.

With CERN's development of the World Wide Web, all of that began to change. The new graphic environment allowed images to be added to information.

Lynx brought a little more style to text browsing, allowing users to access hypertext documents and the new formatting. Graphic browsers such as W3C's Arena, Cello, and the groundbreaking

Mosaic from NCSA really opened up a world of possibilities for a new breed of Net content authors.

Mosaic became the most popular Web browser, due to its free distribution, support, and availability on a wide variety of platforms. The Netscape Navigator grew out of this heritage and became the king of browsers.

The Netscape Navigator grew in circulation and support, bringing new innovations for how Web content could be shared. Microsoft had underestimated the importance and scope of the improvements in Web development and usage, so Netscape was able to get a firm hold on the market before Microsoft realized what had happened.

Finally realizing the enormity of its error, Microsoft had to make a mad dash to get a foothold in this rapidly growing market. Thus began the "browser wars."

Many graphic browsers are in circulation today, but none appear to be a serious challenge to either the Netscape Navigator or the Microsoft Internet Explorer.

Two Top Contenders: Netscape Navigator and Microsoft Internet Explorer

Netscape and Microsoft are in a real power struggle to win the Web browser market. The good thing about their browser wars is that developers have seen some great innovations and advances in Web browsers and plug-ins as each company tries to outdo the other. To the victor belongs the Web.

Of course, there is a down side, too. Do you design your page to satisfy the Netscape Navigator users, or do you set your style around the Microsoft Internet Explorer? It is almost impossible to take full advantage of both on the same Web site, although some people try.

You can optimize your page using Netscape Navigator extensions, or you can make it "best viewed on Internet Explorer." Microsoft even offers special incentive programs to designers who put its Internet Explorer logo on their Web sites.

Even when a browser conforms to the W3C HTML standards, it might view the same page differently.

Figure 6.3 shows how Netscape Navigator 4.0 views a page that has a creative use of frames. The page looks nice, clean, and crisp. This really has all the makings of a perfect Web site.

Look at the same page with the Microsoft Internet Explorer (as shown in Figure 6.4), and you can see a world of difference.

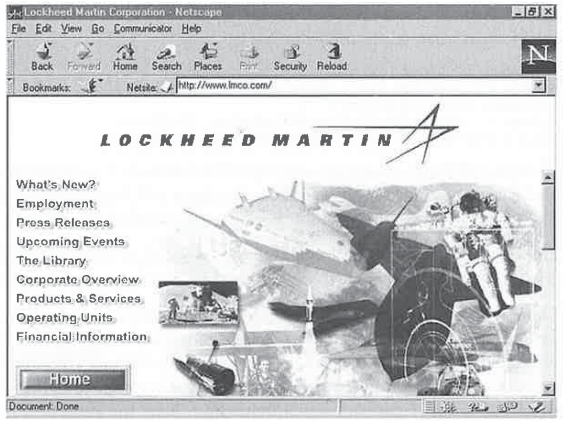


FIGURE 6.3.
The Netscape Navigator shows a clean page with no horizontal scrolling necessary.

FIGURE 6.4.
The same page shown in Figure 6.3 as viewed with the Microsoft Internet Explorer 3.02.

The obvious difference in how the Internet Explorer 3.02 displays the page is the need for horizontal scrolling to see the whole picture in the frame on the right. Notice also how each frame element is clearly defined by hard lines that were not visible in the Navigator rendition of the page. Figure 6.3 looks like a professional publication, whereas Figure 6.4 looks like someone's TV dinner tray.

So why does the output from the same page differ so much? The answers can be found in how the browsers interpret the following code:

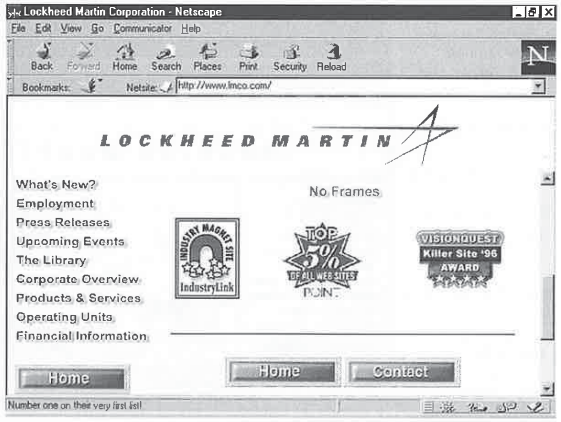
```
<frameset border=0 rows=90,*>
<frame src="lmclogo.html" scrolling="no" noresize>
<frameset border=0 cols=170,*>
<frame src="mainmenu.html" scrolling="no" noresize>
<frame src="composite.html" name="main" noresize>
```

The first and third lines of this excerpt of HTML code turn the border off with `border=0`. This works just fine in Navigator (as you saw in Figure 6.3), but it is ignored in Internet Explorer 3.02, where it shows up as a visible border (as you saw in Figure 6.4).

Lines two, four, and five are where scrolling is turned off with `scrolling="no"` and frame resizing is turned off using `noresize`. These last two bits of code are recognized by both browsers.

One negative thing you should notice about both pages in Figures 6.3 and 6.4 is that the information in the left frame spills off the screen. This is normal for images larger than the computer's display size. However, what makes it a bad design element is that the designer neglected to provide a scrollbar to let users with lower display settings access all the information he worked so hard to create. Figure 6.5 demonstrates how useful the scrollbar is in accessing important information in the frame on the right.

FIGURE 6.5.
Using the scrollbar reveals navigation buttons and other links.



The addition of a scrollbar for the left frame may have been avoided for the sake of graphic design, but at the cost of functionality. If the user was able to change his display settings to at least 800×600, he could see the hidden Contact button on the fixed left frame (shown in Figure 6.6).

FIGURE 6.6.
A Contact button appears when viewed at a higher resolution.

A good solution to this problem would have been to make the elements in the left frame a bit tighter. Of course, the majority of users at Lockheed Martin, who have a mixture of PCs and NCs, are probably not viewing these sites at 640×480. However, the target audience for this Extranet page probably includes the general public and potential customers.

TIP

Be sure all of your page elements—especially those used for navigation—can be viewed on a screen set at 640×480. Add a scrollbar if needed, or make your page tight enough that viewers can see all of the elements without scrolling.

Scrollbars automatically appear on most browsers when the data or image areas are larger than the browser's viewing window. However, scrollbars can be turned off in the code as you have already seen.

Now look at that site with Microsoft Internet Explorer 4.0 (shown in Figure 6.7).

In Figure 6.7, you can see that Microsoft has cleaned up its act by recognizing the `border=0` tag. Also, notice that you can see the Contact button in the left frame without resetting the display to 800×600. Hey, this browser now has potential!

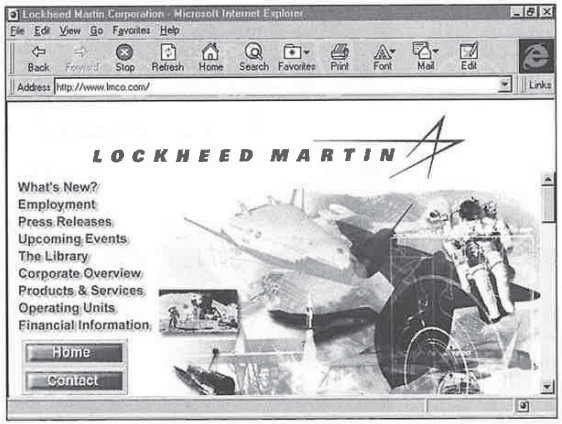


FIGURE 6.7.
Look Ma, no frames! Internet Explorer 4.0 fixes some of the problems found in Internet Explorer 3.02.

Netscape Navigator

The Netscape Navigator was the first graphic browser to really take the Web by storm. Netscape Navigator 4.0 carries on the tradition as part of an integrated Web communications suite called Communicator 4.0.

Besides Navigator 4.0, this suite consists of Messenger, Collabra, Conference, and Composer. Messenger is the mail client, and Collabra Message Center is an integrated newsgroup reader and e-mail package.

Conference allows users to engage in voice chat and video teleconferencing. An interactive white board allows users to exchange ideas. Users can even exchange files easily while keyboard or voice chatting. One of the most exciting features of Conference is *collaborative browsing*. This feature allows one user to take others with whom he is chatting for a surf to his favorite sites on the Web.

Web sites could use some of these features to provide live, interactive customer feedback and technical support. Suppose that a user came to your site to locate a video driver for a card he just purchased from your site. He's totally lost, but a navigation button takes him to Netscape Conference and he's hooked up with technical support. While support and the customer are talking, support could engage collaborative browsing and take him directly to the right download page.

Netscape Composer is a full-featured WYSIWYG development tool with a toolbar that is typical of Web editing packages. (See Figure 6.8.)

FIGURE 6.8.
Netscape Composer has an easy-to-use, intuitive interface.

Well-defined buttons on the toolbar provide easy access to elements such as links, targets, images, horizontal rules, tables, and lists.

Of course, just about every plug-in imaginable is supported here, as are the W3C's Cascading Style Sheets (CSS).

Netscape Navigator has some of the most popular extensions to the HTML standards. These extensions give you ways to liven up unordered lists with various bullet shapes, have more control over image maps, and create special animations through a technique called *server push*. Of course, this only touches on a few of the capabilities that are possible. (See Table 6.2 for more Netscape Navigator extensions.)

Table 6.2. Netscape Navigator extensions.

Extension	*Description*
`<area coords >`	Area coordinates for image map
`<area nohref>`	Defined area has no action
`<area shape>`	Area shape (such as circle)
`<basefont size>`	Specifies base font size
`<big>`	One size larger than base font
`<blink>`	Blinking text
`<body alink >`	Activated link color

continues

Table 6.2. continued

Extension	*Description*
`<body bgcolor >`	Background color
`<body link>`	Hypertext link color
`<body text>`	Normal text color
`<body vlink>`	Visited link color
`<caption align>`	Aligns table or figure caption
`<center>`	Center
`<div>`	Divisions
`<embed>`	Embed content
`<font size=value>`	Font size
`<hr align>`	Horizontal rule alignment
`<hr noshade>`	Horizontal rule shading off
`<hr size>`	Horizontal rule thickness
`<hr width>`	Horizontal rule width
`<img border>`	Image border size
`<img hspace>`	Image horizontal spacing
`<img usemap>`	Image map
`<img vspace>`	Image vertical spacing
`<isindex prompt>`	Text of new prompt to use
`<li type>`	Type of bullets to be used
`<map name>`	Image map name
`<nobr>`	No break
`<ol start>`	Starting ordered list number
`<ol type>`	Type of bullets for ordered list
`<server push>`	Animation by *pushing* data to browser
`<small>`	One size smaller than current font
`<sub>`	Subscript
`<sup>`	Superscript
`<ul type>`	Type of bullets for unordered list
`<wbr>`	Word break

Microsoft Internet Explorer

The word *free* is one of the big reasons that the Microsoft Internet Explorer has gained popularity so quickly. After all, that's how Netscape started out. Now that Netscape is charging for Navigator, and Microsoft has a feature-packed browser, more and more people are turning toward the Internet Explorer.

Internet Explorer CD-ROMs are getting almost as much circulation as AOL disks. Many CD-ROM magazines are including a gratis copy of Internet Explorer along with their other software on disk. Other computer magazines are getting into the act by providing browser CD-ROMs from Microsoft.

Let's face it. For most of the general public, it is hard to compete with free. That's why Netscape has found it impossible to dismiss the Internet Explorer. Of course, Microsoft has used this free browser to test some features and plug-ins (which Microsoft calls add-ons and ActiveX controls) that have made Netscape work even harder to keep its market share. According to industry leaders at a recent Web authoring convention, Netscape has fallen to about 60 percent over the past year, leaving Internet Explorer with a well-earned 30 percent. A scattering of other browsers make up the rest.

Microsoft's Internet Explorer 4.0 comes as part of an integrated communications package, much like in Netscape's Communicator 4.0. This new package contains e-mail, broadcasting, and a system shell that tries to fully integrate the Web with the user's desktop. This is very much in keeping with Microsoft Office 97 in making the Internet a seamless extension of Windows. In this new scheme of things, the Internet is accessible through hypertext links in applications such as Word, Excel, and PowerPoint. The Internet Explorer shell even adds the Internet as a "drive" that you can access through the Windows Explorer file manager (as shown in Figure 6.9).

The Microsoft Internet Explorer makes expanded use of Microsoft ActiveX, Active Video, Visual Basic, and Java support for added flexibility. It also can use many of the Netscape plug-ins and has plenty of its own add-ons, as well. These features make Internet Explorer 4.0 a Web designer's paradise.

All of these features and the desktop interface will definitely make the Microsoft Internet Explorer 4.0 tough competition to beat. After all, if you already have an integrated Web browser, why would you want to buy another one?

FrontPad, the Web authoring tool that comes with Explorer 4.0, is greatly underpowered compared to the many WYSIWYG authoring packages currently on the market. But, if they made it too powerful, no one would go out to buy Microsoft FrontPage or Publisher, would they?

Developers who have used Microsoft's Internet Assistants will find themselves on familiar ground. The current page being browsed comes up in FrontPad when the Edit button on the Explorer 4.0 toolbar is pressed. This gives you a semi-WYSIWYG environment that shows unknown HTML and other code as tags rather than integrating them seamlessly into the document while in edit mode. (See Figure 6.10.)

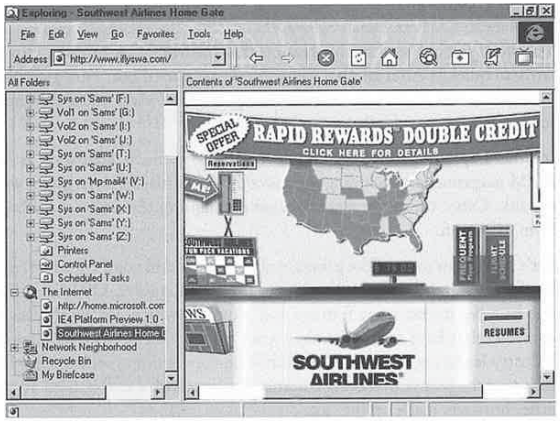


FIGURE 6.9.
Connecting to the Net through the Windows Explorer file manager. This is just one of the slick features of Internet Explorer 4.0 shell.

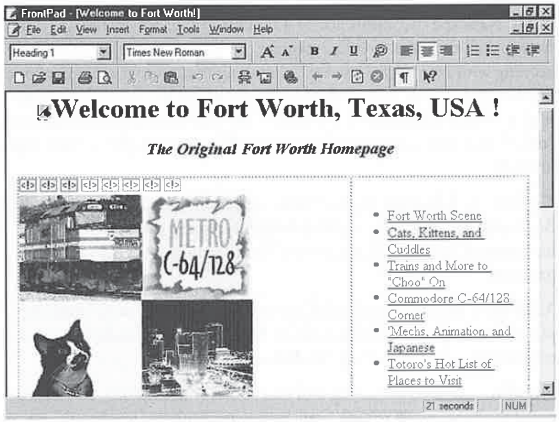


FIGURE 6.10.
Microsoft Internet Explorer 4.0's FrontPage editor. Well, it's almost WYSIWYG.

However, valid tags work just fine when you switch to browse mode (as shown in Figure 6.11).

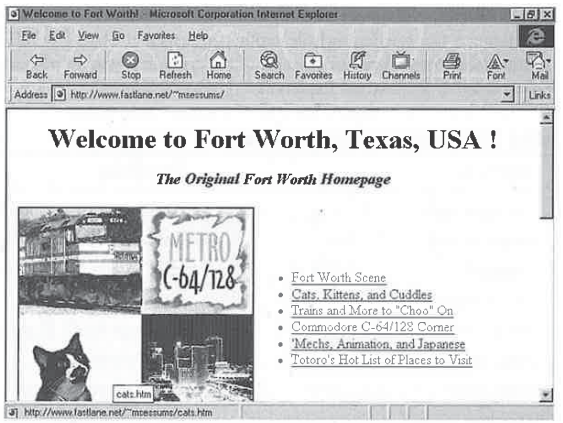


FIGURE 6.11.
Everything is sorted out when the code shown in Figure 6.10 is viewed with Microsoft Internet Explorer.

TIP

FrontPad is included in only the *enhanced* or *full* installation of Internet Explorer 4.0. If you've chosen one of these installations and FrontPad isn't configured as your editor, you can easily set it up yourself. Follow these steps from within Microsoft Internet Explorer:

1. Select the View pull-down menu.
2. Select Options from the list.
3. Choose the Programs tab.
4. Click the File Types button.
5. Choose Microsoft HTML Document 4.0 from the list.
6. Click the Edit button.
7. Choose edit from the list.
8. Enter the path of the FrontPad editor in the space labeled *Application used to perform action.*

One of the best things about the FrontPad editing interface is that you can directly publish your Internet site without saving it to your local disk. Be aware, though, that if you make a mistake in your edits, the mistake is already visible for the world to see when you save it. Be sure to always test your updated pages before releasing them onto the Web.

Microsoft Internet Explorer has only a handful of extensions compared to Netscape Navigator. Marquees (those irritating little scrolling messages) and background sounds that sometimes seem to go on forever are just a couple of the extensions you'll find here. (See Table 6.3 for a look at some of the Internet Explorer's HTML extensions.)

Table 6.3. Microsoft Internet Explorer extensions.

Extension	*Description*
`<bgsound loop>`	Background sound loop
`<bgsound src>`	Background sound URL
`<comment>`	Hidden text for comments
`<font color>`	Font color
`<font face>`	Font type face
`<img dynsrc>`	Inline video
`<img loop>`	Loop animated image
`<marquee align>`	Marquee alignment
`<marquee behavior>`	Marquee scrolling behavior
`<marquee bgcolor>`	Marquee background color
`<marquee direction>`	Direction of marquee movement
`<marquee height>`	Marquee area height
`<marquee hspace>`	Marquee horizontal spacing
`<marquee loop>`	Repeat marquee text message
`<marquee scrollamount>`	Marquee scroll speed
`<marquee scrolldelay>`	Marquee text blink
`<marquee vspace>`	Marquee vertical spacing
`<marquee width>`	Marquee area width
`<plaintext>`	Plain text
`<s>`	Strikethrough text
`<strike>`	Strikethrough text
`<table background>`	Table background
`<td background>`	Table data background

Various versions of Internet Explorer are available for Windows (3.1, 95, NT), Mac, and UNIX.

Other Browsers

You would think that, with the popularity of Netscape Navigator and Microsoft Internet Explorer, nobody would use anything else. However, a few other free and shareware browsers deserve to be mentioned. Of course, this is not a definitive list of browsers, but it covers some of the more popular ones.

Amaya

This browser is the W3C's test bed for the HTML 3.2 standards and features. It replaces Arena, which is now being developed by a separate group. These users will probably not be the greatest target audience for your Web pages, because Amaya is available only on the UNIX-based platforms and doesn't have the support found in other browsers.

AOL for Windows 3.0

There are more than 8 million potential users for your Web page at AOL. At one time, the AOL browser was pretty bad compared to more popular browsers such as Microsoft Internet Explorer and Netscape Navigator. However, AOL has teamed up with Microsoft to provide the AOL for Windows 3.0 browser. Compare the AOL browser in Figure 6.12 to the Microsoft Internet Explorer 3.02 in Figure 6.4.

FIGURE 6.12.
AOL's browser looks suspiciously like Internet Explorer with a few extra buttons.

AOL has also opened up its membership to people who already have Internet access. This allows them to use the browser of their choice, rather than AOL's proprietary browser.

NOTE

On some of the commercial services such as AOL and CompuServe, users sometimes connect at 2400 baud, so images can take a painfully long time to load.

Offer a low-bandwidth alternative to your site, such as text-only or lower-resolution graphics pages.

No special Web site development should be necessary for this browser.

Lynx

Here is another good reason to provide a text-only alternative to your site. Some people have only text-based access to the Web using text browsers such as Lynx. Although Lynx doesn't display graphics, it does recognize many of the HTML standard conventions. Other users may be connecting through a shell account with their terminals set to VT100 emulation. Web developers need to provide either text-only or text-alternate sites to support Lynx users.

Lynx is freely available for a variety of platforms including Amiga (Alynx), Atari, DOS (DosLynx), OS/2, PowerPC (Lynx), UNIX, Windows (Lynx 2.5), and VMS.

Mosaic

The final version of Mosaic has been released, and there is no longer any official technical support from the NCSA Software Development Group.

Mosaic 3.0 is packed with features. It has support for animated GIFs, tables, frames, and client-side image maps—almost everything that Web developers and browser users demand. What you won't find is support for HTML standards higher than HTML 3.2, because no further versions have been planned past Mosaic 3.0, which was completed in late 1996.

This freeware is available for Windows, Macintosh, and UNIX. Some earlier versions are also available for the VAX VMS platform, but it is extremely crippled by today's browser standards.

Amosaic, a popular browser for the Amiga platform and based on NCSA's Mosaic, continues to receive support and development. This shareware package can be found at any Aminet site on the Internet.

Microsoft Pocket Internet Explorer

Microsoft developed the Pocket Internet Explorer so that Windows CE-based palmtop users could surf the Web. (See Figure 6.13.)

The Pocket Internet Explorer does not support ActiveX, Java, JavaScript, plug-ins, VBScript, frames, or colored table cells.

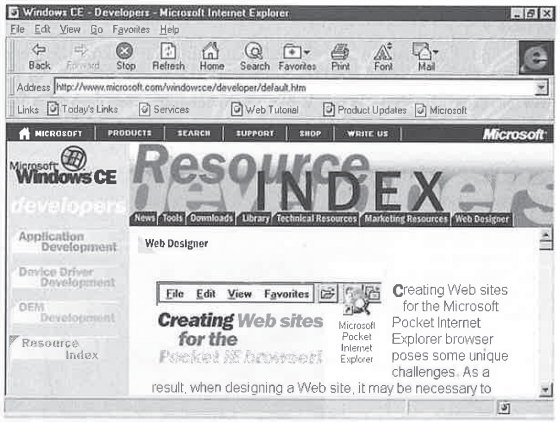


FIGURE 6.13. *Microsoft Pocket Internet Explorer developer's site.*

What is supported? Animated GIFs, background images, and the HTML tags that were supported by the full-size Internet Explorer 1.5. Microsoft does have plans for Windows CE to handle Java, so that could be a possibility in the future.

The Pocket Internet Explorer automatically resizes graphics to fit its 430-pixel display. Any text wider than this automatically wraps. However, Web designers can help speed up site access time by providing an alternate page in either text-only or reduced graphics. The Pocket Internet Explorer supports only 2-bit grayscale graphics. Microsoft has special charts and downloadable custom palettes to help site developers get the optimum results. If you don't want to go through that much trouble, don't worry. The Pocket Internet Explorer converts color graphics to the required 2-bit grayscale.

More information on developing sites for the Pocket Internet Explorer can be found at this address:

```
http://www.microsoft.com/windowsce/developer/data/designer/default.htm
```

The Wave

Even little 8-bit machines will get into the act with the new Wave graphical Web browser for the Commodore 128D. Those equipped with GEOS, the new Super CPU, and a RamLink can have up to 32MB of RAM running at 20MHz. On the downside, there is no support for plug-ins or programming standards higher than HTML 1.5.

WebTV

This is one of the first "Internet boxes" that is aimed at making the Net accessible to everyone. The WebTV Internet terminal is manufactured by Sony (`www.sony.com`) and Philips/Magnavox (`www.philips.com`). The WebTV has a browser-like interface and conforms to most HTML 3.2 standards. (See Figure 6.14.)

FIGURE 6.14.
Net appliances such as WebTV give the masses access to your Web site. Here is an actual WebTV browser screen within a Netscape Window.

The WebTV Network (`www.webtv.net`) acts as the Internet service provider, and the Web terminal connects between the phone line and the TV. Microsoft has acquired the WebTV Network, so you can probably expect some interesting developments in the future.

The WebTV browser gives Web designers the best of both worlds, and then some. It conforms to most Microsoft Internet Explorer and Netscape Navigator extensions. And, as if that weren't enough, there are even WebTV extensions to offer greater flexibility for your site.

Remember that designing pages for a TV audience is a bit different from designing for a regular computer audience. Most don't have storage space for downloading files, so sites aimed solely at downloads will probably be passed over. However, if you have a product to sell, some of these devices have credit card slots for quick transactions.

The WebTV software automatically resizes your existing Web pages to fit the meager 544×378 pixel display. Vertical scrolling is allowed, but the width is fixed at 544 pixels. The idea is to keep the Net experience similar to regular TV programming. After all, when was the last time you had to scroll around while watching *Baywatch*?

Your pages should be as friendly as regular TV programming, because that is just what they have become. Design your WebTV pages as if you are making a TV commercial or series. Try to keep the load times short because the TV generation is not used to waiting for its entertainment. Keep the screen simple and uncluttered. Besides the faster load times, be sure to make your pages active and interesting.

Although the original version of WebTV didn't support much Web technology other than JPEG and GIF, the newer software reportedly will. Features such as MIDI, MPEG, WAV, AIFF, and Shockwave audio are tagged for support.

Don't use red or white backgrounds because they can cause distortions and audio noise on some equipment. If you must use these colors for background, choose a variation other than the full value of white or red. Charcoal makes an excellent background color and can be created by using `<body bgcolor=#191919>`. Choose light-colored text on dark backgrounds. However, be aware that full white text can bleed over into the background on some equipment.

Avoid large images and images that are very busy or complicated with too much data. These images can be too hard to see on a TV set. Text that is smaller than the base font of your site also can be too difficult to read. Text on image buttons can be a legibility nightmare.

More information on WebTV extensions, style guides, and system requirements can be found at this address:

`http://webtv.net/primetime/`

Cable and telephone companies have big plans to jump into the Web appliance market, so Web developers can look forward to an ever increasing audience for their sites.

Summary

Great Web design starts by planning and knowing the capabilities of your target audience. The end user's level of expertise, connection speed, platform, and browser are just a few of the factors to consider during site development. Not only do you have to reach your audience, but your data also has to be accessible to them.

Remember that providing a text alternative to your home page will give more users access to your site. You can also offer "no frames" and low-resolution alternatives. Saving graphics at 640×480, 100 dpi, and 256 colors or less helps reduce load time. And load time is really the name of the game here. In this day of instant gratification, you will lose your audience quickly if your page doesn't get moving within just a few seconds. End users shouldn't have time to read a novel while they wait for your page to load. HTML standards set by the W3C and their Web-centric members such as Sun, Microsoft, and Netscape will help determine how your pages will look—now and in the future.

PART II

Basic HTML

IN THIS PART

CHAPTER 7

Structural Elements and Their Usage

by Rick Darnell

IN THIS CHAPTER

As explained in Part I of this book, "Understanding HTML," HTML documents are created by combining special markup codes called *tags*. Tags define the structure of the document and provide the framework for holding the actual content, which can be text, images, or other special content.

The elements covered in this chapter are called *structural* elements, because their primary purpose is to implement the form of the document. These include the document itself, the head and body sections, page titles, and other basic document identifiers.

NOTE

Actually, all HTML tags are structural in their purpose of defining the relations of elements to each other within the document. The reason this particular set is referred to as structural is because they really have no formatting purpose for what appears on the screen. All other tags control the appearance of the page in one form or another.

The easiest way to illustrate the purpose and form of structural tags is to begin with a page with no structure. (See Listing 7.1.)

Listing 7.1. This HTML document lacks any HTML structural form.

```
Missoula Rural Fire District Home Page
Welcome to the MRFD Home Page
Thank you for visiting our home page. From here, you can
link to several destinations, including fire safety
information, district budget, minutes from board meetings
and upcoming training.
```

If the code in Listing 7.1 were displayed on a browser, it would look something like the page in Figure 7.1. Note the complete lack of organization. In addition, it could also generate an error in a nonstandard Web browser because the type of content is not identified.

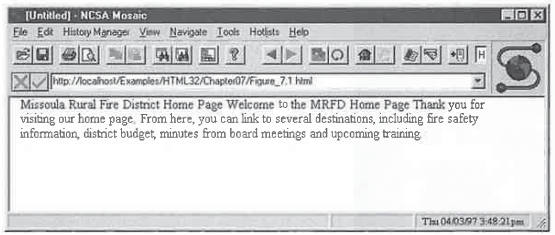


FIGURE 7.1.
Without some basic structural elements in place, this Web page is formless and pointless.

Good page design begins with good organization, and that's what the structural tags enable.

The <!DOCTYPE> Declaration

Theoretically, the <!DOCTYPE> tag should be the first tag in your HTML document. This tag tells the Web server what it's dealing with when it delivers the document; in turn, the server informs the browser what kind of tags it can expect inside.

The usage is

```
<!DOCTYPE HTML idString>
```

where *idString* is a specification for the version of HTML in use within the document. For example, the following code identifies a document built according to the standards laid out for HTML 3.2:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
```

TIP

A reasonably complete list of DOCTYPE declarations is found at the following address:

`http://ugweb.cs.alberta.ca/~gerald/validate/lib/catalog`

In reality, use of <!DOCTYPE> is difficult. Many variations of HTML are in use, the standards are continuously evolving, and browser implementations continue to change the rules by which we play. The mix of extensions in a document can include tags from HTML 2.0 and 3.2 along with customized extensions from both Netscape and Microsoft. This is where the realm of universal standards and the rules of the marketplace meet head-to-head. As a practical matter, many browsers ignore <!DOCTYPE>, or just ignore the tags they don't understand, regardless of whether or not they're part of the standard expressed for the document.

NOTE

This tag is also used in customized forms by applications that need to identify product-specific content, such as Microsoft FrontPage.

With this bit of preamble out of the way, it's time to move on to the document itself.

Setting the Boundaries with <HTML>

The first tag to be concerned with is <HTML>, which is the next line in a document after <!DOCTYPE>. It is paired with </HTML> to encase all other tags in an HTML page, and as such, the two mark the absolute beginning and end of the file.

The usage is

```
<HTML>
...document and tags...
</HTML>
```

where *document and tags* are the rest of your HTML document. Using these tags with the first example in this chapter results in the code shown in Listing 7.2.

Listing 7.2. The first step in providing a structure for an HTML document is marking its beginning and end.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
Missoula Rural Fire District Home Page
Welcome to the MRFD Home Page
Thank you for visiting our home page. From here, you
can link to several destinations, including fire safety
information, district budget, minutes from board meetings
and upcoming training.
</HTML>
```

It is possible to create an HTML document without the <HTML> tags, and most browsers will know what to do with the document after it's loaded. However, there are a couple of good reasons to use them. First, HTML isn't the only markup language on the Web. Cousins of HTML, such as Extensible Markup Language (XML), are interpreted in slightly different ways. In the absence of this tag, most browsers default to HTML. Those that don't become confused when handed a document without any clear indication of type, resulting in a page that is displayed in an unpredictable manner, or one that refuses to load at all.

Second, using the <HTML> tag is good style and shows that whomever built the document had some idea what to do. And if you're taking the time to read this book, you know what to do.

The appearance of Listing 7.2 won't change on the browser with the addition of the opening and closing <HTML> tags, but at least now its boundaries are delineated and the browser knows what to do with the file after it's loaded.

Now that the document's type and boundaries are marked and identified, it's time to divide the HTML page into two operational parts—the header and the body. The tags and their compatriots, which govern these two major sections, are explained next.

The HEAD Element

Theoretically, every document has a header and a body. The header of the document is where global settings are defined; it is contained within the <HEAD> and </HEAD> tags.

The usage is

```
<HEAD>
header content
</HEAD>
```


where *header content* includes one or more items from the six tags used exclusively within the header portion of an HTML document. It is also a favorite place to include scripting language function definitions.

The addition of these tags to our ongoing example results in the code shown in Listing 7.3.

Listing 7.3. The HTML document has received an extra boost of structure with the <HEAD> tag.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
Missoula Rural Fire District Home Page
</HEAD>
Welcome to the MRFD Home Page
Thank you for visiting our home page. From here, you
can link to several destinations, including fire safety
information, district budget, minutes from board meetings
and upcoming training.
</HTML>
```


Displayed on a browser, the results are made slightly clearer by the elimination of the first line, which has disappeared completely. It is destined for use within the next header tag.

Headers also serve another important function. Using the HTTP protocol, it's possible to download the header information only. The vast majority of users aren't aware of this capability and probably don't care. It is used primarily by search engines and automated robots, which can download a header and get some basic information about the page—title, file format, last-modified date, keywords— without spending the extra time to load or look at the rest of the document.

Giving Your Page a <TITLE>

Probably the most commonly used HEAD feature in HTML is the <TITLE> tag.

Its syntax is

```
<TITLE>text</TITLE>
```

where *text* is a short, one-line name for the document that is displayed in the browser's title bar. Without a title, most browsers default to the HTML filename. Because filenames are not always terribly descriptive and often long and clunky, it's a good practice to provide a title for all HTML documents.

The addition of the <TITLE> tag to your HTML document results in the code shown in Listing 7.4 and the page shown in Figure 7.2.

Listing 7.4. Adding a title to the HTML document reinforces the structure.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Missoula Rural Fire District Home Page</TITLE>
</HEAD>
Welcome to the MRFD Home Page
Thank you for visiting our home page. From here, you can
link to several destinations, including fire safety
information, district budget, minutes from board meetings
and upcoming training.
</HTML>
```

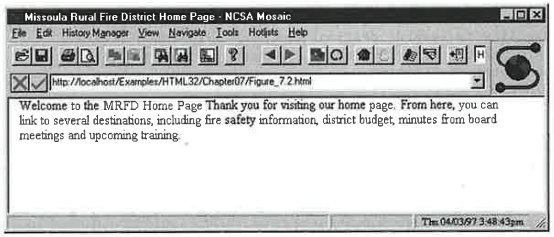


FIGURE 7.2.
With the addition of some structural elements and a title (see the top bar of the browser), this Web page is beginning to take a recognizable shape.

Only one title is allowed per document (as covered in the following Tip box), and its size is limited by the size of the user's browser window. You can include a title that is as long as you want, but it is truncated in the browser's title bar if it stretches beyond the limits.

TIP

To make your titles easier for the user, make sure they are short and to the point, but not so short as to be vague. For example, "MRFD" is not a good title. What does MRFD mean? Where are we? What are we doing here? On the other hand, "Missoula Rural Fire District Home Page" is a good title. You've identified the page and its purpose, which is what the user needs to know.

Keep in mind that many browsers keep their name in the title bar along with your title, leaving less space for the title of your page. Keep the title long enough to be useful but short enough to fit.

ONE TITLE ONLY, PLEASE

Once upon a browser, when Netscape 1.0 was the biggest show in town, there was a slight glitch with the <TITLE> tag. You could stack up a series of tags like this:

```
<TITLE>M</TITLE>
<TITLE>MR</TITLE>
<TITLE>MRF</TITLE>
<TITLE>MRFD</TITLE>
<TITLE>MRFD H</TITLE>
<TITLE>MRFD Ho</TITLE>
<TITLE>MRFD Hom</TITLE>
<TITLE>MRFD Home</TITLE>
<TITLE>MRFD Home </TITLE>
<TITLE>MRFD Home P</TITLE>
<TITLE>MRFD Home Pa</TITLE>
<TITLE>MRFD Home Pag</TITLE>
<TITLE>MRFD Home Page</TITLE>
```

The result would be a title that gradually built up across the title bar. This aberrant behavior was fixed in the next version, and the result was pages with some strange behavior. Different browsers interpret multiple titles in different ways. For example, Netscape 4.0 picks the first title from the list, and Internet Explorer 3.01 uses the last tag. For the preceding example, your document would appear to have a title of "M" or "MRFD Home Page," depending on which browser the user preferred.

The moral of this little story? Use one title in your document so that the browser won't have to guess, and your users won't have to wonder what's going on.

A <BASE> for Hyperlinks

This tag controls the actions of relative hyperlinks in the body of the document. Relative links within a document by default refer to the same server where the page is located. However, with the <BASE> tag, you can define that the relative links are resolved relative to a different location, whether it's another directory on the host or a completely different server. (For more information on hyperlinks and anchors, see Chapter 11, "Linking Documents and Images.")

The <BASE> tag takes a single attribute, HREF, with the following syntax:

```
<BASE HREF="protocol://servername/path/">
```

In the preceding line, *protocol* is a valid Internet communication standard, such as HTTP; *servername* is a server name or address such as www.wossamotta.edu or 89.123.32.21; and *path* is any additional mapping on the server. The path is an optional value to the URL. If the path is included by itself, it refers to the host server.

For example, a page for an educational institution is located on the server at www.frostbitefalls.org. The page is copied straight off the university's server at www.wossamotta.edu and placed in the new location. Any relative links will no longer work because they're referring to locations at the home location.

If the user clicks on a link to look at the football schedule, the browser takes the content of the link, `/football/schedule.html`, and combines it with the host, `www.frostbitefalls.org`, to create `http://www.frostbitefalls.org/football/schedule.html`. But, alas, Frostbite Falls doesn't have this directory, and the user receives the dreaded `404 Not Found` error.

The `<BASE>` tag circumvents this problem, like so:

```
<BASE HREF="http://www.wossamotta.edu/">
```

Now, when a user clicks on a link for the football page, the base information is substituted for the host information, which results in `http://www.wossamotta.edu/football/schedule.html`—and now the information on the match-up versus Tick Tock Tech is easily accessed.

<ISINDEX> for Searching with CGI

The `<ISINDEX>` tag is used during interactive searches of a Web page. Generally, it is placed in a document built by a special CGI script. It generates a prompt for the user to enter a search string. (See Figure 7.3.)

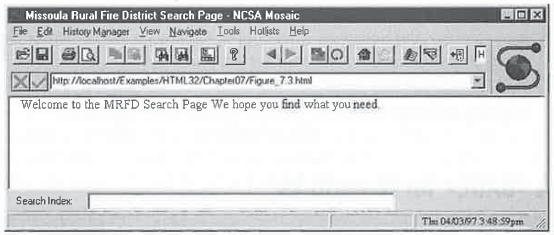


FIGURE 7.3.
The `<ISINDEX>` tag results in a slightly different display, depending on the browser. In Mosaic, an additional bar is added to the bottom of the browser with a "Search Index" prompt.

Any text provided by the user is appended to the document's URL and passed back to the CGI script for processing in the form *`URL?search1+search2+...+searchN`*. This is the standard syntax for return values to CGI scripts, with the question mark indicating the beginning of the string, and each plus sign marking a space.

The `<ISINDEX>` tag is typically inserted and handled exclusively by the CGI script. Customizing the search interface is accomplished by providing scripts with customized forms. (For more information on building forms, see Chapter 15, "Building and Using HTML Forms.")

This tag has one attribute, `prompt`, which changes the default message for the text field. The syntax is

```
<ISINDEX prompt="string">
```

where *`string`* is the new message. The length of the `prompt` string varies with individual browsers. For example, Navigator uses an HTML form to gather the index information, and Mosaic uses a custom browser element at the bottom of the screen. For this reason, try to keep the message as short as possible (about 35 characters) to retain compatibility with as many browsers as possible.

Showing Relationships with <LINK>

The `<LINK>` tag has been around since the early days of HTML, although it has yet to earn acceptance or implementation from many browsers. In its current definition, `<LINK>` provides a media- and platform-independent method for defining relationships between the current HTML page and other documents and resources. All popular browsers, with the exception of Mosaic 3.0, still ignore the tag.

In theory, `<LINK>` is used to create document-specific toolbars or menus, to control how collections of HTML files are connected when printed, and to link associated resources such as style sheets and scripts. Five attributes are used with `LINK`:

Attribute	*Description*
`href`	The URL of the linked document. This is provided in standard URL format, but it is typically a relative URL.
`rel`	This attribute and its value indicate the relationship of the current document to the document referenced in `href`. The values that are used include `Precede`, `Annotation`, `Present`, `History`, and `Made`. For example, the following line indicates that annotations (margin notes, footnotes, and so on) for the current document are located in a file called `annotations.html`: `<LINK rel=Annotation href="annotations.html">`
`rev`	This attribute is similar to `rel`, except it indicates a reverse relationship between the document and the URL. For example, the following line indicates that the current document has annotations for the document specified in the URL: `<LINK rev=Annotation href="chapter1.html">` It accepts the same values as `rel`.
`title`	This attribute, related to `href`, is the name of the document referenced in `href`, and it must match the value contained in the `<TITLE>` tags in order to be valid.
`name`	Assigning a meaningful name to the link allows for easier reference elsewhere in the document.

Currently, the only supported use for `<LINK>` is to specify style sheets as external to the document.

For more information on using <LINK> with cascading style sheets, see Chapter 19, "Introducing Cascading Style Sheets."

Using <META> to Give More Information

Web servers send their own header with HTML documents to help clients interpret the document. This header is separate and different from the header you've been working on in this section.

Like the <!DOCTYPE> tag, the <META> tag is used to pass additional information about how the document should be handled, and it is often used with add-on content provided by applications such as Microsoft FrontPage and Macromedia Backstage. In addition, it is used to provide extra information about the document that can be used by search engines to classify and identify the document without downloading the entire document.

Three attributes accompany this tag:

- HTTP-EQUIV
- NAME
- CONTENT

Sending Information: HTTP-EQUIV and NAME

The first two—HTTP-EQUIV and NAME —serve the same basic purpose, with one important difference. Any <META> tag using HTTP-EQUIV is added to the response header supplied to the browser. If the tag uses the NAME attribute, the information is available for reference in the document header but not included in the server-generated response header.

Keep in mind one important consideration when using the HTTP-EQUIV attribute. It should never override a standard header value such as last-modified or contenttype. This could result in a conflict with the server that prevents delivery of the document by the server or interpretation by the browser. Here are some examples of HTTP-EQUIV:

```
<META HTTP-EQUIV=refresh CONTENT=60>
<META HTTP-EQUIV=keywords CONTENT="fire department public safety">
<META HTTP-EQUIV=reply-to CONTENT="mrfd@montana.com">
```

The following list shows three of the most used variables for HTTP-EQUIV:

- Refresh implements a process called *client-pull*, which directs the browser to reload the document after the number of seconds specified in content. It is used in situations when the document is updated on a periodic basis. Content can also take the form *N*; URL=*url*, where *N* is the number of seconds to wait, and *url* is the new URL to load.
- Keywords specifies a list of words separated by spaces, which are used by some search engines to classify the document for quicker retrieval. Many search engines load only the head and not the body, so it's important to include the keywords in the <META> tag so that the engine knows what is found in your document.

- Reply-to provides an e-mail address at which users can respond to the author or party responsible for the page. Its display is typically triggered by the server, which adds it as a server-side include. This attribute is not commonly used.

WHAT IS CLIENT-PULL?

Client-pull provides a mechanism for pages to automatically reload after a certain amount of time has passed, or for a series of pages to automatically load themselves with a pause between them. If you want the browser to reload the current page in four seconds, you add this tag to your HTML page:

```
<META HTTP-EQUIV="Refresh" CONTENT=4>
```

If the value of CONTENT is 0, the page is refreshed as fast as the browser can retrieve it, which could be rather slow if the user has a slow or poor-quality connection. It's definitely not fast enough for any sort of quality animation.

After the REFRESH attribute is added to a page, the browser reloads it *ad infinitum*. To stop the process, you need to provide a hyperlink to another page without a client-pull tag.

Continuously loading the same page is useful for pages that are updated constantly—such as documents containing stock quotes or sports scores. However, you can also use client-pull to load a different page. Continuing the process of loading a new page allows you to automatically lead a user through a series of slides or instructions. The CONTENT attribute is modified to provide this capability:

```
<META HTTP-EQUIV="Refresh"
CONTENT="8;URL=http://www.mrfd.com/safety/tip2.html">
```

The URL must be a full URL. Relative URLs consisting of only pathnames are not allowed. Inside the target page, you can include a pointer to the next page, and so on. This technique allows you to load any number of pages in a sequence. However, it's still a good idea to provide a link to move out of the automatic process so that your readers aren't forced to sit through the entire show.

The NAME attribute is used to provide other information about the document that might be useful to someone looking at the document but not critical to deliver in the header. Here are some examples:

```
<META NAME=author CONTENT="Dave Herzberg">
<META NAME=description CONTENT="Home page for MRFD">
<META NAME=copyright CONTENT="Copyright 1997, Missoula Rural Fire Dist.">
```

The attributes in the following list are used more often by HTML editors:

- Author identifies the person who created the page and sometimes includes the name of the HTML editor.
- Description provides a one-line explanation of the page or its use. It is sometimes used by search engines to provide a summary of the page when it displays search results to a user.

- `Copyright` is the official copyright notice for your page. Anything you create is subject to protection by copyright law, regardless of whether it has this statement. However, including this statement prevents the excuse of ignorance ("I didn't know") from would-be plagiarizers.

As seen in the preceding examples, `CONTENT` is the actual value that is contained in the tag. Although most browsers and servers don't require quotation marks around the value, it's a good practice to use them. This removes ambiguity and opportunity for misinterpretation, especially for values requiring more than one word.

Defining a <STYLE> For the Page

The `<STYLE>` tag was included without a recommended implementation by the World Wide Web Consortium. It was intended to be a placeholder for introducing style sheets and client-side scripts in future versions of HTML. True to the form of the Web, the future is here, pushed along by Netscape and Microsoft.

You can find more about style sheets and using the `<STYLE>` tag in Chapters 20 through 22, which cover cascading style sheets (CSS) and their usage, JavaScript-based style sheets, and other alternatives to CSS.

The BODY Element

Like the `<HEAD>` tag, the `<BODY>` tag's primary purpose is to delineate the main portion of the document—the part that is seen by the user. Its usage is

```
<BODY attributes>
...document contents...
</BODY>
```

where *`attributes`* includes any, all, or none of the six attributes controlling basic document display attributes, and the *`document contents`* is any valid HTML content, including text, forms, graphics, special content, and so on.

TIP

If omitted, the `<BODY>` tag is assumed by the browser after the `<HEAD>` tag. If the `<HEAD>` tag is also absent, the `<BODY>` tag is assumed immediately after the opening `<HTML>` tag.

In addition to marking the body of the text, the `<BODY>` tag is also used to set various display attributes for the document, including background colors and images, and the color of various types of text. These attributes are discussed in the following sections.

Background Wallpaper Images: BACKGROUND

The first attribute to mention is `BACKGROUND`, which is used to identify a graphics image to tile behind the document (as shown in Figure 7.4), much like wallpaper forms a background for the crayon scribbles of a small child.

FIGURE 7.4.
Use the `BACKGROUND` *attribute to specify a graphics image for tiling in the background of the document.*

The syntax is

```
<BODY BACKGROUND="[url][path][filename]>
```

The first two values, *`url`* and *`path`*, are optional and default to the server and directory that contain the HTML document if omitted. The *`filename`* should be a GIF or JPEG image to ensure compatibility with all browsers. PCX and BMP are not universally accepted standards on the Web.

TIP

For best results, pick an image rendered in light, muted colors. Light grays tend to work best, although readable pages have also been produced with light blues and yellows. Backgrounds done in bright colors or designs are very distracting for readers and typically annoy them into not spending much time looking at what you produced.

Background Colors: BGCOLOR

The next attribute sets the background color of the entire document, much like the `BACKGROUND` attribute sets the wallpaper. Its usage is

```
<BODY BGCOLOR="colorValue">
```

where *`colorValue`* is a recognized color literal such as `black` or `aliceblue`, or a hexadecimal triplet representing the red, green, and blue mix for the color. A pound sign (#) must precede the value if a hexadecimal number is used.

WORKING WITH COLOR DEFINITIONS

The list of color literals is fairly extensive and should be more than adequate for most pages. Those who want precise control over colors can use the hexadecimal triplet definition.

A hexadecimal triplet is a set of three double-digit hexadecimal numbers. The first number corresponds to red, the next to green, and the last to blue. Each number ranges in value from `00` to `FF`, representing decimal `0` to `255`. For example, `00FF00` is green, and `FFFFFF` is white. The total absence of color is `000000`, or black.

In the ongoing battle between page authors and Web users, a custom color is not always accepted on browsers. Both Internet Explorer (2.0 and later) and Navigator (2.0 and later) include options to override a background color attribute on the page with the user's preferences set in the browser. If it's absolutely imperative that you have a specific color as the background, you need to use the `BACKGROUND` attribute with an image consisting solely of the desired color block.

In the same vein, inherent limitations are also set by the user's equipment. If old VGA monitors or black-and-white laptops are still in use, your color probably won't translate well. To be safe, let the user's machine set the background.

TIP

If the image set with `BACKGROUND` is a transparent GIF file, any color set with `BACKGROUND` will show through it.

Document Text Color: TEXT

The `TEXT` attribute sets the default color of any nonhyperlink text in the document. The default value is black. The usage is

```
<BODY TEXT="colorValue">
```

where *`colorValue`* is a recognized color literal such as `black` or `aliceblue`, or a hexadecimal triplet representing the red, green, and blue mix for the color. A pound sign (#) must precede the value if a hexadecimal number is used.

Take care when setting the default text color to make sure it doesn't conflict with the colors in the background. For example, red text on a green background might seem like a good idea at Christmas time. In reality, it's never a good idea. Viewing the results is like rubbing your eyes with a belt sander. On the other hand, proven combinations such as yellow on dark blue fare very well and can actually brighten a page beyond the typical default of black on medium gray.

TIP

Like the `BGCOLOR` attribute, any of the color attributes to the `<BODY>` tag can be overridden by any browser, such as recent versions of Internet Explorer, Navigator, and the various versions of Mosaic. The fact that you've picked a specific color scheme doesn't guarantee that the user will see it.

The `TEXT` color does not take priority over other formatting settings. Settings contained within the `<FONT>` tag override the values set in text attributes, including `TEXT`, `LINK`, `ALINK`, or `VLINK`.

Hyperlink Color: LINK

In most browsers, the default value for hyperlinks not yet visited is dark blue. You can change this by using the `LINK` attribute. Its usage is

```
<BODY LINK="#0000FF">
```

where *`colorValue`* is a recognized color literal such as `black` or `aliceblue`, or a hexadecimal triplet representing the red, green, and blue mix for the color. A pound sign (#) must precede the value if a hexadecimal number is used.

Most browsers also specify an underline attribute for hyperlinks, but as yet that setting cannot be changed from within the HTML `<BODY>` tag. Underlining for a hyperlink is accessible from a style sheet, which is discussed in Chapter 19.

Active Hyperlink Color: ALINK

Anyone who has used a browser for any length of time notices a color change when clicking on a text-based hyperlink. When the mouse button is depressed, the text color changes. The color varies from computer to computer and browser to browser, but the change is there—something different from both the link color and the visited link color. The `ALINK` (for *Active Link*) attribute is used to change this value from its default, which is usually red. The usage of this attribute is

```
<BODY ALINK="#FF00FF">
```

where *`colorValue`* is a recognized color literal such as `black` or `aliceblue`, or a hexadecimal triplet representing the red, green, and blue mix for the color. A pound sign (#) must precede the value if a hexadecimal number is used.

TIP

You can have a little fun with different settings for this attribute, but avoid setting a color equal to the background color. Having the link disappear for a moment is disconcerting to many users and makes them wonder about your intentions. After all, what are you trying to hide?

Visited Hyperlink Color: VLINK

The last attribute of the <BODY> tag sets the color of hyperlink text after the hyperlink has been visited. This helps users differentiate between where they've been and where they haven't been. Its usage is

```
<BODY VLINK="#FF0000">
```

where *colorValue* is a recognized color literal such as `black` or `aliceblue`, or a hexadecimal triplet representing the red, green, and blue mix for the color. A pound sign (#) must precede the value if a hexadecimal number is used.

This color should be a contrasting color to the unvisited link color (LINK); otherwise, it's very hard for users to tell where they've been. Unless there is an overwhelming reason, don't set VLINK and LINK to the same value.

TIP

Keep in mind the differences in quality that Web users' displays are likely to have—everything from Super VGA to old EGA and CGA monitors. On lower-resolution monitors, the range of colors is much more limited. So, even though you've defined a light blue for VLINK and dark blue for LINK, the limitations of a graphics display could force both to a basic blue with no difference.

WHEN DOES THE COLOR CHANGE BACK?

Each browser handles this differently. Most browsers allow the user to set expiration dates on visited links, after which time they revert back to the original color. As a very general rule, this period is from two weeks to 30 days after the link is clicked, and it can be reset by the user to as little as a day.

The SCRIPT Element

This particular element is a bit of a free spirit, and it is used in two ways. In either application, it indicates a specific scripting language that is being used in the document. At present, two scripting languages are in widespread use on the World Wide Web: JavaScript and Visual Basic Script (VBScript). This element's syntax is

```
<SCRIPT LANGUAGE="language">
```

where *language* is VBScript or JavaScript. If used without a closing tag, it indicates the scripting language used for any scripts throughout the entire document. More information on specifying and creating scripts is provided in Chapter 28, "Integrating JavaScript and HTML," and Chapter 30, "Integrating ActiveX and VBScript."

In its typical usage, <SCRIPT> is used in combination with a closing tag, </SCRIPT>, to contain a section of programming code. If this code is placed in the header, it's interpreted before the rest of the document is loaded. This is a common practice for segments of code that serve as functions, where it's imperative to have the function loaded and available before anything in the document has a chance to invoke it.

Code that is designed to execute when the document is loaded or to accept interaction from the user is placed in the appropriate part of the document, such as forms and hyperlinks.

Browsers that don't support scripting languages probably won't recognize the <SCRIPT> tag. If this happens, the contents of the tag are displayed as regular text, which results in a lot of strange reading for the user. To avert this problem, use this simple workaround:

```
<SCRIPT>
<!--
script stuff
-->
</SCRIPT>
```

When you encase the script in HTML comment tags within the <SCRIPT> tags, an incompatible browser will ignore the script contents. The script will ignore the comment tags and process normally.

Summary

Structure tags set the stage for the rest of your HTML and provide its basic framework. They mark the beginning and end of the file, header, and body sections. Additional tags and attributes within the header set basic behaviors, define a title, and pass information to the browser. By using the <BODY> tag, you can also set background images and colors, and set default colors for various text items on the page.

Although these tags serve an important purpose in your HTML document, they're not the fun stuff. The meat and potatoes are next.

CHAPTER 8

Text Alignment and Formatting

by Rick Darnell

IN THIS CHAPTER

In Chapter 7, "Structural Elements and Their Usage," you saw how to build a frame for an HTML document. As you learned, the basic structure includes the boundaries of the HTML page, a header and a footer, and a handful of other items to provide a framework for the page content. With that little bit of HTML, you have more than enough to build a Web page.

There's a small problem, however. You could put all of your text between the two body tags, and the user could read it. But it would be a tedious task. Web browsers don't look at text the way a person does. When the browser sees the end of a line in your document, it adds a space and keeps right on going. The result is one long string of words on the page that is neither attractive nor friendly to use.

The HTML elements presented in this chapter are designed to do everything for your page. There are tags for headings, addresses, and quotations. There are tags to make the text bold, italic, and green. There are tags to begin new paragraphs, start new lines, and add horizontal lines to help divide the page.

This chapter presents the workhorses for organizing and formatting your document, and you'll find that virtually every page on the Web uses at least one or two of these tags, and many make liberal use of more than that. Read on to find out how to use HTML text formatting elements to give visual and logical structure to your pages.

Headings <H1> through <H6>

You don't begin a newspaper story with the lead paragraph, you begin it with a headline. And that's where this chapter begins with the text formatting tags. The six HTML heading styles are a way of showing the level of importance among different parts of your page, much like this book uses different levels of headings to visually organize the information.

The syntax is

```
<Hn [align=left¦center¦right]>Heading Text</Hn>
```

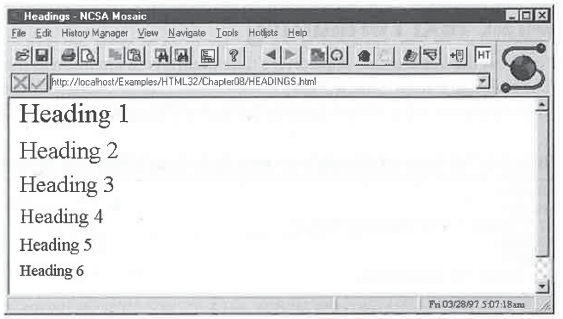
where *n* is an integer from 1 to 6 indicating the level of heading used, with 1 being the most important and 6 being the least. The optional `align` attribute controls the horizontal alignment of the heading. If omitted, it defaults to left alignment.

Although browsers vary in the size and typestyle given to the six heading levels, every browser follows the basic rule of giving the biggest and boldest style to `<H1>`, and the smallest and most insignificant style to `<H6>`. (See Figure 8.1.)

TIP

On some browsers, the lowest heading levels are actually smaller than the body text. The result appears slightly odd.

FIGURE 8.1.
HTML heading styles show a descending level of emphasis and importance for the text they precede.

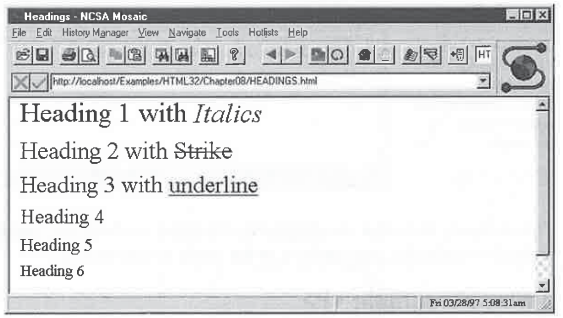


Headings allow for physical formatting within the container tags. You can set the style (bold, italic, or underlined) and typeface and size within a heading to give it additional emphasis.

```
Next on our site is <H1>Reflections on <I>The Devils
of Loudun</I></H1> followed by pictures of my cat.
```

Changing the style is possible within any of the heading styles, regardless of their level (as shown in Figure 8.2).

FIGURE 8.2.
To further control the appearance of a heading, you can use physical formatting to control all or part of the text within.



In Figure 8.2, note how the browser handles the text before and after the heading tags. A heading is always placed on its own line, even if it's placed inline with other material. A new paragraph is started for the heading, and any material following it is placed on the next new line.

Basic Text Formatting

Now that the headings are in place, it's time to turn your attention to the rest of the document. The next step is to start breaking the text into paragraphs. Listing 8.1 appears to be a Web page that is broken into logical paragraphs, but look at Figure 8.3 to see how a browser interprets it.

Listing 8.1. This document attempts to use extra hard returns to force each statement to its own line.

```
<HTML>
<HEAD>
<TITLE>Hot Water Safety</TITLE>
</HEAD>
<BODY>
Hot water is dangerous.

Turn down the hot water heater thermostat.

Always supervise children in the bathtub.

Don't encourage children to play around the tub.
</BODY>
</HTML>
```

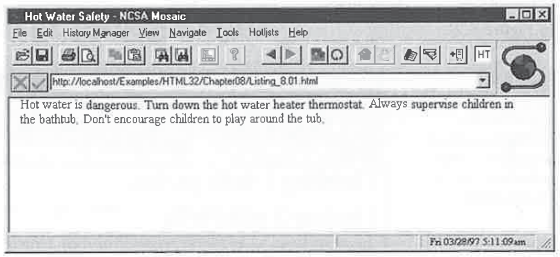


FIGURE 8.3. *The code in Listing 8.1 creates a page that is formatted differently in the browser than it appears in the listing. The browser has ignored the extra line breaks and ran the lines together into one long paragraph.*

This obviously isn't what the designer had in mind, so check out the next three elements for methods to make the page conform to the way it should appear.

A New Paragraph: <P>

The first way to break text into paragraphs is to use the paragraph tag—`<P>`. The syntax is

```
<P [align=left¦center¦right>...Text...</P>
```

where *Text* is a line or paragraph text, which should begin on a new line and remain together, and `align` sets whether the *Text* is aligned to the left (default), center, or right of the browser window. The closing tag is optional and can be omitted.

Using this tag on the preceding listing results in the code shown in Listing 8.2 and the page shown in Figure 8.4.

Listing 8.2. This page is now divided into separate lines by defining each line as its own paragraph.

```
<HTML>
<HEAD>
<TITLE>Hot Water Safety</TITLE>
</HEAD>
<BODY>
<P>Hot water is dangerous.</P>

<P align=right>Turn down the hot water heater thermostat.</P>

<P>Always supervise children in the bathtub.</P>

<P align=right>Don't encourage children to play around the tub.</P>
</BODY>
</HTML>
```

Notice the behavior of the alignment. A new paragraph supersedes any alignment settings in the previous paragraph. And, if an alignment is not specifically set, it defaults back to the left.

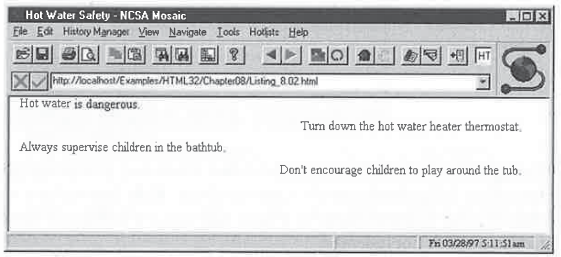


FIGURE 8.4. *The code in Listing 8.2 is now interpreted the way you want it to be by using the <P> tag to mark each sentence as its own paragraph. Note that the extra return between lines is still ignored.*

A New Line:

A line break tag, `<BR>`, is similar to a paragraph tag, but it behaves slightly differently. It starts a new line within the current paragraph, but it doesn't start a new paragraph. The syntax is

```
<BR [clear=left¦right¦all]>...Text...
```

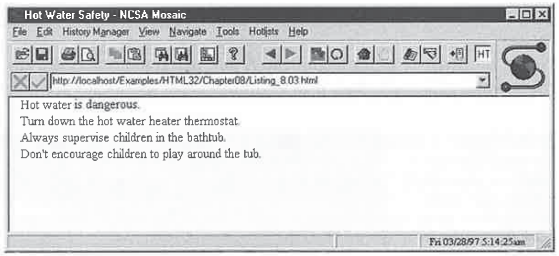
where *Text* is the material that should appear on the next line and the optional `clear` attribute defines how the following material should flow around floating images. An ending tag is not used.

Used with the preceding example, this tag creates the same basic effect as using a paragraph tag. (See Listing 8.3 and Figure 8.5.)

Listing 8.3. Instead of new paragraphs, each line is started with a line break tag. This provides the same basic effect while keeping all of the text within the same paragraph.

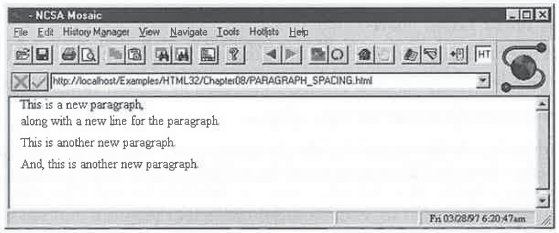
```
<HTML>
<HEAD>
<TITLE>Hot Water Safety</TITLE>
</HEAD>
<BODY>
<P>Hot water is dangerous.

<BR>Turn down the hot water heater thermostat.

<BR>Always supervise children in the bathtub.

<BR>Don't encourage children to play around the tub.</P>
</BODY>
</HTML>
```

FIGURE 8.5.
The code in Listing 8.3 creates a page with each sentence on its own line, although now the lines are slightly closer together because they're still within the same paragraph.



The big difference you'll notice between line breaks and new paragraphs is line spacing. (See Figure 8.6.) A line break uses the same spacing as if the line had just scrolled down from the preceding line. A new paragraph typically uses an extra half-line of space because the first line of a paragraph is not indented.

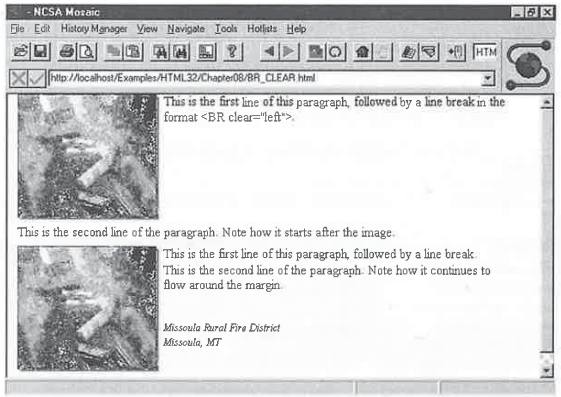
FIGURE 8.6.
The primary difference between a line break and a new paragraph is how much space is left between the lines.



Any alignment or other text formatting set previous to the line break within the same paragraph is carried to the new line.

The `clear` attribute is used to move past floating images on either margin. A value of `left` moves the text after the line break past any floating images on the left margin (as shown in Figure 8.7), while a value of `right` performs the same function for floating images on the right. As you might expect, `all` does the same for floating images on either margin. Other options for causing text to flow around an image are covered in Chapter 12, "Adding Images to Your Web Page."

FIGURE 8.7.
Using the `clear` attribute allows text to move past a floating image.



Standard inline images don't "float," so they're not affected by the `clear` attribute. An inline image, by definition, appears only on the line where it was inserted.

Preformatted Text, Spaces and All: <PRE>

The last method for making your four lines of text appear the way you want is to mark the whole section as preformatted text. The syntax is

```
<PRE [width=characters]>...Text...</PRE>
```

where *Text* is any text, including returns, spaces, and other hard formatting. The `width` attribute tells the browser how wide of a space, in *characters*, to leave for the text.

The `width` attribute allows the browser to choose an appropriately sized font or indent as necessary so that the text is properly displayed. It is not supported by many browsers.

There are several key differences between this method and the two preceding tags. Look at Listing 8.4 and Figure 8.8. All of the text is contained by the <PRE> tags, which causes the browser to display everything in between *exactly as it finds it.* It also uses a monospaced font for display, which facilitates formatting of the text into rows and columns for data presentation.

Listing 8.4. Instead of new paragraphs, each line is started with a line break tag. This provides the same basic effect while keeping all of the text within the same paragraph.

```
<HTML>
<HEAD>
<TITLE>Hot Water Safety</TITLE>
</HEAD>
<BODY>
<PRE>Hot water is dangerous.

   Turn down the hot water heater thermostat.

      Always supervise children in the bathtub.

         Don't encourage children to play around the tub.</PRE>
</BODY>
</HTML>
```

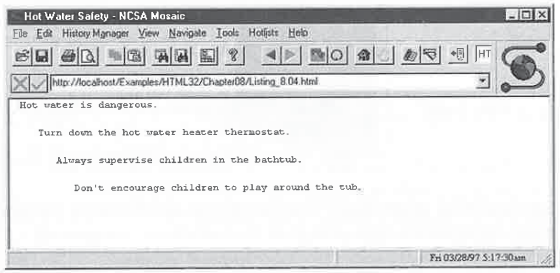


FIGURE 8.8.
The code in Listing 8.4 uses preformatting to display the lines exactly as they were typed in the file. The monospace font preserves character spacing for accurate display of indents and other formatting.

TIP

For browsers that aren't compatible with tables, preformatted text is an effective way to display a table made out of textual elements representing the border lines.

The monospaced font helps when preserving the line breaks and spaces inserted by the user. If you start a new line, insert five spaces and then start typing; that's exactly how your text appears using the <PRE> tag. Browsers disable automatic word wrap to further ensure that your text is displayed exactly as you typed it.

Obtaining Preformatting the Old-Fashioned Way: <XMP>, <LISTING>, and <PLAINTEXT>

An old-fashioned way of doing something on a Web page usually means one thing: The old way is obsolete, and its tags are just kept around to be compatible with as many pages as possible. This is the category to which <XMP>, <LISTING>, and <PLAINTEXT> belong.

They work the same as <PRE>, requiring an opening and closing tag. However, according to the HTML 3.2 specification, browsers can support these for backward compatibility. To be safe, avoid usage of these items.

Breaking a Document into Divisions: <DIV>

The <DIV> element is used to structure an HTML document into a series of divisions. Its syntax is

```
<DIV [align=left¦right¦center>...Text...</DIV>
```

where *Text* is one or more lines of text that comprise the division. The align attribute is optional; it sets the horizontal alignment for text within the block and behaves the same as the attribute used in the <P> tag.

The <DIV> tag is a block element that acts very similarly to a <P> tag. If a <P> tag doesn't have a closing </P>, <DIV> effectively closes it and starts a new paragraph. Other than this behavior, <DIV> doesn't generate paragraph breaks before or after its placement.

USING <CENTER> INSTEAD OF <DIV align=center>

The <CENTER> tag is identical to <DIV align=center>. This duplication occurred because the <CENTER> tag was introduced by Netscape at about the same time the <DIV> element was added to the HTML 3.0 specification.

Both tags are included in the HTML 3.2 specification, although using the align attribute of <P> or <DIV> is the preferred way to center text. The <CENTER> tag was placed in the specification due to its widespread implementation.

Formatting Text by Its Usage

One of the ways HTML defines the appearance of text is by its use within the document. This is a little difficult to understand for people just getting started with putting together pages.

In the traditional world of word processing and publishing, people tend to think in physical styles—font, size, bold, italic, and so on. But remember that HTML is a way of defining structure as much as appearance. HTML is a simple markup language used to create hypertext documents that are portable from one platform to another. Within this limitation, each browser is

given a lot of latitude in how it interprets tags. The HTML 3.2 specification includes recommendations on preferred appearances, which browsers deviate from based on the needs and limitations of their respective platforms.

With that preamble out of the way, it's time to introduce the logical tags. These are the tags that describe the usage for a piece of text, rather than the physical attributes it should have. It's much the same as defining a style on your word processor for addresses. Every time you have an address, you apply the "address" style. You cease to think about the text in physical formatting by substituting the new style.

The big difference between your word processor and the browser is that the browser decides how it's going to display the address style, not you.

TIP

For each of the following styles, the preferred browser interpretation is listed even though this may vary across applications and platforms.

Basic and Strong Emphasis: <EM> and <STRONG>

There are times when you need to make sure that the reader doesn't miss the point of your message. And in order to do that, you need to emphasize a word or more in a sentence. HTML provides the emphasis tag to fill this need. The syntax is

```
<EM>...Text...</EM>
```

where *Text* is the word or words to emphasize. Italics are the recommended rendering of emphasized text.

Related to the emphasis tag is the strong emphasis tag (<STRONG>), which should relate a higher level of importance to the reader. The syntax is

```
<STRONG>...Text...</STRONG>
```

where *Text* is the word or words needing extra emphasis. Boldface type is the preferred rendering of strong text. The difference between these two tags is illustrated in Figure 8.9.

OTHER OPTIONS FOR EMPHASIZING

Later in this chapter, you'll also see how the italic (<I>) and bold (<B>) tags are used to generate the same effects as emphasis and strong emphasis. So, why should anyone use the logical styles? They're longer and aren't as clear about their final appearance.

Remember that part of the purpose of HTML is to show structure and meaning within a document. This is the role the two emphasis tags play. Using the physical bold or italics tag would achieve the same visual effect, but using the logical tags also shows the intention and is stable across all platforms–not just the ones that support a variety of typestyles.

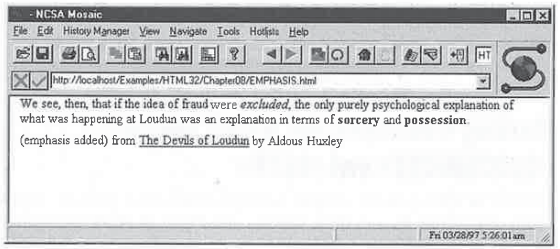


FIGURE 8.9.
Emphasized text is usually displayed with italics to make it stand out from surrounding text, and strong text uses a bold presentation.

Address Information: <ADDRESS>

The <ADDRESS> tag is used to mark contact information for the current document, whether it's an e-mail address or complete mailing address and phone number. It behaves much like the paragraph tag, forcing the text within its confines to be separated from surrounding material by additional line spacing. The syntax is

```
<ADDRESS>...ContactInformation...</ADDRESS>
```

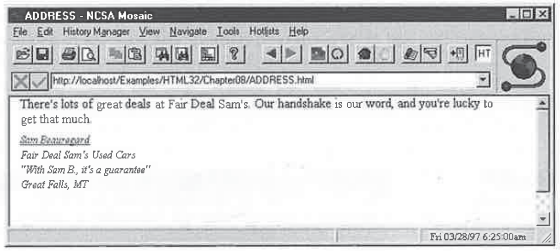
where *ContactInformation* is the address information, along with any paragraph-level formatting such as line breaks. It is typically displayed as italic body text.

You can nest other items within the address tag, including hyperlink information. For example, one option for displaying e-mail contact information is

```
<ADDRESS><A href="mailto:sam@fairdeals.com">Sam Beauregard</A><BR>
Fair Deal Sam's Used Cars<BR>
"With Sam B., it's a guarantee"<BR>
Great Falls, MT
</ADDRESS>
```

This would create a hyperlink to connect to the user's e-mail software while maintaining the formatting for address information. (See Figure 8.10.) For more ideas on working with hyperlinks and mail-to destinations, see Chapter 11, "Linking Documents and Images."

FIGURE 8.10.
This text is formatted with the <ADDRESS> tag, and it also includes a nested tag for defining a hyperlink.



Marking Quotations and Noting Sources: <BLOCKQUOTE> and <CITE>

Sometimes when you are composing pages, you'll want to insert a quotation out of another work, or attribute facts to their sources. (See Figure 8.11.) Once again, HTML provides tags that identify how the text is being used, and leaves the details of its physical appearance to the browser.

For shorter quotations, it's certainly fine to use quotation marks and leave the text inline with the rest of the content. However, if the amount of text to quote exceeds more than a couple of sentences, it is easier for the reader to take note of the quote if it is separated from the rest of the text. This is when the <BLOCKQUOTE> element is used. The syntax is

```
<BLOCKQUOTE>...Text...</BLOCKQUOTE>
```

where *Text* is the quotation that should be separated from the rest of the surrounding material. This creates a separate paragraph for the text and, in most browsers, indents the entire paragraph from the left. (See Figure 8.11.) Some browsers also include a slight indent from the right, or set the text in italics.

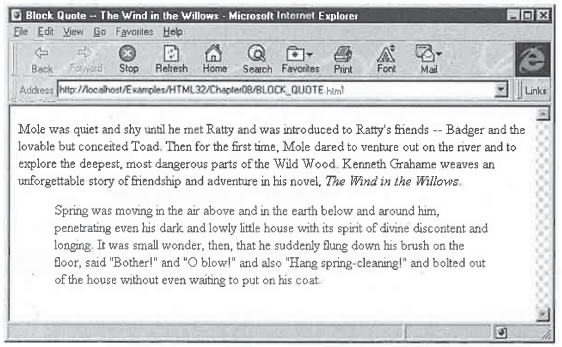
The block quotation allows other formatting within its borders, including paragraphs, line breaks, and headings. Individual browsers vary in their handling of special formatting. Most will indent special formatting such as headings along with the rest of the block quotation.

Keep in mind that not all browsers handle the block quotation the same way, especially in regards to the type style (regular or italics). If you include formatting such as italics within a block quote, and the browser normally displays block quote in italics, then your added emphasis is lost to the reader. In some cases, the browser might change the typestyle of the rest of the block quotation.

TIP

For safety and consistency, try to avoid the use of other formatting within a block quotation.

FIGURE 8.11.
A block quotation is separated from surrounding text by creating a new paragraph and indenting it from the left. The citation to the source of the quote is marked in italics.



The citation tag, <CITE>, is used to identify sources of information outside the current document. It's used most often in research and professional papers, although you might find use in other areas such as book reviews and Frequently Asked Questions. The syntax is

```
<CITE>Source</CITE>
```

where *Source* is the name of the citation. The usual rendering for <CITE> is italics.

Defining a Term: <DFN>

When you think about terms and definitions, a dictionary or glossary is the image that usually comes to mind. HTML offers one way of presenting this type of information through different types of lists, which are covered in Chapter 9, "Using Lists to Organize Information." Although HTML lists are the usual method used for presenting terms and definitions, there is also another alternative.

The other option for showing a definition term is using the <DFN> tag. Its syntax is

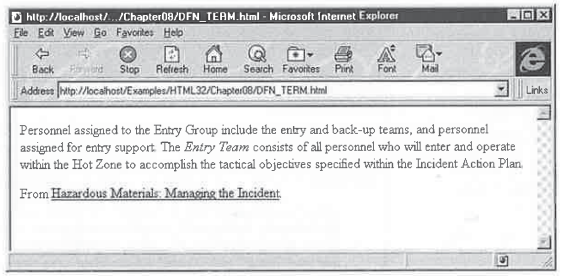
```
<DFN>Term</DFN>
```

where *Term* is the word that is defined. It's used to identify words within a body of text that are defined in the same sentence. (See Figure 8.12.) The typical rendering for a definition term is italics.

Indicating Program Code: <CODE>

You've seen examples of syntax and usage of HTML code throughout this chapter, and you will continue to see it throughout the rest of the book. Sams.net uses a special monospaced typestyle to show these examples so that they stand out from the rest of the text.

FIGURE 8.12.
The italicized words "Entry Team" are the definition term, which is defined within the same sentence.



HTML has a similar feature in the <CODE> tag that it uses to format programming or other similar code lines. The syntax is

```
<CODE>...CodeLine...</CODE>
```

where *CodeLine* is the line or lines of code samples. If multiple lines of code are included, the line-break tag
 is also needed.

It's important to note that although the typical display of this element uses monospace text similar to the PRE element, <CODE> still requires the
 or <P> tags to force new lines. Except for the appearance of the text, anything within the <CODE> container behaves the same as other HTML text.

The <VAR> tag is often used in conjunction with the <CODE> tag to help explain the code. Its syntax is

```
<VAR>Variable</VAR>
```

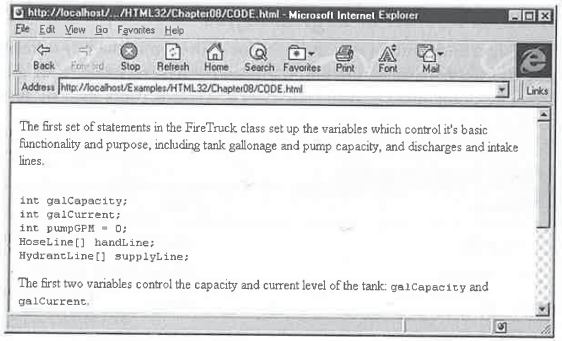
where *Variable* is the name of a variable being described, much like <DFN> works for definition terms. The typical representation is italics, which is also the same as <DFN>.

<VAR> is similar to <DFN> in use; it explains a variable or argument of a piece of code within the context of a normal sentence or paragraph.

Figure 8.13 includes an example of using <CODE> and <VAR> together. After the code listing, there is a brief explanation of two of the variables.

Two more companions to the code elements are used in similar situations: <SAMP> and <KBD>.

FIGURE 8.13.
The lines of Java code listed in this display are treated with a monospaced format to separate them from the explanatory text.



Sample Output and User Typing: <SAMP> and <KBD>

These two tags identify things the computer would say to the user and examples of what the user should say to the computer. Like <CODE> and <VAR>, they are hangers-on from the old days of the Internet, when just about everyone worked from a text-based UNIX system. The syntax is

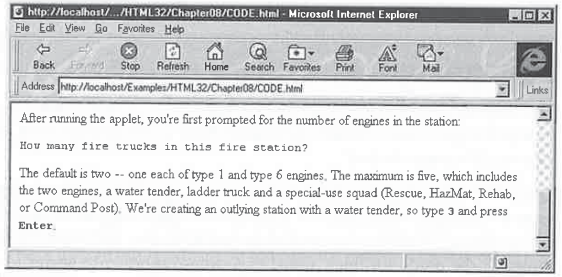
```
<SAMP>SampleOutput</SAMP>
```

where *SampleOutput* is an example of a message the user might receive from the computer, and

```
<KBD>KeyboardInput</KBD>
```

where *KeyboardInput* is an example of something the user might type on the keyboard. The typical representation of sample output is monospaced text, while keyboard input is usually displayed with bold monospaced text. (See Figure 8.14.)

FIGURE 8.14.
The simulated output from a computer is presented in monospaced type on a line by itself. The response requested from the user is presented in the text as monospaced bold type.



A Quick Review of Logical Styles

You might have noticed that there is a lot of overlap in the way different HTML logical elements are represented within a browser. Although there is no set standard for these items, there are some commonly accepted conventions. Table 8.1 lists the logical styles and their typical rendering.

Table 8.1. Logical element styles.

Element	*Style*
<ADDRESS>	*Italics*
<BLOCKQUOTE>	Normal with indent
<CITE>	*Italics*
<CODE>	Monospace
<DFN>	*Italics*
<EM>	*Italics*
<KBD>	**Bold Monospace**
<SAMP>	Monospace
<STRONG>	**Bold**
<VAR>	*Italics*

The various browsers may interpret these items in a variety of ways, but remember that this is okay. Part of the purpose of HTML is to make documents that are platform independent. The end user really shouldn't care which style is used for which item, as long as it's consistent.

When you use the logical styles, the browser can decide how to display a certain type of text based on the capabilities of the platform, reducing your concerns about where your document can be viewed.

Formatting Text with Physical Styles

The first half of this chapter is devoted to exploring the logical styles; now it's time to look at how to force text into a certain appearance. After all the reasons given for using logical styles, why would anyone do this?

Quite simply, the logical styles don't cover all the possibilities that designers and page authors seem to thrive on. Using physical styles allows a page designer to take control over many details of the page's appearance, including type style, size, and color. For users with compatible browsers, the result is a finely tuned page whose appearance begins to approach the quality found on the pages of many popular magazines.

Basic Text Formatting Styles: <I>, <B>, <U>

The first three elements are standards in virtually all text-processing environments. They set the basic appearance of the text (as shown in Figure 8.15), and they are the basis for many of the logical styles.

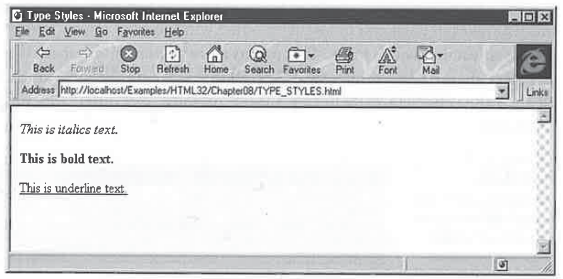


FIGURE 8.15.
The three basic physical styles are italics, bold, and underline.

The syntax for each of the elements is

```
<I>...Text...</I>
<B>...Text...</B>
<U>...Text...</U>
```

where <I> stands for italics, <B> stands for bold, and <U> stands for underline. In addition to using the tags individually, you can also nest them for a combined effect, like this:

```
This is <B>bold text with <I>bold and italics text</I></B>.
```

Use of the underline tag can confuse users because it is also used to mark hyperlinks to other documents. Use it sparingly, and try to limit its use to documents that don't also contain text hyperlinks.

TIP

From the days of typesetting and proofreader marks, underlined text was used to indicate items that should be set in italics. This includes book and magazine titles.

Strike-Through Text: <STRIKE>

Strike-through text is used more often in traditional word processing than in HTML pages. It indicates text that has been deleted but is still left on the page for review. This way other readers can review the changes and know how the revision of the document has progressed. The syntax is

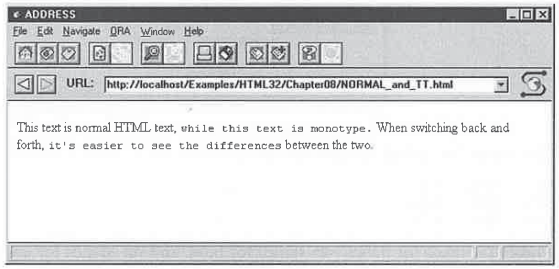
```
<STRIKE>...Text...</STRIKE>
```

The HTML committee of the World Wide Web Consortium is considering replacing <STRIKE> with the shorter version <S>. This is only an idea; you can expect <STRIKE> to remain valid for at least another year or two.

Teletype or Monospaced Text: <TT>

Several logical text styles take advantage of monospaced text. Simply stated, monospace text means that each character takes up the same width on the screen. Normally, graphical browsers use adjustable-width fonts. This means an *m* takes up more space than an *i*, which takes up less space than a *j*. In a monospace font, all three take up the same amount of space along with all the other characters, including spaces, periods, and other punctuation marks. (See Figure 8.16.)

FIGURE 8.16.
The difference between normal and monospace font isn't very evident until they are viewed adjacent to each other.



The syntax is

```
<TT>...Text...<TT>
```

where *Text* is the font to display in monospace font. The <TT> designation in the tag hearkens back to the days of teletype machines. Essentially, these were typewriters hooked up to networks that delivered news and bulletins. As was standard for typewriters at the time, they used a monospace type style.

Superscripts and Subscripts: <SUP> and <SUB>

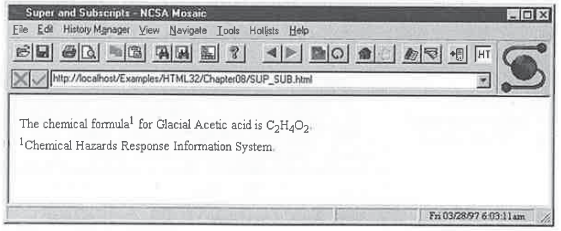
Next in the list of physical styles are the two tags for creating superscripts and subscripts. A superscript is text set slightly higher than the base text, and a subscript is set slightly lower. Both typically use a smaller typeface. The syntax for both is

```
<SUP>Text</SUP>
<SUB>Text</SUB>
```

In normal usage, a superscript or subscript is used for only a character or two. The following snippet uses this feature to indicate an endnote source (superscript) and notation for a chemical formula (subscript). (See Figure 8.17.)


```
The chemical formula<SUP>1</SUP> for Glacial Acetic acid
is C<SUB>2</SUB>H<SUB>4</SUB>O<SUB>2</SUB>.
<BR><SUP>1</SUP>Chemical Hazards Response Information System.
```

FIGURE 8.17.
Superscripts and subscripts are typically used for small pieces of text, such as marking endnotes and other special notation.



Like the other elements, you can nest these tags for additional usefulness. For example, you can make a reference to an endnote superscript also serve as a link to the actual endnote, like this:

```
The chemical formula<A HREF="#1"><SUP>1</SUP></A> for Glacial Acetic acid
is C<SUB>2</SUB>H<SUB>4</SUB>O<SUB>2</SUB>.
<BR><A NAME="1"><SUP>1</SUP></A>Chemical Hazards Response Information System.
```

Clicking on the superscript for the endnote moves the user to that location in the document. For more information on working with anchors and hyperlinks, see Chapter 11.

Working with the Letters: <FONT>

With HTML 3.2, it's now possible to override the default size and color of text selections on a page. This is accomplished using the FONT element and its two attributes—size and color.

Some browsers also support a face attribute, which is used to specify one or more typefaces to use for the selection. It is not part of the HTML 3.2 specification, although it is currently supported by the two leading browsers. To specify the Garamond typeface as the first choice and Arial as the second, the tag would look like this:

```
<FONT face="Garamond,Arial">...text...</FONT>
```

If neither typeface is found on the user's browser, the default typeface for that particular application is used. If you choose to use the attribute, be sure that your page also displays correctly if the user's browser doesn't support it.

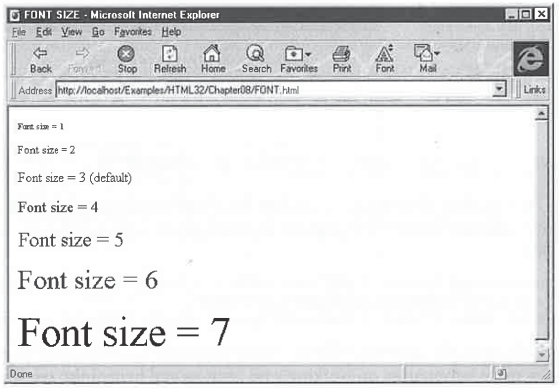
Setting the Size: size and <BASEFONT>

The first attribute of <FONT> is size. This sets the size of the contents of the tag to a specific or relative size. The syntax is

```
<FONT size=number>...Text...</FONT>
```

where *number* is the desired size expressed as an integer from 1 to 7, or as a relative value from –6 to +6. If a relative value is used, it is added to the current setting for <BASEFONT>. (This is covered further later in this section.) Unlike the headings, where <H1> represents the largest heading and <H6> is the smallest, the SIZE attribute uses 1 for the smallest type size and 7 for the largest. (See Figure 8.18.)

FIGURE 8.18.
This is how different text sizes are represented using the FONT *element.*



UNDERSTANDING HTML TYPE SIZES

For people used to dealing with type sizes represented in points, the range of sizes for HTML text doesn't appear practical. After all, a line of 1-point type creates fine print not even a lawyer could understand.

HTML type sizes do correspond to physical sizes, but the actual physical size varies between browsers and platforms. So, don't try to think of size 1 being equal to 6-point text. Instead, just think of size 1 as "really small," size 7 as "really big," and 2 through 6 as a range of "fairly small" to "fairly big."

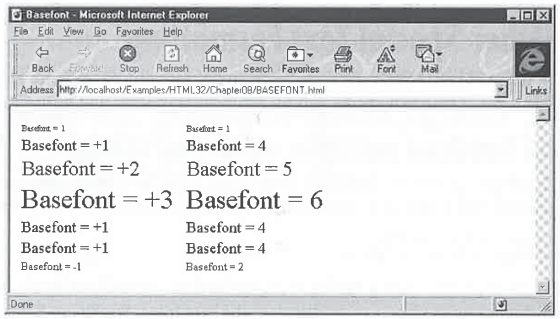
As a companion to the SIZE attribute of <FONT>, the BASEFONT element provides a way to set the base size of the body text. Its syntax is

```
<BASEFONT [size=number]>
```

where *number* is the size of the base font, expressed as an integer from 1 to 7. If omitted, it defaults to 3. It sets the base for all text following it and does not have a closing tag.

In addition to specifying an absolute text size, you also can use a relative number for <BASEFONT>. The syntax is the same, but a plus or minus sign is placed in front of the number. This method uses the default base size (3) and adds the value of the SIZE attribute. (See Figure 8.19.)

FIGURE 8.19.
No matter what the size of the previous font is, a relative value for the base font size is always added to the default size.



Using relative sizes, the range of values for the base SIZE attribute is –2 to +4. For both <BASEFONT> and <FONT>, relative values that result in a size less than 1 or more than 7 default to the minimum and maximum values for physical sizes.

A Different Shade of Text: COLOR

You can control the color of a block of text using the COLOR attribute. The syntax is

```
<FONT color=name¦RGBValue>...Text...</FONT>
```

where the value of color is a *name* of one of the 16 standard colors, or an RGB hexadecimal triplet that specifies the mix of colors to use. Remember that default text colors for the entire document are set with attributes of the <BODY> tag, which was covered in the previous chapter.

NOTE

You can define a color in two ways. The first is the RGB hexadecimal triplet, which is very specific but not very friendly.

Or, you can use the name of the color. It's much more friendly, but you can't use just any name. Personally, I really liked the color of my friend John's car. It was taupe. Unfortunately, I can't type <FONT color="taupe"> and expect the browser to understand.

If you want to use color names, you'll need to stick to one of the 16 standard values–black, green, silver, lime, gray, olive, white, yellow, maroon, navy, red, blue, purple, teal, fuchsia, and aqua. Some browsers use other names. To be safe, use one of these names, or get out the old scientific calculator and start figuring 32 percent blue as a hexadecimal number.

Be careful when changing text colors, especially when using a non-default background such as an image or other color. Some colors don't work well together, such as green text on red, or red text on purple. The resulting effect will do more to drive your readers away than to illustrate your point.

Other Special Text Formatting

In the world of HTML text formatting, some tags don't fit well into either logical or physical text styles. These include the tags for relative size changes and horizontal rules.

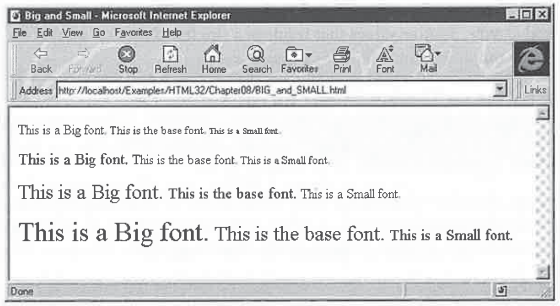
All Text Great and Little: <BIG> and <SMALL>

An additional two tags in HTML control size. They are a hybrid between the logical styles and physical styles; they don't specify a size, but they don't indicate usage either. The syntax is

```
<BIG>...Text...</BIG>
<SMALL>...Text...</SMALL>
```

where the text is set to a smaller or larger size than currently set by the base font. Typical rendering is one size larger or smaller than the base font. (See Figure 8.20.)

FIGURE 8.20.
Using <BIG> or <SMALL> results in the enclosed text being set one size larger or smaller than the base font, respectively.



If the base font is set to the smallest size (1), <SMALL> defaults to a size of 3. The same occurs when the base is set to the largest size (7), when <BIG> also defaults to 3.

Drawing a Line on the Page: <HR>

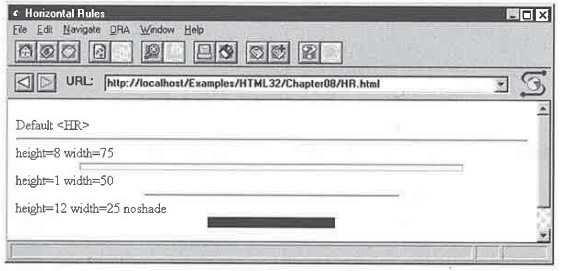
Horizontal lines are an easy-to-use element for dividing a page into logical sections. They signal the reader to be alert for a change in subject or style, or they can separate figures and captions from body text. Depending on your design needs, <HR> also includes several attributes to fine-tune its appearance. The syntax is

```
<HR attributes>
```


where *attributes* is one or more of four controls controlling height, width, alignment, and shading.

The first two attributes—height and width—set the basic size of the line. (See Figure 8.21.) The height of a horizontal rule is set in pixels, such as height=8. The width is set either in pixels or as a percentage of the browser window width. If you're using the percentage value, include a percentage sign (%) after the value. The typical defaults for height and width are 3 pixels high and 100 percent of page width.

FIGURE 8.21.
The first line on this page uses the browser size defaults. The subsequent lines adjust the height and width values for varying effects.



The next attribute is align, which controls horizontal placement. The values for align are left, right, or center. Omitting this attribute usually results in left alignment of the line.

The last attribute is noshade. Normally, a browser displays a horizontal rule in some form of three-dimensional shading. This varies from browser to browser; some show it as depressed, some raised, some as an outline. Using noshade forces the horizontal rule to appear as a bold line. Again, this may vary among browsers. Some use black for the rule, and others use shades of gray.

Summary

This chapter is designed to give you a head start in formatting text on your Web pages so that you can give your message the maximum impact. With the exception of a couple of tags for hyperlinks and images, this includes all of the most-used tags in HTML.

First and foremost are the tags to break the text into paragraphs and lines. Without them, every HTML document would just be one long, hard-to-read document. They give the basic structure to the page for the reader.

After breaking the document into paragraphs, HTML 3.2 offers two basic sets of choices to organize and format the text on your Web pages. First, you can use the logical styles. These format the text based on its purpose and usage, such as citations and addresses. They are typically supported across platforms and browsers, although the actual rendering might be atypical.

The physical styles don't necessarily give clues to the use or purpose of the text, but they do allow closer control over text appearance. You can define font size and color, along with using monospaced fonts and applying bold, italics, and underline styles.

A multitude of other HTML tags also control text appearance and formatting, but they are unique enough to deserve treatment by themselves. These include the list tags (Chapter 9, "Using Lists to Organize Information"), hyperlinks and anchors (Chapter 10, "Creating Tables for Data and Page Layout"), and forms (Chapter 15, "Building and Using HTML Forms").

CHAPTER 9

Using Lists to Organize Information

by Rick Darnell

IN THIS CHAPTER

Lists are everywhere. On the refrigerator for groceries, on a scratch pad for to-do lists, in the front of books as a table of contents, in lists of directions for accomplishing tasks. Lists are one of the most natural ways we organize our information.

HTML has a special set of tags just for the purpose of displaying lists, and HTML 3.2 has added additional attributes to some of these tags to give you greater control over their appearance.

At the most basic level, lists are divided into two categories:

- **Ordered Lists**. These lists are typically used to indicate a sequence of events or priorities. They're also used to specifically identify sections and relationships when creating outlines.
- **Unordered Lists**. An unordered list is typically used to display a group of items that are somehow related, but necessarily in a hierarchical fashion. Three special HTML subsets of unordered lists are illustrated later in this chapter—definitions, directories, and menus.

The next sections look at each type of list and the ways to customize them to your own specific circumstance.

Ordered (or Numbered) Lists: <OL>

A list is defined by its opening and closing tags. For ordered lists, they are <OL> and </OL>. However, two tags do not a list make. If you're going to have a list, you'll need something to put in it. Look at Listing 9.1, which is a list encased with the ordered list tags.

Listing 9.1. A simple ordered list enclosed with <OL> tags.

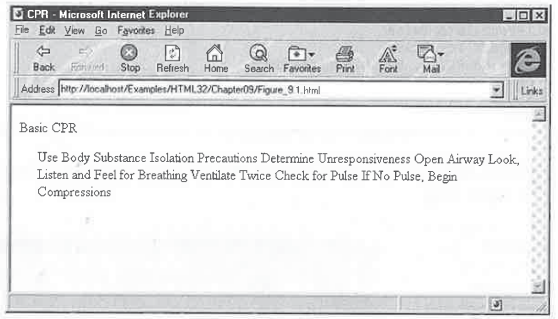
```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Basic CPR</TITLE>
</HEAD>
<BODY>
Basic CPR
<OL>
Use Body Substance Isolation Precautions
Determine Unresponsiveness
Open Airway
Look, Listen and Feel for Breathing
Ventilate Twice
Check for Pulse
If No Pulse, Begin Compressions
</OR>
</BODY>
</HTML>
```


NOTE

To save a little ink, examples and listings in this chapter won't explicitly include the basic structural codes (<HTML>, <HEAD>, and <BODY>). They're still assumed to be in place, and should be included if you enter the examples by hand.

Viewed on a browser, you can see where the text was indented in preparation for a list, but no actual numbering took place. (See Figure 9.1.)

FIGURE 9.1.
A list of information enclosed with list tags doesn't result in an HTML list. Additional tags are needed to complete the formatting.



Why did this happen? Remember that HTML doesn't recognize line breaks unless it's told explicitly where they occur with the <P> or
 tags. The same principle applies to lists, although a different tag is used to tell the browser where each specific item begins. This is accomplished with the list item tag—<LI>.

With the <OL> and <LI> tags in hand, the basic syntax for an ordered list is as follows:

```
<OL>
<LI>ListItem1
<LI>ListItem2
...
<LI>ListItemN
</OL>
```

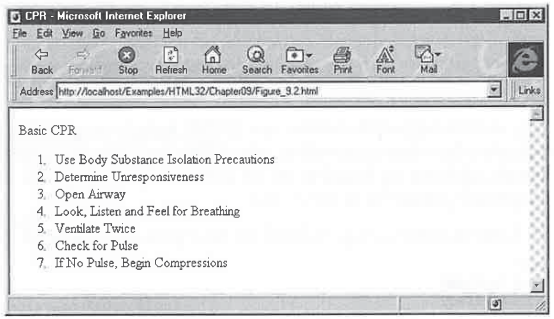
ListItem is each separate item in the list. See Listing 9.2 for an example of the usage of these tags.

Listing 9.2. The same list shown in Listing 9.1, but now it has the beginning of each separate item marked with a list item tag.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Basic CPR</TITLE>
</HEAD>
<BODY>
Basic CPR
<OL>
<LI>Use Body Substance Isolation Precautions
<LI>Determine Unresponsiveness
<LI>Open Airway
<LI>Look, Listen and Feel for Breathing
<LI>Ventilate Twice
<LI>Check for Pulse
<LI>If No Pulse, Begin Compressions
</OR>
</BODY>
</HTML>
```

For each item identified with <LI>, the browser starts a new line, indents, and adds a number. (See Figure 9.2.)

FIGURE 9.2.
After each item is marked at the beginning with a list item tag, the browser completes the formatting with indenting and automatic numbering.



TIP

Although some HTML editors include it, a closing </LI> tag is not required for each list item. However, you can include it if you want without adverse effect.

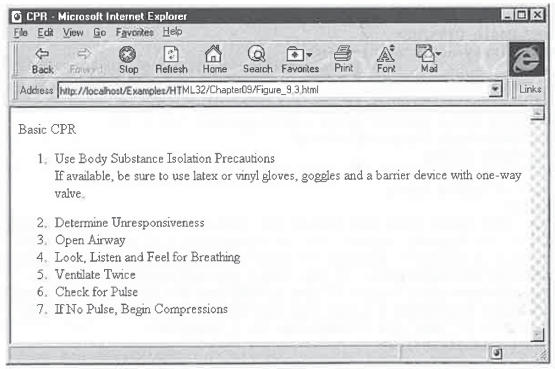
The line break tags—<P> and
—are also allowed within the body of a list to further control its appearance and formatting (as shown in Listing 9.3).

**Listing 9.3. You can organize your list into headings and subtext using the
 and <P> tags.**

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Basic CPR</TITLE>
</HEAD>
<BODY>
Basic CPR
<OL>
<LI>Use Body Substance Isolation Precautions<BR>
If available, be sure to use latex or vinyl gloves,
goggles and a barrier device with one-way valve.<P>
<LI>Determine Unresponsiveness
<LI>Open Airway
<LI>Look, Listen and Feel for Breathing
<LI>Ventilate Twice
<LI>Check for Pulse
<LI>If No Pulse, Begin Compressions
</OR>
</BODY>
</HTML>
```

When viewed on a browser (as shown in Figure 9.3), the tags affecting line breaks don't affect the line numbering. When the list begins with the <OL> tag, only an <LI> tag will cause the browser to start a new line with the next number. You can force other lines to begin with the line break tags, but because it doesn't begin a new list item, the browser treats it like a continuation of the current item and doesn't add a number for the next item in the sequence.

FIGURE 9.3.
Additional text formatting and page breaks are allowed within a list without affecting the basic list numbering or style.



TIP

You can also use any other basic text formatting tags for list items, including logical styles such as <ADDRESS>, or physical styles such as <I>.

For cleaner code and later revision, make sure any other formatting tags opened within a list item are closed before the next item begins.

In addition to the formatting you can apply to individual list items, there are two other ways to customize an ordered list. This includes choosing a new beginning for list numbering (`start`) and the numbering characters used (`type`). These two attributes are covered in the following sections.

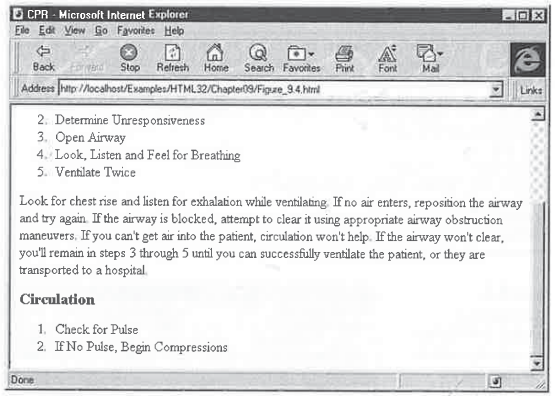
Where to `start` a List

Sometimes it's necessary to interrupt a list for some explanatory text or other material where indenting isn't needed. Look at Listing 9.4 and its corresponding Figure 9.4.

Listing 9.4. The second ordered list is a continuation of the first. It's interrupted by some explanatory text that you did not want indented.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Basic CPR</TITLE>
</HEAD>
<BODY>
<H2>Basic CPR</H2>
<H3>Airway and Breathing</H3>
<OL>
<LI>Use Body Substance Isolation Precautions<BR>
If available, be sure to use latex or vinyl gloves,
goggles and a barrier device with one-way valve.<P>
<LI>Determine Unresponsiveness
<LI>Open Airway
<LI>Look, Listen and Feel for Breathing
<LI>Ventilate Twice
</OL>
Look for chest rise and listen for exhalation while ventilating.
If no air enters, reposition the airway and try again. If the
airway is blocked, attempt to clear it using appropriate airway
obstruction maneuvers. If you can't get air into the patient,
circulation won't help. If the airway won't clear, you'll remain
in steps 3 through 5 until you can successfully ventilate the
patient, or they are transported to a hospital.
<H3>Circulation</H3>
<OL>
<LI>Check for Pulse
<LI>If No Pulse, Begin Compressions
</OR>
</BODY>
</HTML>
```


FIGURE 9.4.
List numbering is not consecutive between individual lists. A new list starts the numbering process over from the beginning.



Two sets of <OL> tags are used—one for the first half of CPR (airway and breathing), and another for the second half (circulation). Because each portion of the list is contained within its own pair of ordered list tags, the browser treats each as a separate list and begins numbering from 1 for the second list. The browser doesn't know it's really a continuation of the first list.

To work around this problem, use the `start` attribute for the <OL> tag to start numbering anywhere you want. (See Listing 9.5 and Figure 9.5.)

Listing 9.5. The `start` attribute can be used to override default list numbering when a list is broken into multiple parts.

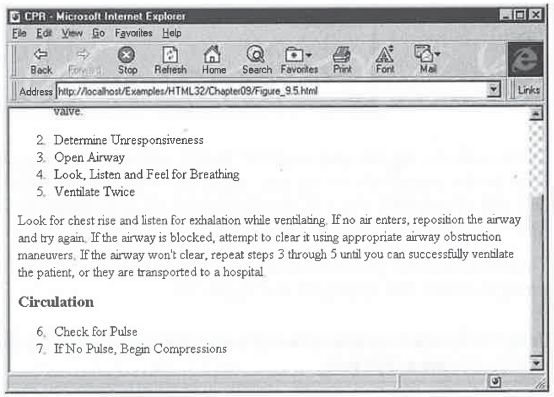
```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Basic CPR</TITLE>
</HEAD>
<BODY>
<H2>Basic CPR</H2>
<H3>Airway and Breathing</H3>
<OL>
<LI>Use Body Substance Isolation Precautions<BR>
If available, be sure to use latex or vinyl gloves,
goggles and a barrier device with one-way valve.<P>
<LI>Determine Unresponsiveness
<LI>Open Airway
<LI>Look, Listen and Feel for Breathing
<LI>Ventilate Twice
</OL>
Look for chest rise and listen for exhalation while ventilating.
If no air enters, reposition the airway and try again. If the
```

continues

Listing 9.5. continued

```
airway is blocked, attempt to clear it using appropriate airway
obstruction maneuvers. If you can't get air into the patient,
circulation won't help. If the airway won't clear, you'll remain
in steps 3 through 5 until you can successfully ventilate the
patient, or they are transported to a hospital.
<H3>Circulation</H3>
<OL start=6>
<LI>Check for Pulse
<LI>If No Pulse, Begin Compressions
</OR>
</BODY>
</HTML>
```

FIGURE 9.5.
Using the `start` *attribute with the* `<OL>` *tag makes it possible to create lists that are interrupted by blocks of text or other material and then resumed with appropriate numbering.*



The syntax for `start` is

```
<OL start=number>
```

where *number* is any integer from 2147483647 to –2147483648. Be sure not to use commas in your numbers, because commas cause the numbers to be misinterpreted by the browser.

TIP

If numbers outside this range are used, the numbers "roll over" to the beginning. For example, if a list begins with 2147483647, the next number displayed in line is -2147483648. If your lists are hampered by a number system that extends to only approximately two billion, you probably have way too much time on your hands.

What type of List Is Needed

In the early days of HTML, there was only one type of ordered list—with numbers beginning at 1 and ending wherever the list stopped. Then, as people began stretching its uses into things such as online books, the desire arose to use something other than numbers. The `type` attribute was developed in response.

The syntax of `type` is

```
<OL type=numberingSystem>
```

where *numberingSystem* is one of five characters—*1*, *A*, *a*, *I*, or *i*. Examples of the five numbering units are shown in Table 9.1.

Table 9.1. Values and styles for the `type` attribute.

Value	*Style*	*Example*
`1`	Arabic	1,2,3,...
`A`	Uppercase alpha	A,B,C,...
`a`	Lowercase alpha	a,b,c,...
`I`	Uppercase Roman	I,II,III,...
`i`	Lowercase Roman	i,ii,iii,...

The ability to use something other than numbers leads to a useful feature of ordered lists: nested lists. This is a list within a list, and can extend several levels. To create a nested list, include a new set of ordered list tags within the current list tags (as shown in Listing 9.6). The browser begins a new list for the new tags, while remembering where the parent list left off after the nested list ends. (See Figure 9.6.)

Listing 9.6. A nested list is created by adding a new set of ordered list tags within existing list tags.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Patient Assessment</TITLE>
</HEAD>
<BODY>
<!--BEGIN MAIN LIST-->
<OL type=A>

<LI>Safety Considerations
<OL type=1>
  <LI>Body substance isolation
```

continues

Listing 9.6. continued

```
  <LI>Scene safety
  <LI>Initial size-up
</OL>

<LI>Initial Patient Assessment
<OL type=1>
  <LI>General Impression
  <LI>Unresponsiveness
  <OL type=i>
    <LI>Alert to person, place and time
    <LI>Verbal response to audible stimuli
    <LI>Pain evokes verbal or physical response
    <LI>Unresponsive to all stimuli
  </OL>
</OL>

<LI>Patient Critical Needs
<OL type=1>
  <LI>Airway
  <LI>Breathing
  <OL type=i>
    <LI>Use oxygen if indicated
    <LI>Consider use of assisting with bag valve mask
  </OL>
  <LI>Circulation
  <LI>Bleeding
</OL>

<!--END MAIN LIST-->
</OL>
</BODY>
</HTML>
```

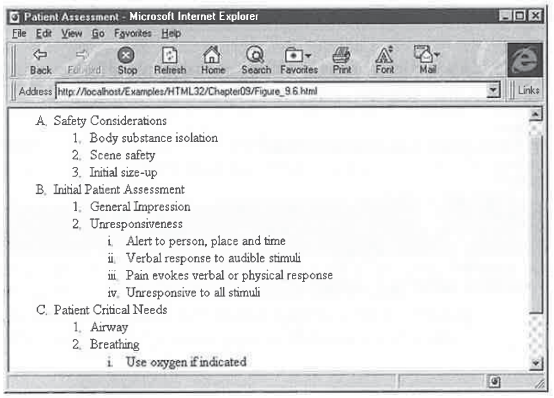


FIGURE 9.6.
Using the `type` *attribute in conjunction with nested lists results in text formatted into outline form.*

One of the main problems with nesting lists is the confusion they can generate. It's easy to lose track of which list is which and what's subordinate to what after the second or third embedded set of items. Remember that extra leading and trailing spaces are stripped from HTML along with extra carriage returns. Feel free to use extra returns and indenting in conjunction with comment tags to make your page's source code easier to read and edit.

TIP

If you use `type` in conjunction with `start`, be sure to use an integer with `start` regardless of the `type` specified.

For example, if you're including an ordered list that uses capital letters and begins at the letter *D*, the opening tag should be `<OL type=A start=4>`. The browser will translate the starting position into the appropriate letter.

Now that you've worked through the various types and variations of an ordered list, it's time to take a look at its half-sister—the unordered list.

Unordered Lists: <UL>

Unordered lists are typically used to represent a set of items that are somehow related to one another, but don't necessarily need to follow a specific order. The syntax is similar to that of the ordered list:

```
<UL>
<LI>ListItem1
<LI>ListItem2
...
<LI>ListItemN
</UL>
```

`ListItem` is each separate item in the list. See Listing 9.7 for an example.

Listing 9.7. Use of the unordered list is the same as the ordered list, with the substitution of <UL> for <OL>.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Jump Kit Inventory</TITLE>
</HEAD>
<BODY>
EMS Jump Kit Contents
<UL>
<LI>Rescue Scissors and Penlight
<LI>Stethascope and Sphygmanometer
<LI>Oxygen Bottle
<LI>Non-Rebreather Mask and Nasal Cannula
```

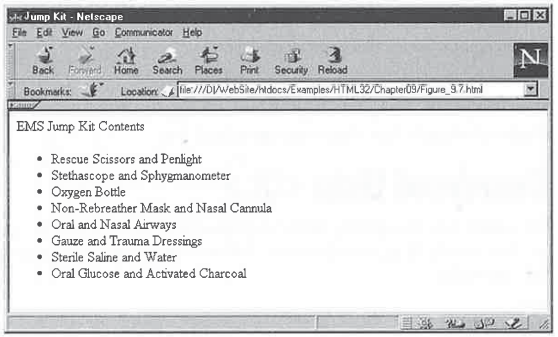
continues

Listing 9.7. continued

```
<LI>Oral and Nasal Airways
<LI>Gauze and Trauma Dressings
<LI>Sterile Saline and Water
<LI>Oral Glucose and Activated Charcoal
</UL>
</BODY>
</HTML>
```

As you can see in Figure 9.7, the unordered list is typically represented the same as an ordered list, except bullets are used instead of numbers.

FIGURE 9.7.
An unordered list is displayed much like an ordered list, but it includes bullets in place of numbers.



The actual appearance of the bullet varies from browser to browser. Internet Explorer uses small text bullets, and NCSA Mosaic uses a graphical bullet.

Numbering is obviously not an issue for an unordered list. However, HTML 3.2 offers some additional choices for the default bullet appearance with the `type` attribute.

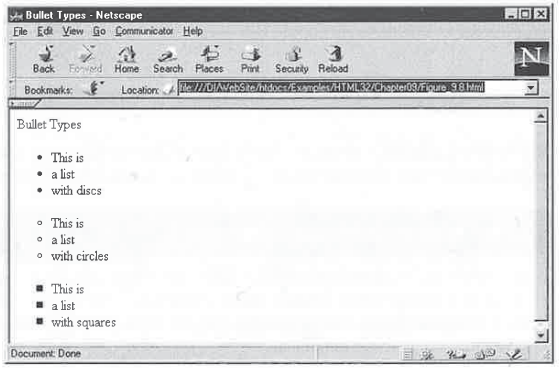
What `type` of Bullet Do You Want?

Three basic types of bullets are supported by HTML 3.2, although not all browsers support all three. If a browser doesn't recognize the attribute, the default bullet representation is used. The syntax for `type` is

```
<UL type=bulletType>
```

where *`bulletType`* is one of three values—circle, square, or disc. Their representations are displayed in Figure 9.8.

FIGURE 9.8.
Three styles of bullets are supported by HTML 3.2—circles, squares, and discs.



Also like the ordered list, an unordered list can contain other lists within a list. This helps to show relation among items, even though there isn't an underlying hierarchy. It is implemented in the same manner, with another set of list tags replacing a list item. (See Listing 9.8.) The `type` attribute helps to graphically differentiate the relationships.

Listing 9.8. Using different types of bullets helps to graphically separate sublists.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Jump Kit Inventory</TITLE>
</HEAD>
<BODY>
EMS Jump Kit Contents
<UL>
<LI>Rescue Scissors and Penlight
<LI>Stethoscope and Sphygmanometer

<LI>Airway and Breathing Support
<UL type=square>
<LI>Oxygen Bottle
<LI>Non-Rebreather Mask and Nasal Cannula
<LI>Oral and Nasal Airways
</UL>

<LI>Gauze and Trauma Dressings
<LI>Sterile Saline and Water

<LI>Medications
<UL type=square>
<LI>Oral Glucose
<LI>Activated Charcoal
```

continues

Listing 9.8. continued

```
</UL>
</UL>
</BODY>
</HTML>
```

Notice that the list item preceding the embedded list also serves as a heading for the list. This is an aid to the reader because it identifies how the sublist relates to the main list.

A LITTLE MORE ABOUT LISTS AND ATTRIBUTES

One additional attribute is allowed for both types of lists, although its implementation in browsers is sparse at best. This is `compact`, which signals the client to try to display the list in the most space-efficient manner possible. Given the other possible variables within a list, most browsers ignore this attribute.

List items are also available for modification beyond their containing list tag. For ordered lists, the optional attributes for `<LI>` are `type` and `value`. The possible values are the same as the corresponding attributes in the `<OL>` tag, with `value` working the same as `start`.

For unordered lists, the lone optional attribute is `type`, with the possible values matching its `<UL>` counterpart.

The actual implementation of `<LI>` attributes varies from browser to browser, so be aware of differences. In Netscape Navigator, for example, setting an `<LI>` attribute in the middle of a list also affects the following list items.

A Definition or Glossary List: <DL>

The last specialty list tag is `<DL>`, which stands for definition list. This tag is used to create a glossary-style listing, which is handy for items such as dictionary listings and Frequently Asked Questions pages.

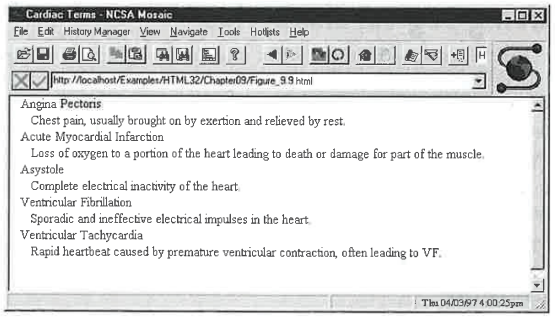
It is used similarly to the unordered list tag, except that it doesn't use the `<LI>` tag to mark its various entries. This is because a definition list requires two items for every entry—a term and its definition. This is accomplished with the corresponding `<DT>` and `<DD>` tags. The syntax is

```
<DL>
<DT>Term1<DD>Definition1
<DT>Term2<DD>Definition2
...
<DT>TermN<DD>DefinitionN
</DL>
```

where *`Term`* is the word requiring a definition, and *`Definition`* is the block of text that serves the purpose. An alternate form of including the tags on the page places each `<DT>` and `<DD>` tag on separate lines, although the first method is a bit clearer in purpose. The browser presents the content in the same fashion, regardless of which way you choose.

Its implementation can vary a bit from browser to browser, but a common implementation is displayed in Figure 9.9.

FIGURE 9.9.
A definition list is formatted slightly differently from other lists, with a series of hanging indents to separate terms from definitions.

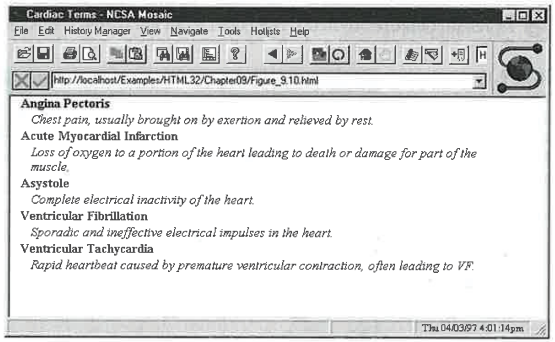


Like the other list tags and styles, it's also possible to include additional physical formatting tags within the terms or definitions.

```
<DT><B>Term</B>
<DD><I>Definition</I>
```

A popular method of implementing the definition list is to provide a bold style to the term, and regular or italics for the definition (as shown in Figure 9.10).

FIGURE 9.10.
Additional formatting is allowed within the definition list to help emphasize the graphical separation between term and definition.



Neither the `<DT>` nor the `<DD>` tag includes any attributes. Any additional formatting is handled through other tags, such as `<B>` (bold) or `<I>` (italic).

Now that you've seen the major styles for creating and displaying lists, it's time to take a look at the last two styles.

Using Logical List Styles

HTML offers an additional set of list tags that don't dictate a corresponding physical style. Like the logical block tags illustrated in the preceding chapter, these tags help show the relation of the list items to the rest of the document. Their actual implementation and display is not strictly defined and varies with different browsers and platforms.

The logical list styles are not in as widespread usage as <OL> or <UL>, but you can still use them to make your HTML source code more readable. Each style uses the same <LI> tag to mark each separate item, except for the definition list.

The first two tags are <MENU> and <DIR>, and their syntax is the same as an unordered list.

```
<MENU> <!-- or DIR -->
<LI>ListItem1
<LI>ListItem2
<LI>ListItemN
</MENU> <!-- or /DIR -->
```

Neither item has any attributes, other than the often ignored `compact`. The first tag, <MENU>, represents a list of choices, such as those used with submitting a form or selecting a response to a question. It is typically rendered the same as an unordered list.

HTML standards from the World Wide Web Consortium recommend a multicolumn directory list rendering for the <DIR> tag. None of the graphics-based browsers follow this advice, and instead use the standard <UL> implementation.

The representation of both items, compared to an unordered list, is shown in Figure 9.11.

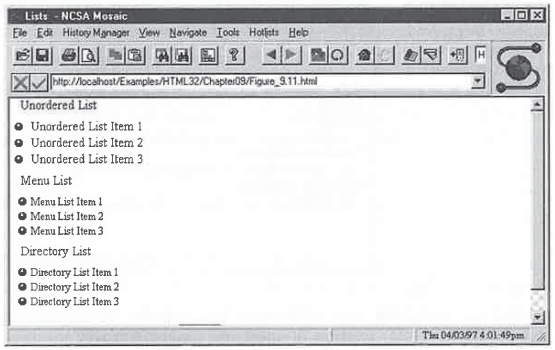


FIGURE 9.11.
Although slightly different typefaces are used in Mosaic, the basic representation of a menu and directory list is still an unordered list.

Summary

Two main categories of lists are used with HTML. First is an ordered list, which can be used with its attributes to create a variety of formats for your information. Second is the unordered list, with its options for displaying unprioritized information.

Also in this grab bag of display items are the definitions. These are unique in that they don't require a list item tag, but rather use a pair of tags to delineate the term from the definition.

Two additional tags, now virtually obsolete but still part of the standard, are for menus and directory listings. Their preferred implementation is ignored by the vast majority of browsers, which tend to represent the two as simple unordered lists.

The next chapter, "Creating Tables for Data and Page Layout," adds another option for displaying text. Together with the text formatting tags in the preceding chapter and the various types of lists in this chapter, you're developing quite an arsenal to get your point across with HTML 3.2.

CHAPTER 10

Creating Tables for Data and Page Layout

by Rick Darnell

IN THIS CHAPTER

Tables are kind of like lists. We're introduced to them at an early age through the mirth of games such as tic-tac-toe and checkers. Later in life, someone forces us to use a spreadsheet, and suddenly tables are not so fun anymore—unless you're one of those folks who also enjoys things like actuarial tables relating sedentary lifestyles to colon polyps.

HTML tables have established themselves as one of the most used and powerful tools in formatting Web pages. Tables were originally introduced to facilitate formatting of tabular data, such as spreadsheet and database information.

Using a little creative thought, Web designers have implemented table tags and their various attributes to allow a variety of uses not directly supported by HTML, including formatting text into multiple columns and hanging indents. As you begin to explore the nuances of the various table tags and their attributes, you'll probably begin to see additional applications.

On the downside, table tags are also some of the most complicated to understand and use. In the linear space of a page of HTML coding, you must define a two-dimensional form. It's hard to keep track of which row belongs to which information, and how a particular cell will look in its final form. A table-enabled browser is crucial if you're not using a visual HTML editor.

With that preamble and warning out of the way, it's time to get out the angle iron, nuts, and bolts to build the basic frame for your new vehicle.

Setting the Basic Constraints with <TABLE>

Like other block elements illustrated in earlier chapters, a table is marked at its beginning and end. The <TABLE> tag is used to fit this basic purpose. The syntax is as follows:

```
<TABLE attributes>
<CAPTION align=top¦bottom>captionText</CAPTION>
<TR attributes>
<TH>TableHeading</TH>
<TD>Cell1</TD>
<TD>Cell2</TD>
...
<TD>CellN</TD>
</TR>
...additional rows as needed...
</TABLE>
```

Each of the tags within the table is explained in this chapter, but here's a quick look at how each one relates to the whole:

- <TABLE> marks the beginning and end of a single table.
- <CAPTION> is an optional tag providing an attached caption for the table on the top or bottom.
- <TR> marks the beginning of a new row in the table.
- <TH> formats text as a heading in the affected row.
- <TD> defines the contents of a single cell within the table.

Following the sequence of tags, you can see that a table is constructed the same way a brick wall is made. Each row begins with a layer of cement (<TR>) and then adding bricks (<TD>) until the row is completed. Then, a new course is added until the wall is finished and capped (</TABLE>).

Now that the basic structure of a table has been explained, the next section looks at the basic attributes of <TABLE> for setting its overall appearance, both within the table itself and in relation to the browser window.

The Table in the Browser: align and width

The first two attributes for <TABLE>—align and width—control how the table relates to surrounding text and other elements when displayed on the browser.

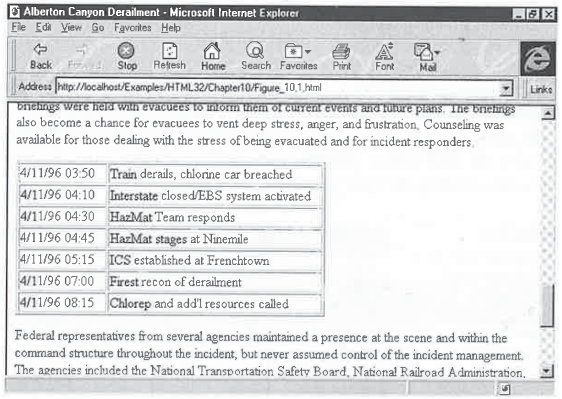
The syntax is

```
<TABLE align=left¦center¦right width=pixels¦percent%>
```

where align is the table's horizontal alignment on the page and width is the width of the table expressed in pixels or as a percentage of the browser window.

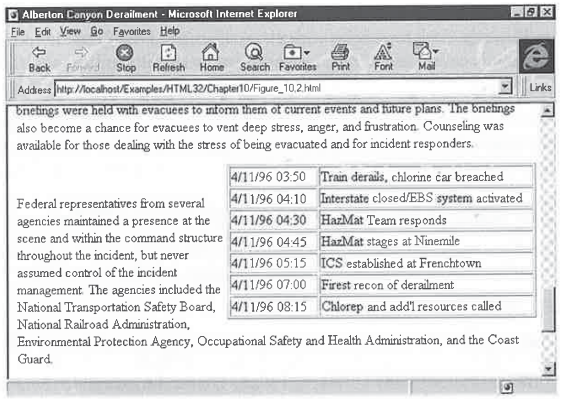
Let's take a look at align first. When this attribute is not included, the table defaults to the whims of the browser. Typically, this is flush with the left border, and surrounding text is interrupted by its presence. (See Figure 10.1.)

FIGURE 10.1.
The default alignment for a table breaks text before the table and continues it afterward, keeping the table flush against the left margin.



Using the values for the align attribute forces the table to one side of the browser window or the other, and the text now wraps around the outside border. (See Figure 10.2.)

FIGURE 10.2.
Adding `align=right` *to the* `<TABLE>` *tag results in the table being forced flush right with the text flowing around the outside border.*

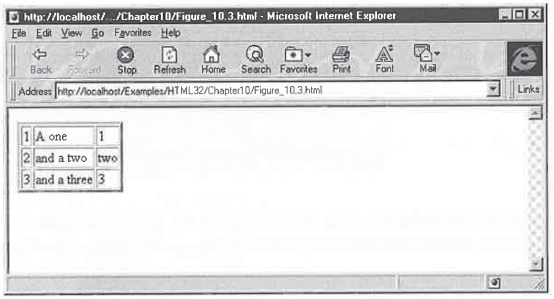


TIP

Although `center` is included as an option for the `align` attribute, it is not currently supported by any of the popular browsers. Using `align=center` results in a table that receives the browser's default treatment.

The next attribute for `<TABLE>` is `width`, which controls how much horizontal space is occupied in the browser window. By default, just enough space is allotted to display the widest value in each column. (See Figure 10.3.) For example, if the largest value in each column is a small integer, the table will be relatively narrow. On the other hand, if several columns have large obfuscated words, the table will be relatively wide.

FIGURE 10.3.
The default width for a table is just enough space to display its contents. Each column is sized to accommodate the widest element in the column.



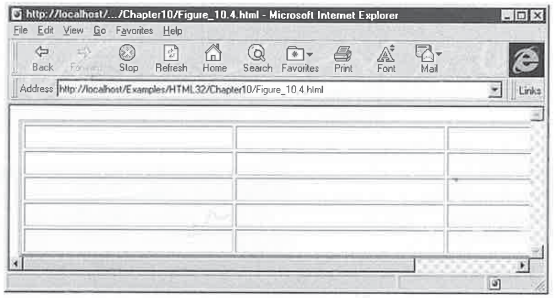
The `width` attribute helps to remove some of the guesswork about just how wide a table should be. It can accept values in two measurements—pixels or a percentage of window space. To specify pixels, use a single integer. To indicate a percentage of the browser window, use a single integer followed by a percentage sign.

TIP

If the table contains graphics, go with pixels. This forces the table to be wide enough to contain your images. Otherwise, select the percentage value for compatibility with a wide range of user displays.

The main difference between the two forms of measurement is how they're treated when the browser window is resized. If a table is set to 400 pixels wide, it remains 400 pixels wide, whether the browser is full-screen or occupies only a narrow strip on the left side. (See Figure 10.4.)

FIGURE 10.4.
Setting a physical width in pixels results in a constant table width, regardless of browser window size.



On the other hand, if a table is set to 100 percent of the browser window, its physical width will vary depending on the size of the browser window. (See Figures 10.5 and 10.6.) This ensures compatibility across different screens (VGA or SVGA) and browsers.

NOTE

Which measuring system should you use with your table? That's a hard call to make for Web authors because they have no control over the user's viewing environment, such as monitor resolution, color compatibility, browser window size, or whether the browser can handle tables at all. Make the best guess you can about the capabilities of your users, and

continues

continued

tailor your system to fit those folks. You might also need to find a way to present the information in a nontabular format for those with less-endowed browsers.

The bottom line in all of this is one simple fact: Someone will be left out, and there's nothing you can do about it. So be as accommodating as you can, and don't lose too much sleep over those who can't see what you've done.

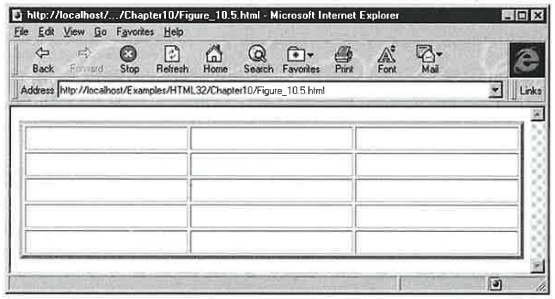


FIGURE 10.5.
This table is set to 100 percent of the width of the browser window. See what happens when the window is resized in Figure 10.6.

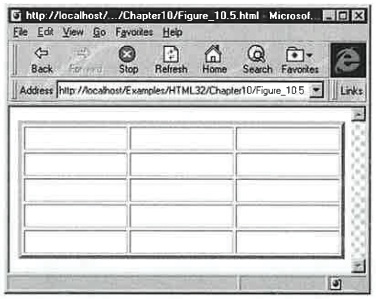


FIGURE 10.6.
The table still fills the browser window, without spilling over. It is automatically resized to fit 100 percent of the new window's size.

General Table Appearances: `border`, `cellspacing`, `cellpadding`

The next set of attributes control how the table itself looks by setting the width of the outside border (`border`), the width of the border between cells (`cellspacing`), and the space between cell contents and corresponding borders (`cellpadding`) The syntax is

```
<TABLE border=size cellspacing=size cellpadding=size>
```

where *size* is a value in pixels. If an attribute is omitted, the typical defaults are no border, two pixels for cell spacing, and one for cell padding. This may vary slightly, depending on the browser.

NOTE

The user has a say in the matter, too.

Many browsers have additional settings through preference dialogs that allow users to control the appearance of tables. Usually, this is limited to whether a table is displayed using one-dimensional lines or with the appearance of beveling and shading.

Keep in mind that your tightly formatted table might not appear exactly as you expect, depending on the software preferred by the end user.

Many people confuse the purposes of the `cellspacing` and `cellpadding` attributes. If it helps to relieve the confusion, think about creating more space between inmates in padded cells, or some other equally innocuous illustration that makes it easier to imagine the spaces associated with each cell. For the purposes of illustration, I've included some examples in Figure 10.7.

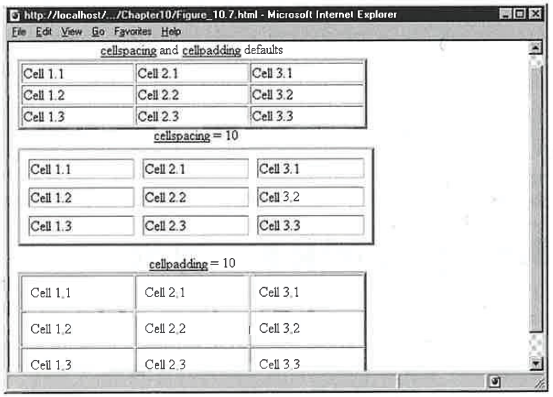


FIGURE 10.7.
Three variations for a table include a version with default values, one with increased cell spacing, and another with increased cell padding.

NOTE

Although not included in the HTML 3.2 specification, some browsers also support a `bordercolor` attribute to the `<Table>` tag. It takes a value of a recognized color literal (white, black, blue, and so on) or a hexadecimal color triplet preceded by `#`.

It's just one more control to further fine-tune the appearance of your Web pages and their contents, because the software manufacturers care so much about you.

<CAPTION>

You can add a brief description for your table by adding a `<CAPTION>` tag immediately following the opening `<TABLE>` tag.

The `<CAPTION>` tag is to the table what the `<TITLE>` tag is to the `<HEAD>` of the document, except you can specify whether the caption appears on the top or bottom of the table. The syntax is

```
<TABLE>
<CAPTION align=top¦bottom>CaptionText</CAPTION>
...table elements...
</TABLE>
```

where *`CaptionText`* is the description for the text, and the `align` attribute indicates whether the caption is attached to the top or bottom of the table. The default placement for a caption if the `align` attribute is not used is at the top.

TIP

The three tables in Figure 10.7 all used a top caption to identify what each one was showing.

Although not recommended by the HTML 3.2 specification, it's possible to format a caption in the same way as other HTML text by including physical styles such as bold and italic, or using the `<FONT>` tag and its attributes. There is no way to change the horizontal alignment of a caption; the text always remains centered over the table. Block-level tags, such as the heading tags and logical formatting styles, are not allowed within a caption. More information on using physical styles for formatting text is found in Chapter 8, "Text Alignment and Formatting."

Building the Table Row by Row

After the basic attributes of the table are defined and a caption is added (if desired), it's time to get down to the nitty-gritty of adding each table row. Remember that a table is built by marking a row, adding cells to it, and then starting a new row under it. In this way, the structure of the table in HTML code mimics the structure as it appears in the browser window.

Where the Row Starts: <TR>

The beginning of a row is marked using the `<TR>` tag with this syntax:

```
<TABLE>
<TR align=left¦center¦right valign=top¦middle¦bottom>
<TD>Cell1.1<TD>Cell2.1<TD>Cell3.1

<TR align=left¦center¦right valign=top¦middle¦bottom>
<TD>Cell1.2<TD>Cell2.2<TD>Cell3.2

...Additional rows and cells...

<TR align=left¦center¦right valign=top¦middle¦bottom>
<TD>Cell1.N<TD>Cell2.N<TD>Cell3.N
</TABLE>
```

In this syntax, `<TR>` is the beginning of a row of cell definitions. The `<TR>` tag doesn't require a closing tag, although you can use one to help make the boundaries of each row easier to identify.

There are two attributes for `<TR>` that set the default behavior for all cells in that row. Individual cells can override the values of `<TR>`.

The first attribute is `align`. This is similar to the `align` in the `<TABLE>` tag, but it is limited to controlling the horizontal alignment of the material in cells. The available values are `left` (default), `right`, or `center`. Figure 10.8 shows a table with these three options.

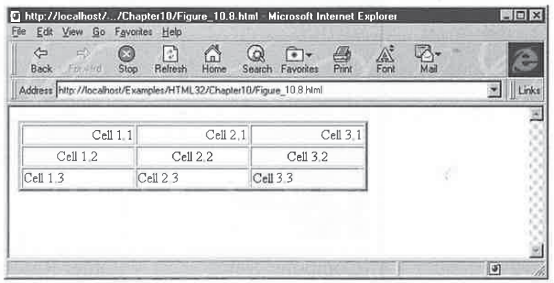


FIGURE 10.8.
This table has three rows, with each one set to a different horizontal alignment with the `<TR align>` *attribute.*

The next attribute, `valign`, controls the vertical placement of content in the cells. The default for most browsers is `middle`. As with most defaults, this isn't always the most aesthetically pleasing. (See Figure 10.9.)

As a general rule, rows should be set to `valign=top`. This is the way most people expect their tables to look. (See Figure 10.10.)

FIGURE 10.9.
The default vertical alignment for a table places content in the middle of the cell. With differing content length in adjoining cells, the visual result isn't always appealing.

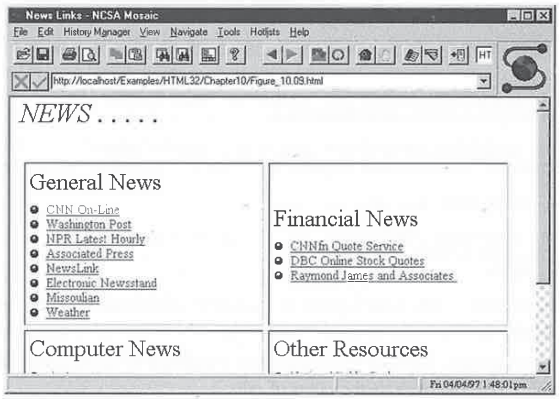
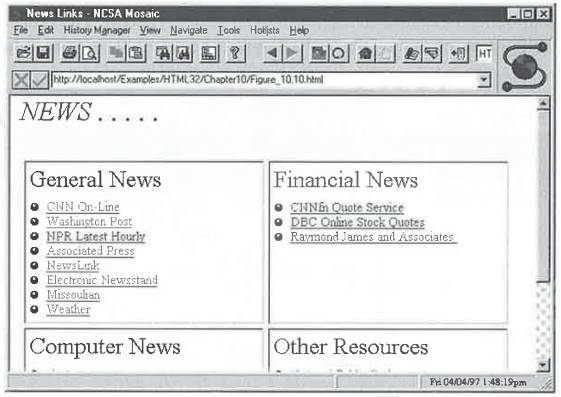


FIGURE 10.10.
This is the same table displayed in the preceding figure with the `valign` *attribute set to* `top` *in the* `<TR>` *tag.*



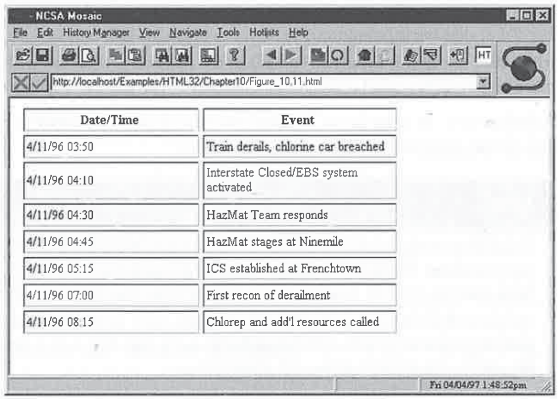
Defining Table Cells

With the beginning of the row marked with `<TR>`, it's time to finally get down to the work of filling each cell. Two types of cell tags are used in a table. The first, `<TH>`, marks a header cell, which is similar to a heading tag on a Web page. The other is a data cell tag, `<TD>`, used for the body of the table.

Header Cells: <TH>

A table header cell is much like any other table cell. Most browsers include a different font style for header cells to help reinforce their purpose to the user. (See Figure 10.11.)

FIGURE 10.11.
The first row in this table is defined using the `<TH>` *tag, and the second uses the* `<TD>` *tag. The browser emphasizes the first row to show its status as a header.*



TIP

Header cells are a logical definition, not a physical one. Interpretation of the `<TH>` tag is left up to the browser, and it varies across platforms and applications.

The syntax for `<TH>` is

```
<TABLE>
<TR><TH attributes>Cell Header
</TABLE>
```

where *`Cell Header`* is the header content of the cell, and *`attributes`* is blank or includes one or more of the attributes and its corresponding value. The various attributes are covered in detail in the next section.

Each header cell needs an opening tag, although a closing tag is not required. The cell can be placed at any point in the table, including mixed in on the same row with `<TD>` tags. A header cell has the same attributes as a content cell; content cells are covered in the following section on the table data tag.

Table Data Cells: <TD>

It's taken a little time to get here, but after the table framework is in place, you can begin adding each brick of content. The syntax for <TD> is

```
<TABLE>
<TR><TD attributes>Cell Content
</TABLE>
```

where *Cell Content* is the actual content of the cell, and *attributes* is blank or includes one or more of the attributes and corresponding values covered in more detail in the following sections.

A closing </TD> or </TH> is not required for individual cells, although many HTML designers and editing programs use it for additional clarity.

NOTE

Depending on the number of cells and their contents, there are two conventions for <TD> tag placement. The first is on the same row as the container <TR> tag. This means you'll have one line of HTML beginning with <TR> for every row in your document.

However, if you have many cells, or they include items such as hyperlinks or images, you'll probably want to go with each cell definition on its own line, with a closing </TR> tag and an extra carriage return at the end of every row.

In both cases, the code will behave the same way. It's up to you to choose a format that helps to clarify the code for future editing and revision.

When considering formatting options for each cell, think of the <TD> and <TH> tags as <BODY> tags. Any tag valid for the body of an HTML document is valid within a cell. This means you can embed text, graphics, forms, plug-ins and objects, applets, and even other tables. This feature is what gives tables their power when used to format a page.

The attributes for table cells are divided into two broad categories—those that affect the cell's size, and those that affect the cell's contents.

Cell Size Attributes: width, height, rowspan, and colspan

The first set of attributes set the size of a cell, including its capability to merge with an adjacent cell.

The syntax for the first two attributes is

```
<TD width=pixels¦percentage height=pixels>
```

where width is an integer representing a physical distance in pixels, or a portion of the overall table width followed by a percentage sign (%). The height of a cell is defined in pixels.

The default value for width is just wide enough to accommodate the cell's contents or the widest corresponding cell in the same position in another row.

The height attribute sets the height of a cell in pixels. The default value is just tall enough to contain the longest content in the current row. For example, if the row included a cell with one sentence and another cell with several paragraphs, both cells would be tall enough to accommodate the paragraphs. This of course leaves a great deal of white space in the cell with the single sentence.

The next two attributes—rowspan and colspan—are used to combine adjacent cells into larger cells. It's important to note that when you are using these attributes, the adjacent cells aren't eliminated; they're just "hidden" while the acquiring cell uses their space. The syntax is

```
<TD rowspan=numRows colspan=numCols >
```

where *numRows* is the number of rows, including the current cell, joined together. Likewise, *numCols* is the number of columns joined together. The default for both values is 1.

Spanning is a little tricky to plan and implement. For example, start with a 3×3 table. This requires three rows with three cells each. (See Listing 10.1.)

Listing 10.1. Basic HTML for displaying a 3×3 table.

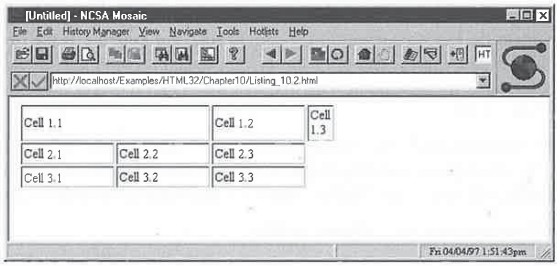
```
<HTML>
<HEAD>
 <META HTTP-EQUIV="Content-Type" CONTENT="text/html">
</HEAD>
<BODY>
<TABLE border=3 width=75%>
 <TR> <TD>Cell 1.1  <TD>Cell 1.2  <TD>Cell 1.3
 <TR> <TD>Cell 2.1  <TD>Cell 2.2  <TD>Cell 2.3
 <TR> <TD>Cell 3.1  <TD>Cell 3.2  <TD>Cell 3.3
</TABLE>
</BODY>
</HTML>
```

Now you can begin merging cells by combining Cells 1.1 and 1.2. Because this reaches across columns, it's a column span. (See Listing 10.2 and Figure 10.12.) To do this, colspan=2 is added as an attribute to the table data tag for Cell 1.1.

Listing 10.2. A column span across Cells 1.1 and 1.2 results in the addition of the colspan attribute.

```
<HTML>
<HEAD>
 <META HTTP-EQUIV="Content-Type" CONTENT="text/html">
</HEAD>
<BODY>
<TABLE border=3 width=75%>
 <TR> <TD colspan=2>Cell 1.1  <TD>Cell 1.2  <TD>Cell 1.3
 <TR> <TD>Cell 2.1            <TD>Cell 2.2  <TD>Cell 2.3
 <TR> <TD>Cell 3.1            <TD>Cell 3.2  <TD>Cell 3.3
</TABLE>
</BODY>
</HTML>
```

FIGURE 10.12.
The revised table with a column span. Notice what's happened with the bottom two rows of cells.



As you see in Figure 10.12, spanning cells isn't quite as simple as just adding the span attributes. Here's what happened. When the browser encountered `colspan=2`, it knew Cell 1.1 needed to take up the space of two cells, and it provided the space accordingly. Then, it continued across the row and finished adding the next two cells. The result was space for a total of four cells (two joined and two individuals).

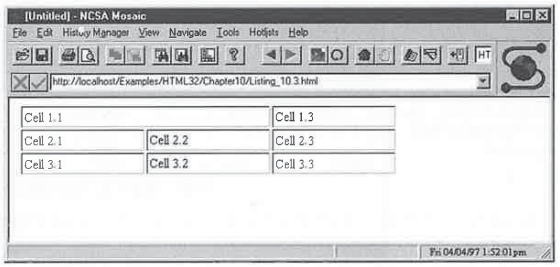
At the next row tag, the browser started a new row by adding three cells. The browser didn't encounter a fourth `<TD>` tag, even though there was room in the table for one. Instead of adding a cell that the user didn't specify, it simply filled in with blank space. The final appearance was a 3×4 table with the bottom two rows only partially defined.

This feature of cell spanning is the cause for a great many headaches when you are working with tables. For each cell you span, you need to remove the corresponding cell definition. (See Listing 10.3 and Figure 10.13.)

Listing 10.3. A column span across Cells 1.1 and 1.2 requires the addition of the `colspan` attribute and removal of the adjoining tag.

```
<HTML>
<HEAD>
 <META HTTP-EQUIV="Content-Type" CONTENT="text/html">
</HEAD>
<BODY>
<TABLE border=3 width=75%>
 <TR> <TD colspan=2>Cell 1.1                <TD>Cell 1.3
 <TR> <TD>Cell 2.1          <TD>Cell 2.2  <TD>Cell 2.3
 <TR> <TD>Cell 3.1          <TD>Cell 3.2  <TD>Cell 3.3
</TABLE>
</BODY>
</HTML>
```


FIGURE 10.13.
With the adjoining data cell removed, the table retains its original 3×3 appearance.



TIP

It might be a bit awkward to edit tables in the graphical manner displayed in Listings 10.2 and 10.3, but it's very useful for determining which cells should be removed for spanning.

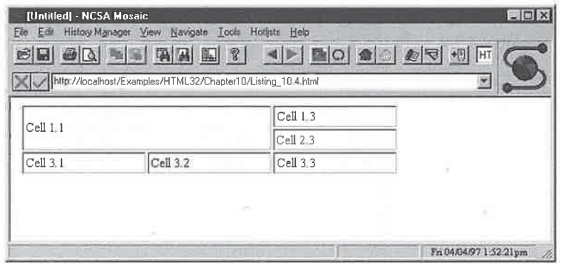
The same rules also apply for the `rowspan` attribute, except cells below the originating tag should be removed. It's also possible to combine the two attributes for other effects. (See Listing 10.4 and Figure 10.14.)

Listing 10.4. Using the same basic table, the first cell is combined with the next row and next column. The result requires removing three adjoining cell tags.

```
<HTML>
<HEAD>
 <META HTTP-EQUIV="Content-Type" CONTENT="text/html">
</HEAD>
<BODY>
<TABLE border=3 width=75%>
 <TR> <TD colspan=2 rowspan=2>Cell 1.1              <TD>Cell 1.3
 <TR>                                               <TD>Cell 2.3
 <TR> <TD>Cell 3.1                    <TD>Cell 3.2  <TD>Cell 3.3
</TABLE>
</BODY>
</HTML>
```

When spanning in two directions at once, you need to delete all the data cells in the path of the merge. This means removing cells from the right, left, and diagonal directions.

FIGURE 10.14.
There are now four adjoining data cells, which have been combined into one large cell. Still, the table retains its original 3×3 appearance.



Cell Content Attributes: `align`, `valign`, and `nowrap`

The next group of attributes control how text and other content is placed within the cell. The syntax for these attributes is

```
<TD align=left¦center¦right valign=top¦middle¦bottom nowrap>
```

where `align` controls horizontal justification, `valign` controls the vertical justification, and `nowrap` forces text content within the cell to remain on one line.

The values and behavior for `align` and `valign` are the same as the `<TR>` tag, and they override its values for the current cell only.

The presence of the last attribute, `nowrap`, disables automatic word wrap within the cell's borders depending on other table settings, such as fixed or proportional width of a cell or the entire table. This is equivalent to using the ` ` character entity for inserting nonbreaking spaces within the content of the cell.

If the table doesn't have a specified width, the affected cell expands horizontally to accommodate the text. The same is true if the table width is specified as a percentage of the browser window, and the table's width will expand to accommodate the larger cell.

If the table `width` is set in pixels, the cell will expand to the pixel width, leaving the absolute minimum space necessary for other columns that don't have a `width` setting in pixels. If that's not enough space, the browser starts to insert line breaks as needed.

In short, if the `nowrap` attribute is used in conjunction with any `width` settings specified in pixels, the `width` settings have priority, and line wrapping is reinstated.

TIP

A similar effect is accomplished by omitting `nowrap` and using nonbreaking spaces (` `) between words that should remain together on the same line. This enables you to keep certain words together and allow the others to flow to a new line as needed.

Whose Default Overrides Whose?

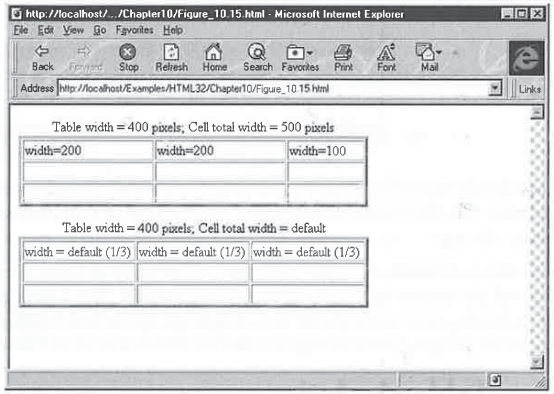
As you've seen in the tags leading to the actual cell data tag, there have been several opportunities to influence the width of a cell. So whose attribute takes precedence?

As a general rule, the last attribute encountered sets the standard for the rest of the table, although this is not always the case. Consider the following table definition:

```
<TABLE width=400>
<CAPTION>Table width=400, Cell width=500</CAPTION>
<TR><TD width=200>200 pixel-wide cell <TD width=300>300 pixel-wide cell
</TABLE>
```

In its definition, the table is set to 400 pixels wide. However, in the first two data cell definitions, the width is forced to 500 pixels. What happens? The `<TABLE>` tag takes control in this situation, with an interesting result. (See Figure 10.15.)

FIGURE 10.15.
Compare these two tables. The first one is set to 400 pixels wide, although the data cells have width settings that add up to 500 pixels. The second table is the same as the first, without width settings for the data cells.



Here's what happens. The `width` setting in the `<Table>` tag sets its physical width, which is cast in stone. When the browser encounters the data cells with their `width` attributes, it looks at the

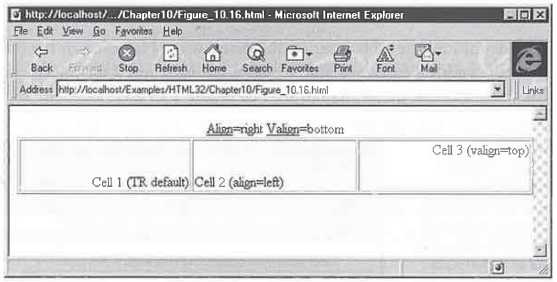
total pixels requested for the row. Because the value is greater than allowed, the browser does a little math and divides space between the two cells in the same proportion as their `width` attributes.

Now, look at another example, this time involving text alignment:

```
<TABLE>
<CAPTION>Align=right Valign=bottom</CAPTION>
<TR align=right valign=bottom>
  <TD height=50>Cell 1 (TR default)
  <TD align=left>Cell 2 (align=left)
  <TD valign=top>Cell 3 (valign=top)
</TABLE>
```

The `height` attribute is used to exaggerate the cell size for a better view of vertical alignment. The first cell in the table accepts the defaults from the `<TR>` tag. Cell 2 overrides the horizontal alignment and forces the text to the left, while Cell 3 overrides the vertical alignment and forces the text to the top. (See Figure 10.16.)

FIGURE 10.16.
Although cell alignment is set to the bottom right, the next two cells each override one of the defaults.



Note that the next cell in a row does not inherit any settings from the preceding cell. Each data cell starts with the defaults defined in the `<TR>` tag until an attribute is specifically overridden in the `<TD>` tag.

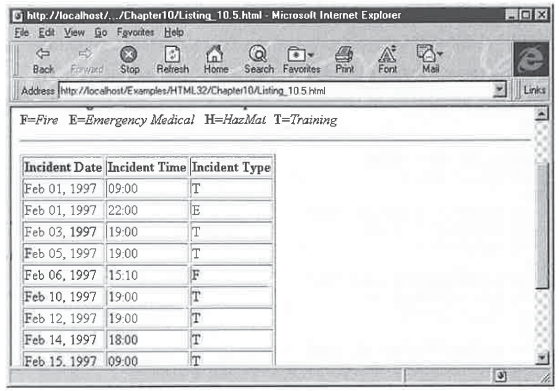
Alas, the world is full of exceptions to rules. In this case, the `height` attribute used in the first data cell also sets the height for all other cells in that row. This is done to ensure the integrity of the borders used to outline the cells. If each cell had its own height, the result would be a mishmash of zigzags and stairsteps as a line was drawn around each unique cell.

Two Tables in Action

Now that you've been through the syntax for tables, it's time to take a look at some practical examples of how tables are used on the World Wide Web.

The exhibit begins with a basic table for displaying tabular data. (See Figure 10.17.) In this case, it's a list of events for a single user generated from a database with a CGI script. Although this wasn't a human-generated page, the table formatting follows the HTML standard. (See Listing 10.5.)

FIGURE 10.17.
This page demonstrates one of the more traditional uses of a table—the display of raw data in table form.



Listing 10.5. This HTML page was generated by a CGI script and includes a table for displaying tabular information.

```
<HTML>
<HEAD>
<!-- This page was generated by Macromedia's Backstage Object Server. -->
<META HTTP-EQUIV="Content-Type" CONTENT="text/html">
<TITLE></TITLE>
<BASE HREF="http://localhost/IncidentLog/">
</HEAD>
<BODY>
<H1><FONT FACE="Garamond" COLOR="Red">Volunteer Time Record</FONT></H1>
<TABLE BORDER="0" WIDTH="75%" CELLSPACING="0" CELLPADDING="0">
<CAPTION ALIGN="TOP"><B><I>Prepared for the
  month of February 1997</I></B></CAPTION>
<TR ALIGN="CENTER" VALIGN="CENTER">
  <TD WIDTH="50%" NOWRAP="NOWRAP"><B>Firefighter ID: 294</B></TD>
  <TD WIDTH="50%" NOWRAP="NOWRAP"><B>Report Date: 11-Mar-97</B></TD>
</TR>
</TABLE>
<B>F</B><I>Fire</I>  
<B>E</B><I>Emergency Medical</I>  
<B>H</B><I>HazMat</I>  
<B>T</B><I>Training</I>
<HR>
<TABLE BORDER>
```

continues

Listing 10.5. continued

```
<TR>
  <TH>Incident Date
  <TH>Incident Time
  <TH>Incident Type
<TR>
  <TD ALIGN=LEFT>Feb 01, 1997
  <TD ALIGN=LEFT>09:00
  <TD ALIGN=LEFT>T
<TR>
  <TD ALIGN=LEFT>Feb 01, 1997
  <TD ALIGN=LEFT>22:00
  <TD ALIGN=LEFT>E
<TR>
  <TD ALIGN=LEFT>Feb 03, 1997
  <TD ALIGN=LEFT>19:00
  <TD ALIGN=LEFT>T
<TR>
  <TD ALIGN=LEFT>Feb 05, 1997
  <TD ALIGN=LEFT>19:00
  <TD ALIGN=LEFT>T
<TR>
  <TD ALIGN=LEFT>Feb 06, 1997
  <TD ALIGN=LEFT>15:10
  <TD ALIGN=LEFT>F
<TR>
  <TD ALIGN=LEFT>Feb 10, 1997
  <TD ALIGN=LEFT>19:00
  <TD ALIGN=LEFT>T
<TR>
  <TD ALIGN=LEFT>Feb 12, 1997
  <TD ALIGN=LEFT>19:00
  <TD ALIGN=LEFT>T
<TR>
  <TD ALIGN=LEFT>Feb 14, 1997
  <TD ALIGN=LEFT>18:00
  <TD ALIGN=LEFT>T
<TR>
  <TD ALIGN=LEFT>Feb 15, 1997
  <TD ALIGN=LEFT>09:00
  <TD ALIGN=LEFT>T
<TR>
  <TD ALIGN=LEFT>Feb 17, 1997
  <TD ALIGN=LEFT>12:30
  <TD ALIGN=LEFT>E
</TABLE>
<HR>
<P> </P><P> </P>
</BODY>
</HTML>
```

In addition to displaying the date, notice the use of the table header tag, <TH>, in the first row to identify the contents of each column. Other than alignment, no other text formatting is used in any of the cells; the difference between the <TH> and <TD> cells comes entirely from the browser's interpretation of each.

The next example is one developed to display information in a slightly different column format. It consists of a simple 2×2 table that includes a hanging headline in the left column and the corresponding story in the right column. (See Figure 10.18 and Listing 10.6.) The headlines are set with `valign` to the top of the cell so that they always appear at the beginning of the story, regardless of story length. The length of the text in the adjoining cell becomes irrelevant because it automatically grows to whatever is required.

No border is displayed, making the table's role in the formatting invisible to the user.

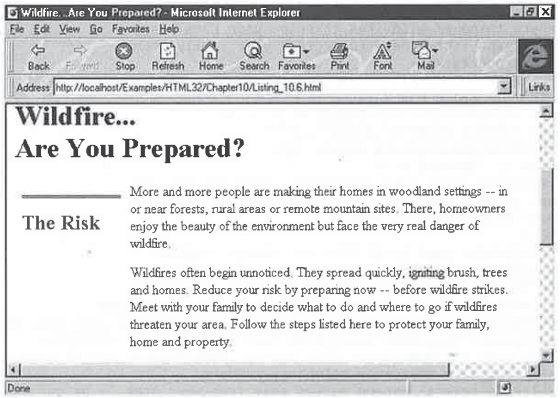


FIGURE 10.18. *Although HTML doesn't directly support columns, the use of tables enables an author to create a page with vertical columns and hanging headlines.*

Listing 10.6. This page uses a table to format two short articles with headlines that "hang" on the left side of the story.

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
<TITLE>Wildfire...Are You Prepared?</TITLE>
</HEAD>
<BODY>
<IMG SRC="/ClipArt/Graphics/bar2.gif" WIDTH="720" HEIGHT="14">
<H1>
<IMG SRC="/ClipArt/Misc/Fire/fire1.jpg"
  WIDTH="214" HEIGHT="148" ALIGN="ABSMIDDLE" HSPACE="8">
  Wildfire...Are You Prepared?</H1>

<TABLE WIDTH=100% CELLSPACING=4" CELLPADDING=4>

<TR> <!--Begin Row 1-->
<TD VALIGN=TOP WIDTH=20%> <!--Headline 1-->
<HR NOSHADE SIZE=4 ALIGN=LEFT>
```

continues

Listing 10.6. continued

```
<H2>The Risk</H2>
<TD> <!--Content 1-->
<P>More and more people are making their homes in
woodland settings -- in or near forests, rural
areas or remote mountain sites.  There, homeowners
enjoy the beauty of the environment but face the
very real danger of wildfire.</P>
<P>Wildfires often begin unnoticed.  They spread
quickly, igniting brush, trees and homes.  Reduce
your risk by preparing now -- before wildfire strikes.
Meet with your family to decide what to do and
where to go if wildfires threaten your area. Follow
the steps listed here to protect your family, home
and property.</P>

<TR> <!--Begin Row 2-->
<TD VALIGN="TOP"> <!--Headline 2-->
<HR NOSHADE SIZE="4" ALIGN="LEFT">
<H2>Practice Wildfire Safety</H2>
<TD> <!--Content 2-->
<P>People start most wildfires...find out how you
can promote and practice wildfire safety around
your home and property.</P>
<UL>
<LI>Contact your local fire department, health
department or forestry office for information
on fire laws.
<LI>Make sure that fire vehicles can get to your home.
<LI>Clearly mark all driveway entrances and display your name and address.
<LI>Report hazardous conditions that could cause a wildfire.
<LI>Teach children about fire safety. Keep matches out of reach.
<LI>Post fire emergency telephone numbers.
<LI>Plan several escape routes away from your home -- by car and by foot.
</UL>
<P>Talk to your neighbors about wildfire safety.
Plan how the neighborhood could work together after
a wildfire.  Make a list of your neighbors' skills
such as medical or technical.  Consider how you
could help neighbors who have special needs such as
elderly or disabled persons.  Make plans to take
care of children who may be on their own if parents
can't get home.</P>

</TABLE>
</BODY>
</HTML>
```


Summary

Tables are a series of nested tags, beginning with the initial <TABLE> tag, descending to the first <TR> tag, and culminating with a series of <TD> tags. Building a table requires the use of a fistful of tags and their associated attributes, but keeping the basic order in mind makes the process easier.

Tables are one of the container classes in HTML that have the most associated tags and attributes, which also makes tables one of the hardest HTML elements to master. However, if you take the time to build up your tables row by row and cell by cell with a clear mind toward your goal, you'll find them to be one of the most powerful and useful tags in the HTML arsenal.

CHAPTER 11

Linking Documents and Images

by Rick Darnell

IN THIS CHAPTER

What made the World Wide Web such a popular place? Was it the easy-to-remember URLs for finding resources? Was it the site dedicated to the Homemakers for Psychokinesis Society? Was it a million-and-one personal home pages that begin with, "Here are some pictures of my cat?" To be sure, those parts certainly had an effect. But probably the biggest draw to the Web was (and continues to be) the ability to wander around and encounter new things that look interesting.

From a page on astrophysics, it's possible to jump to a page about the moon, and from there to a page on Neil Armstrong, then on to information about Congress, and finally to a company that sells paper shredders. All of this is possible via hypertext—interconnected pages of information embedded with bits of highlighted text called *hyperlinks* to click for navigation. For users, the best thing about hyperlinks is that users don't have to know about how to construct a URL. They simply begin with a home page, a search engine, or a list of bookmarks, and start clicking.

I think it's safe to say at this point that anchors are probably the most important tags on the World Wide Web, and they are in large part the reason for the Web's overwhelming success and the acceptance of hypertext as a way to navigate documents.

WHAT EXACTLY IS HYPERTEXT?

Hypertext refers to a way of preparing and publishing documents that enables readers to follow their own paths through the information. The traditional path through a published document is sequential—beginning at page 1 and continuing through to the end. Hypertext allows the reader to jump around within the document and, often, to completely different documents. This method of working through material is well suited to an electronic format.

There are two basic components to hypertext: nodes and hyperlinks. A *node* is a unit of information, such as a Web page. It is small enough to manage and stand on its own. A *hyperlink* is a navigation tool that enables the reader to jump to a new document; typically, it is embedded in the text. A collection of nodes connected by hyperlinks is known as a *web*. Is this all starting to sound familiar? Moving between nodes on a web is called *browsing*.

For example, suppose you were looking at a page (node) about John Wesley that was talking about his influences and place within the Anglican church. The word "Anglican" could be a path (hyperlink) to another document that specifically talks about the Anglicans and their structure and role in 17th century England.

Hypertext lends itself very well to a variety of applications, including large amounts of information (encyclopedias, dictionaries, and other reference books) and multivolume series (magazines, newsletters, and digests).

Each hyperlink is put in place with a special tag, called an *anchor*. Depending on its attributes, an anchor has two functions. First, as explained, it's a path to another document, part of a document, image, sound, or any other file. Second, an anchor marks a destination in a document that is accessible by using the hyperlink version of the tag elsewhere.

The Anchor Tag: <A>

An anchor is a special text element that requires an opening and closing tag. If it is used as a hyperlink, there should be something in between, such as text or an image. When used to mark a destination, an anchor can be included with nothing in between. (For more information on using anchors to mark destinations, see "Creating Anchors to Locations Within Documents," later in this chapter.)

The syntax is

```
<A attributes>[HTMLcontent]</A>
```

where `attributes` are one of five choices (`name`, `href`, `rel`, `rev`, or `title`) and `HTMLcontent` is an optional value, which can include text, graphics, or other valid Web page content.

These tags cannot be nested within each other. Other text formatting tags are allowed within anchors, and anchors are allowed within other HTML elements such as tables and headings.

Picking a Destination with `href`

The first and most often used attribute is `href`, used to create a hyperlink to another document. The syntax is

```
<A href="URL">HTMLcontent</A>
```

where `URL` is an absolute or relative URL to another document or anchor tag within a document, and `HTMLcontent` is the material that appears on the user's browser to click.

TIP

Just in case you're curious, `href` stands for *Hypertext Reference*.

The URL can take several forms, depending on the desired destination. For a location outside the realm of the host server, use a complete URL such as

```
http://www.wossamotta.edu/frostbitefalls/flash.html
```

TIP

Other Internet protocol designations are allowed within `href`, including FTP, Gopher, and Usenet. How these others are handled depends solely on the capability of the user's browser.

As a general rule, most browsers support a wide variety of the most common protocols. All the same, it's a good idea to let users know your intentions before leading them outside the Web into other realms of the Internet.

The appearance of the text between an opening and closing anchor tag with the `href` attribute depends on the link's status, and it varies across browsers and platforms. A general standard has evolved of underlined blue text for unvisited links, although this is not a hard and fast rule. Most browsers allow users to change this setting, and additional attributes of the `<BODY>` tag also permit changing the hyperlink's appearance.

A general color standard for active hyperlinks (the mouse button is clicked on the link, but not yet released) and visited hyperlinks has not been established. For more information on changing the appearance of hyperlink text, see Chapter 7, "Structural Elements and Their Usage," or Chapter 19, "Introducing Cascading Style Sheets."

Nontextual content contained by anchor tags is handled in a slightly different manner. The browser often puts a border around the content with the link color corresponding to the current status. Like other browser features, this feature can be modified within the browser settings or by HTML attributes for the tags that inserted the content.

You can also use the `href` attribute to create hyperlinks to non-HTML content, such as electronic documents and virtual reality pages. For more information on creating these links, see Chapter 13, "Integrating Multimedia and Other File Types."

Working with `href` and Relative Links

A complete URL is not needed for the `href` attribute to identify a Web page. If the Internet protocol and server address are omitted from the URL, the `href` value is treated as a value relative to the current server. For example, the host server `www.wonka.chocolate.com` has a home page with the following lines of HTML:

```
<A href="/newproducts/everlasting.gobstopper.html">Perfect Candy</A>
for children with very little pocket money.
```

When clicked by the user, the URL loaded by the browser is

```
www.wonka.chocolate.com/newproducts/everlasting.gobstopper.html
```

The leading slash (`/`) in the `href` value tells the browser to go to the root directory of the current server and build a path to the file from there.

The value of the base URL changes after connecting to the Everlasting Gobstopper page. Suppose that an anchor tag is on the Gobstopper page that looks like this:

```
<A href="fizzy.lifter.drink.html">It's a gas!</A> Look for the newest Wonka soda.
```

Without any other URL information—no server address or path information—the relative link assumes the path from the current page, leading the browser to load the following address:

```
www.wonka.chocolate.com/newproducts/fizzy.lifter.drink.html
```

For a final wrinkle in this whole relative URL business, assume that a new path called `/comingsoon/` is created for the Fizzy Lifter Drink under New Products. The anchor tag for the Fizzy Lifter page from the Gobstopper page now looks like this:

```
<A href="newproducts/fizzy.lifter.drink.html">It's a gas!</A>
Look for the newest Wonka soda.
```

What does the browser do with this? The same thing it did when the filename stood on its own: It adds all the current path information from the current document to the front, and uses the result to retrieve the target document. The value of the complete URL is now

```
http://www.wonka.chocolate.com/newproducts/comingsoon/fizzy.lifter.drink.html
```

The `<BASE>` Tag Revisited

Back in Chapter 7, you learned a little about the `<BASE>` tag. This tag is part of the HTML document's head when a new starting point for the relative tag is needed. The syntax is

```
<BASE HREF="protocol://servername/path/">
```

where *`protocol`* is an Internet communication standard such as HTTP, *`servername`* is a server name or address such as `www.wossamotta.edu` or `89.123.32.21`, and *`path`* is any additional mapping on the server. The path is an optional value to the URL. If the path is included by itself, it refers to the host server.

Its use follows the same principles illustrated in the preceding section for relative links. For example, all of the hyperlinks on the Wonka Chocolate Factory home page are relative links. If the Wonka Chocolate Factory received space on the Loompa Land server (`www.oompaloompa.lo`) to place a copy of the Wonka home page, all the original hyperlinks on the mirrored home page would need to be rewritten because the rest of the site remains at Wonka's server. As it stands, without the `<BASE>` tag and the page in its new location, the hyperlink

```
<A href="/favorites/wonkabar.html>Our bestselling candy</A>
```

would be interpreted by a browser as `http://www.oompaloompa.lo/favorites/wonkabar.html` and generate a `404 Not Found` error for the user.

To avoid the hassle of rewriting the links, Wonka adds a new line to the <HEAD> of the document, which is placed on the Loompa Land server:

```
<BASE href="http://www.wonka.chocolate.com">
```

Now, all relative links on that page are accessed using Wonka's location instead of Loompa Land. Essentially, by adding one line to the remote Wonka home page, an entire site was mirrored on another server with just one file. Using the preceding example, the link is now interpreted correctly by the browser as

```
http://www.wonka.chocolate.com/favorites/wonkabar.html
```

TIP

The value of <BASE> is ignored by anchor tags when the anchor's href attribute includes a full URL. Remember that a relative link is used only when a server name or path is not used.

Picking a target Frame

HTML 3.2 brings with it the formal advancement of frames, following the lead of frame implementation in popular browsers such as Netscape Navigator and Microsoft Internet Explorer. Frames allow subdividing of the browser window into separate sections. Each section can load and display a different document independent of the other frames on the page. Its primary use in HTML has been for placing information required for a range of pages, such as a table of contents or advertising. For more information about working with frames, see Chapter 18, "Creating Sophisticated Layouts with Frames and Layers."

As a companion to the frame implementation, the target attribute was added to the anchor tag. This enables you to specify a place other than the current window to load a URL. The syntax is

```
<A href="URL" target="frameName">AnchorText</A>
```

where *URL* is the location of the destination document and *frameName* is the name of the frame that will load the destination document. When the user clicks *AnchorText*, the current frame is unaffected while the destination document is loaded in the target.

This assumes that the target frame has been named as part of the <FRAME> tag. Without using a scripting language, it's impossible to identify individual frames without a unique name.

Sending E-mail with an Anchor Tag

As you've probably seen on some HTML pages, it's possible to include a hyperlink whose destination is an e-mail address. This is accomplished with a special protocol—mailto:.

The e-mail protocol is used slightly differently than other protocols. Its syntax with an anchor tag is

```
<A href="mailto:name@domain">AnchorText</A>
```

where name@domain is an e-mail address such as charlie@chocolate.factory.com, and *AnchorText* is a line of text or image that the user clicks to activate the link.

On e-mail–enabled browsers, such as NCSA Mosaic, Netscape Navigator, or Microsoft Internet Explorer, the integrated mail program is started. In any case, the e-mail address is automatically loaded into the "send to" field. NCSA Mosaic and Netscape both support an additional feature to place a default subject line in the message.

Mosaic makes use of the title attribute for e-mail; it's the only browser that does so. The syntax is

```
<A href="mailto:emailAddress" title="SubjectLineText">AnchorText</A>
```

where *emailAddress* and *AnchorText* are the same as the standard syntax for mailto:, and *SubjectLineText* is what will appear in the subject line of the message, such as "About the Wonka Bar page."

For Netscape, the syntax is slightly different. The subject line is appended to the e-mail address and the title attribute is not used. The syntax is

```
<A href="mailto:emailAddress?SubjectLineText">AnchorText</A>
```

For future compatibility, it's a good idea to stick with the title attribute and forgo the Netscape method. Using the e-mail extension is not guaranteed unless you can guarantee that all of your users are using Netscape. With other browsers, the entire e-mail address and extension is placed in the "send to" field, which could cause problems with mail servers.

The title attribute for the anchor tag is a recognized part of the HTML standard and stands a better chance of eventual integration in the majority of browsers. Plus, if a browser doesn't support it, no problems are created for the user in the address field.

Defining Relationships with rel and rev

These two tags are used to define the type of hyperlink specified with the href attribute. As such, rel and rev are used only in conjunction with href. The syntax is

```
<A href="URL" rel=type rev=type>AnchorText</A>
```

where *type* defines the relation of the current document to the link specified in href. The first, rel, indicates a forward relationship; its counterpart, rev, shows a reverse relationship.

The attributes can be used alone or together. Like their counterparts in the <LINK> tag, rel and rev are not supported by the vast majority of browsers, including the most popular offerings. More information about using the <LINK> tag is found in Chapter 7.

Identifying an Anchor by Its name

This attribute is most commonly used by anchors marking a hyperlink destination within a document, although it can also be used to name `href` hyperlinks. Its syntax is

```
<A name="AnchorName">[HTMLcontent]</A>
```

where *AnchorName* is a string unique to the current document and *HTMLcontent* is optional material associated with it that is viewed by the user.

When used in conjunction with the `href` tag, the `name` attribute creates a dual-purpose anchor. The anchor is both a destination and a hyperlink, such as the following:

```
What's new on the <A href="www.wossomatta.edu" name="wossomatta">
Wossomatta Virtual Campus</A>
```

For more information on linking to specific points within a document, see "Creating Anchors to Locations Within Documents," later in this chapter.

Unique naming is important for anchors. Having the same name for a variety of anchors leads to unpredictable results, depending on the browser. Some browsers pick the first anchor that matches, others pick the last, and still others won't pick anything at all.

Although they are permitted by many browsers, avoid using spaces and slashes in the anchor name. This can create confusion and errors when users attempt to type the name directly or create a bookmark to the location. As a form of generally accepted syntax, use an underscore character (_) instead of a space in anchor names.

Preview a Hyperlink with Its `title`

The `title` attribute is used only in conjunction with the `href` to provide information about the link. When used, it should provide the same name that is contained in the `<TITLE>` tag of the document specified with `href`. The syntax is

```
<A href="URL" title="documentTitle">
```

where *URL* is the document's address and *documentTitle* is its formal title specified in the HTML head.

On some browsers, including the title in the hyperlink enables the document title's display in the browser title bar as the Web server starts to deliver the document. However, not all document types support titles. For example, Gopher and FTP directory listings don't provide titles as part of their information.

NCSA Mosaic supports the `title` attribute to specify an e-mail message subject when used with `mailto` references. For more information about this usage, see "Sending E-mail with an Anchor Tag," earlier in this chapter.

Creating Anchors for Locations Within Documents

Some documents are well-served by having internal bookmarks to facilitate navigation within the document itself. This includes FAQs (Frequently Asked Questions), glossaries and definition lists, and long documents covering several topics or subtopics.

The syntax for creating a destination anchor is

```
<A name="UniqueName">[AnchorText]</A>
```

where *UniqueName* is a name for the anchor that is unique to the document and *AnchorText* is optional text displayed on the screen. Duplicate names can cause unpredictable behavior or errors.

There are two generally accepted methods for marking anchors within a document. The first is to place the anchor tag immediately before the content you're marking, without enclosing it. This means the closing tag, `</A>`, is placed immediately following the `<A>` tag. (See Listing 11.1.)

Listing 11.1. The first method for marking an anchor is to place the tags immediately before the heading of the destination, but not enclosing it.

```
<HTML>
<HEAD>
<TITLE>Anchors</TITLE>
</HEAD>
<BODY>
<A name="Topic_1"></A><H2>Topic 1</H2>
Topic 1 stuff.
<A name="Topic_2"></A><H2>Topic 2</H2>
Topic 2 stuff.
<A name="Topic_3"></A><H2>Topic 3</H2>
Topic 3 stuff.
</BODY>
</HTML>
```

This is an effective way to mark anchors within a document, but it does have a drawback. Because the anchor tags don't contain anything, they are sometimes inadvertently separated from the places they're supposed to mark.

The other method is similar to marking text for a hyperlink. (See Listing 11.2.)

Listing 11.2. Another method for marking an anchor is to enclose the heading of the destination.

```
<HTML>
<HEAD>
<TITLE>Anchors</TITLE>
</HEAD>
<BODY>
<A name="Topic_1"><H2>Topic 1</H2></A>
Topic 1 stuff.
```

continues

Listing 11.2. continued

```
<A name="Topic_2"><H2>Topic 2</H2></A>
Topic 2 stuff.
<A name="Topic_3"><H2>Topic 3</H2></A>
Topic 3 stuff.
</BODY>
</HTML>
```

This method works well with sections that have their own headings, such as the document in Listing 11.2, or within blocks such as definition lists. Unlike using tags for hyperlinks, creating a destination anchor does not change the appearance of any enclosed text.

Marking anchors this way is recommended only if you're enclosing short pieces of text. Blocking entire pages is not desirable, especially because it's easy to lose track of the closing </A> tag.

Linking to Anchors Within a Document

The tags for linking to an anchor within an HTML document are identical to the tags for creating a hyperlink to an external document, with an additional piece of address information. The syntax for the hyperlink to a new location is

```
<A href=[URL]#anchorName>AnchorText</A>
```

where *URL* is an optional value indicating an external document followed immediately by a pound sign (#), called a hash, and the name of the anchor within the document. *AnchorText* is the text or other content that the user clicks to activate the hyperlink.

The first and most common method of connecting to an internal anchor is by using the hash and the anchor name without a URL. (See Listing 11.3.) This jumps to the desired location without reloading the document.

Listing 11.3. The three hyperlinks at the beginning of this document link to the three anchors within the document by referencing only the anchor names.

```
<HTML>
<HEAD>
<TITLE>Anchors</TITLE>
</HEAD>
<BODY>
Read about <A href="#Topic_1">Topic 1</A>.
Read about <A href="#Topic_2">Topic 2</A>.
Read about <A href="#Topic_3">Topic 3</A>.
<HR>
<A name="Topic_1"><H2>Topic 1</H2></A>
Topic 1 stuff.
<A name="Topic_2"><H2>Topic 2</H2></A>
Topic 2 stuff.
<A name="Topic_3"><H2>Topic 3</H2></A>
Topic 3 stuff.
</BODY>
</HTML>
```


This is a simple solution for linking to internal anchors. On pages that are developed with a CGI script, this solution doesn't always work. In that case, it becomes necessary to add the filename to the URL. (See Listing 11.4.)

Listing 11.4. The three hyperlinks at the beginning still link to the three anchors within the document, but now the entire page is reloaded at the same time.

```
<HTML>
<HEAD>
<TITLE>Anchors</TITLE>
</HEAD>
<BODY>
Read about <A href="Anchors.html#Topic_1">Topic 1</A>.
Read about <A href="Anchors.html#Topic_2">Topic 2</A>.
Read about <A href="Anchors.html#Topic_3">Topic 3</A>.
<HR>
<A name="Topic_1"><H2>Topic 1</H2></A>
Topic 1 stuff.
<A name="Topic_2"><H2>Topic 2</H2></A>
Topic 2 stuff.
<A name="Topic_3"><H2>Topic 3</H2></A>
Topic 3 stuff.
</BODY>
</HTML>
```

This method also has a drawback. Every time the user clicks one of the links, the browser reloads the page before moving to the desired anchor. For large pages, or pages with lots of graphics, this means a much slower response time. However, you can use this behavior to your advantage by linking to a specific location in an external document. (See Listings 11.5 and 11.6.)

Listing 11.5. This page includes a paragraph with a hyperlink to a definition term in a separate document.

```
<HTML>
<HEAD>
<TITLE>Fire Behavior</TITLE>
</HEAD>
<BODY>
One of the most dramatic and publicized fire behaviors is
<A href="glossary.html#backdraft">backdraft</A>. Backdraft is
 extremely rare, and many firefighters will never have
the opportunity to witness its effects firsthand.
</BODY>
</HTML>
```

Listing 11.6. This is the glossary page referred to in the preceding listing. Note the anchors defined for each definition term.

```
<HTML>
<HEAD>
<TITLE>Fire Glossary</TITLE>
```

continues

Listing 11.6. continued

```
</HEAD>
<BODY>
<DL>
<DT><A name="fire_triangle">Fire Triangle</A><DD>The minimum requirements of
combustion and the basis for all extinguishment techniques. Its sides
represent fuel, heat and oxygen.
<DT><A name="fire_tetrahedron">Fire Tetrahedron</A><DD>A four-sided figure
representing the three elements of the <A href="fire_triangle">Fire Triangle</A>
plus an extra element for the chemical process causing flames.
<DT><A name="backdraft">Backdraft</A><DD>An explosive condition created when
oxygenated air is suddenly introduced to a superheated flammable atmosphere
with deficient oxygen for combustion.
<DT><A name="incipient">Incipient</A><DD>The initial stage of a fire, when
its size is still relatively small and temperatures are low. A fire is most
easily extinguished at this phase.
<DT><A name="flashover">Flashover<DD>The point at which all the combustible
material in a room reaches its ignition temperature and ignites simultaneously.
</DL>
</BODY>
</HTML>
```

When the user clicks the term `Backdraft` in Listing 11.5, the browser loads the glossary document and navigates to the anchor named after the hash in the hyperlink URL.

Summary

Hypertext is the key to the World Wide Web's success. The use of hyperlinks to other documents turns the typical dead-end page of information into a major crossroads, with a multitude of destinations from one leaping-off point.

Anchor tags are the key to how the Web implements hypertext. As their primary usage, anchors mark hypertext links that connect with other Web nodes and resources. In its typical form, the anchor tag encases a small snippet of text and includes a single attribute, `href`, which identifies the destination.

In its other form, the anchor tag also marks a destination within a document. This is accomplished by using the `name` attribute without `href`. The result is a location that is accessed using a hyperlink anchor tag.

With these two basic capabilities, the anchor tag is the backbone of World Wide Web navigation, making jumps between pages, Web sites, and continents as easy as clicking a mouse.

CHAPTER 12

Adding Images to Your Web Page

by Rick Darnell

IN THIS CHAPTER

Once upon a time, before the World Wide Web got its name, there was a network that looked and acted like the Web, but it wasn't available to everyone. It was pretty limited in scope and appearance and was used by a bunch of guys with funny hats and whose hobbies included missile button design. These folks didn't care much for aesthetics and entertainment, so their Web didn't support images. Besides, they were still working in the days of yore when computer screens were green. Their browsers only displayed text, and there was no need or demand for presenting graphics across the network connection.

Some of the people with access to the early incarnations of the Web were at universities and other institutions of higher learning. As more and more people came into contact with the Web, it became more popular and more ideas surfaced about other uses and how it could look much nicer than it looked.

But all of the browsers in those early days were text-based. As the desire to share ideas grew, someone came up with the bright idea of including images as part of a Web page.

TIP

One of the most popular choices at the time was Lynx. It's still around today and in wide use among people who use nongraphics-based computer systems, and those who just prefer not to bother with all of the Web adornments such as images, applets, virtual reality, and so on.

Enter the National Center for Supercomputing Applications at the University of Illinois (NCSA). They put together a little program called Mosaic that supported the display of images right on the page with the rest of the text. Thus, the graphical browser was born, and this first work became the basis for virtually every other browser created since, including Netscape Navigator and Microsoft Internet Explorer.

NCSA added an additional tag called `<IMG>` to the fledgling HTML, which enabled early authors to insert a picture inline with the rest of the text. That tag is still around, and it has some brand new tricks, which I show you in this chapter. This chapter also includes some friends and relations of the tag that also relate to images. This feature has come a long way since its first inception, and you'll learn a multitude of ways to manipulate appearance and placement.

The Basic <IMG>

The `<IMG>` tag is an empty element used to insert inline images. This includes items such as small icons and graphics, in addition to large image maps that occupy most of the browser window. Because the tag is a single resource element (one tag for one image), an ending tag is not supported.

In addition to identifying which image to use, the tag also has various attributes for defining its position relative to the surrounding text and Web content. This includes floating the image in the left or right margin, or placing it on top of, below, or centered on the textline it appears on.

The syntax for an image tag is

```
<IMG src="[URL]filename" [alt="textDescription">
```

where the `src` attribute identifies the image *`filename`* through a physical or relative URL and filename. The `alt` attribute is used to define a brief text description of the image or its use for browsers that don't load the image.

TIP

Browsers don't load images for a variety of reasons. First, the browser could be a nongraphical browser such as Lynx. Second, the user might have image autoloading turned off to speed up download time. Providing a value for `alt` ensures that the user has some idea of what's supposed to be going on with that big blank space.

The attribute is vital for interoperability with speech-based and text-only user agents. For disabled persons, the `alt` value can provide a brief description of what the image is. For text-only browsers, it's the only indication that the user is missing any content.

The `src` attribute is required for every image. It identifies the specific image to use and its type. The two most popular image file types are GIF (Graphics Interchange Format) and JPEG (Joint Photographic Experts Group, also used as JPG), although PNG (Portable Network Graphic) images are starting to gain acceptance and wider usage.

The second attribute in the syntax definition, `alt`, is optional but recommended. It provides a textual description of the image and is the only portion of the tag used by browsers that don't support inline images.

Where to align the Image

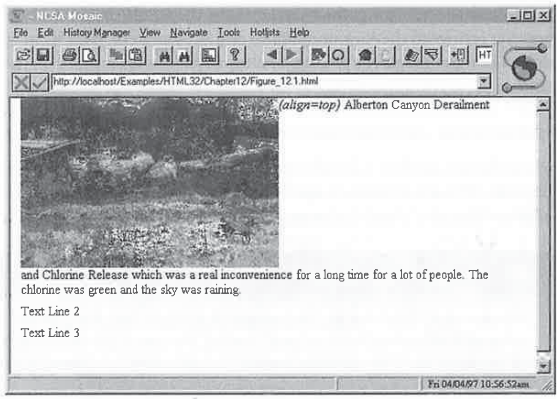
The `align` attribute controls how the image is positioned in relation to the line of text in which it occurs. Unlike other alignment attributes for items such as tables, `align` controls both the horizontal and vertical placement. Its syntax is

```
<IMG src="URL" align="position">
```

where *`URL`* includes the name of the file, and *`position`* is one of five values: `top`, `middle`, `bottom`, `left`, or `right`. The specific action of each value is as follows:

- `top` positions the top of the graphic with the top of the current line. (See Figure 12.1.) If the text line is formatted with a tag such as `<H1>`, the image will appear to occupy more space within the line itself than if it occurs within a line of standard body text.

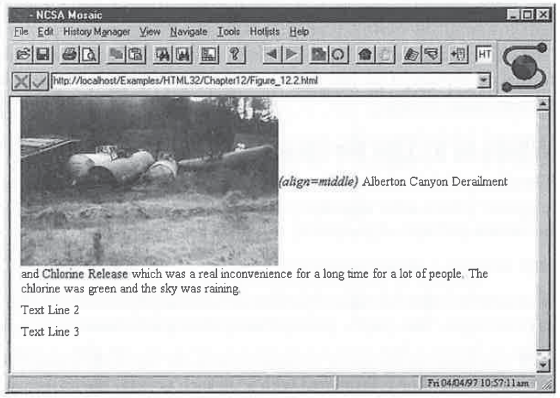
FIGURE 12.1.
Top alignment causes the top of the image to line up with the top of the current line. Note the position of the top border of the image in relation to the letters next to it.



Browsers differ concerning whether the line immediately preceding or following is used to determine alignment.

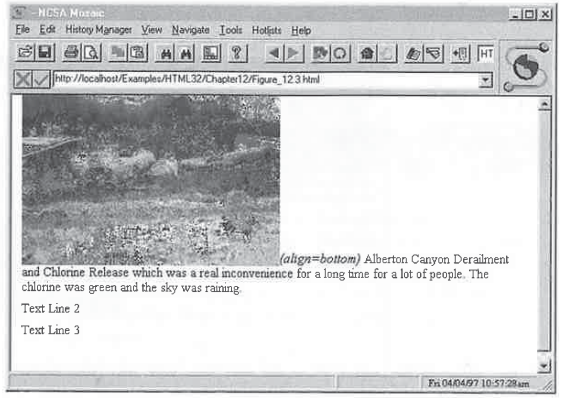
- `middle` is similar to `top`, but the vertical midline of the image is aligned with the baseline of the current line. (See Figure 12.2.) Its interpretation is consistent across browsers.

FIGURE 12.2.
Middle alignment matches the vertical halfway point of the image to the baseline of the current line of text.



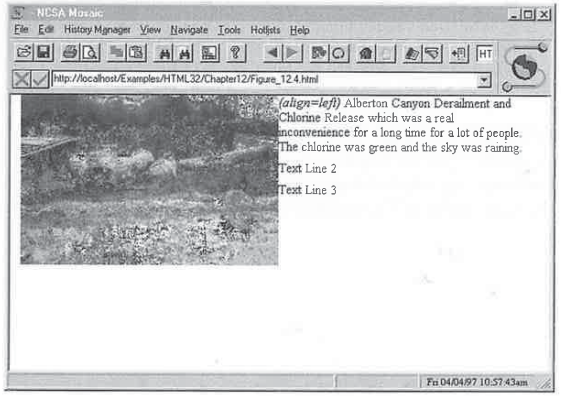
- `bottom` is the default value if `align` is omitted from the `<IMG>` tag. (See Figure 12.3.) In this case, the bottom of the image rests on the baseline for the current line.

FIGURE 12.3.
Bottom alignment is the default placement for images. The bottom of the image rests on the baseline for the current line of text.



- `left` forces the image to the current left margin, and any text following the image flows around the right margin of the image. (See Figure 12.4.) Its interpretation depends on whether any images or other material with `left` alignment appear earlier. Preceding text generally forces the image to wrap to a new line, with the subsequent text continuing on the line preceding the image.

FIGURE 12.4.
Left alignment results in the text following the `<IMG>` tag flowing around the right side of the image.



- `right` is similar to `left`, but the image is forced to the right margin. (See Figure 12.5.) Any following text is wrapped along the image's left side. It exhibits the same behavior as `left` in the opposite direction, depending on the alignment of preceding text and other material.

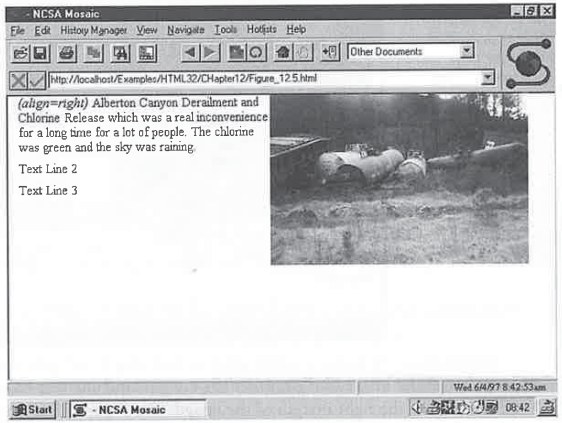


FIGURE 12.5.
Right alignment is the same as left, only the subsequent text flows around the left side of the image.

TIP

Some browsers introduce extra line spacing with multiple images using `left` or `right` alignment. Don't depend on the spacing to be uniform across all browsers and all platforms. For more information on controlling text flow, see Chapter 8, "Text Alignment and Formatting."

When placing an image on a page, remember that it is an inline feature that is displayed right along with any text on the same line. If it's not separated from surrounding material with a line break or paragraph, the image is placed on the same line as the current line of text according to the `align` attribute. This can lead to some rather undesirable appearances for images.

Making Space with `width` and `height`

The two size attributes set the desired horizontal and vertical space for the image in pixels. They are typically used as a pair to reserve space in the browser window before the image is loaded. The syntax is

```
<UMG src="URL" width=pixels height=pixels>
```

where *`URL`* is the name and *`pixels`* is the amount of space reserved for the image.

NOTE

For consistent and predictable results, images should be used at actual size; however, that's not always possible for a variety of reasons. In most browsers, the `width` and `height` attributes take precedence over the actual image size, allowing an easy way to force the image into a different-sized hole.

There are some drawbacks to this method of resizing an image. First, if a small image is enlarged, the overall image quality suffers. If a large image is reduced, download time is increased for a larger file than needed.

If the values specified in the `<IMG>` tag don't maintain the height to width ratio of the original, the image will appear distorted on the Web page.

There are a couple of reasons why you'll want to specify a size for your image. The first, already mentioned, is that it can speed display time for the rest of the page. When size information is omitted, some browsers set aside a minimal amount of space and then begin downloading the first bit of the image, which includes its size information. While the browser is working on that, it doesn't work on downloading any more of the body of the page.

Depending on the browser, the rest of the page might not display until it knows how much space each image needs. Or, the display might update and reload as each piece of information is acquired. Using the `width` and `height` attributes removes the guesswork for the browser.

The two attributes also preserve page formatting. If your page's overall appearance is dependent on the size and relation of the images to the text (like a newspaper or magazine), specifying the image size ensures that the proper amount of space is blocked. Although the image still won't appear, the right amount of space is held open to ensure the desired effect.

Would You Care for a border with That Image?

When an image appears as part of a hypertext link, the browser usually responds by drawing a colored border (usually blue) around the image. The width of this border is set using the `border` attribute. The syntax is

```
<IMG src="URL" border=pixels>
```

where *URL* is the path and name of the image file, and *pixels* is the width of the border in pixels. Use a value of `0` to hide the border. The color of the border is controlled by the link color attributes in the `<BODY>` tag. For more information, see Chapter 7, "Structural Elements and Their Usage."

TIP

Most browsers also indicate a hyperlink image by changing the mouse pointer when it passes over the graphic, so that the user doesn't have to guess whether the borderless image is a hyperlink.

Give the Image a Little hspace and vspace

The space attributes set up a *buffer zone* around the perimeter of the image. This is very useful when white space is needed immediately adjacent to the image. The `hspace` and `vspace` attributes set the width of this white space in pixels. The syntax is

```
<IMG src="URL" hspace=pixels vspace=pixels>
```

where *URL* is the image file and *pixels* is the number of pixels added to the appropriate side. By default, both are small nonzero numbers. This provides just enough white space to keep the image from touching adjacent text.

NOTE

White space is a design term meaning "space without anything in it." The space doesn't necessarily have to be white. Depending on the background color of the page, it could also be green, blue, pink, or any other color.

The space is added to both sides of the attribute. Therefore, if you include a value of `40` for `hspace`, 40 pixels of space will be added to the right and left sides of the image. This will make the image appear not in alignment with the other margins on the page.

ismap for Server-Side Maps

The `ismap` attribute identifies the image as an image map. Image maps enable you to associate specific areas of an image with hyperlinks to other documents.

In order to have validity, an `<IMG>` element with `ismap` is encased with a hypertext anchor tag. When the user clicks the image, the `ismap` attribute passes the x and y location of the click to the server. The syntax is

```
<a href="URL/file.map"><img src="imageURL" ismap></a>
```

where *URL/file.map* is the name of the MAP file with the coordinate information and *imageURL* is the name of the actual image. This is called a *server-side* image map because all of the hyperlink information is processed by the Web server.

TIP

The `ismap` attribute is used only with server-side image maps. In order to function, the server must have the appropriate CGI software to process the image map information and the file that defines each of the hot spots.

The location on the image where the user clicks is passed to the server by creating a new URL from the URL specified in the anchor tag. A question mark is added to the end, followed by the value of the x and y coordinates separated by a comma. The link is then followed using the new URL. For example, if the user clicks the location `x=10` and `y=27`, the URL sent to the server is `/cgibin/navbar.map?10,27`.

TIP

It is a good idea to use `border=0` with the image and use graphical clues to let the user know that the image is a clickable map. Otherwise, it's easy to confuse an image map with a one-shot hyperlinked image. Graphical clues include techniques such as bold shapes with descriptive text, or a long horizontal shape in the traditional navigation bar layout.

A recent feature added to browsers is the capability to interpret an image map without any help from the server. These are called client-side image maps, and they include all the necessary information to process their hyperlinks without further communication from the server. This type of image map is covered with the next attribute—`usemap`. More information on image maps is provided in Chapter 14, "Creating Image Maps."

usemap for Client-Side Image Maps

The usemap attribute is used to mark an image as a client-side image map that is used with a <MAP> element. In most cases, a client-side map is preferable to a server-side map because it requires no communication across network lines and no additional support from the server to operate. The syntax is

```
<IMG src="URL" usemap="[mapURL.html]#mapName">
```

where *URL* is the image file and *[mapURL.html]#mapName* is an incomplete URL specifying the location of the <MAP> element that has the coordinate and hyperlink information, and is identified similar to a target anchor. If the filename is omitted, the <MAP> element is assumed to be in the current HTML file.

A MAP FOR A FILE

Note the difference between the extension of the file specified for the hyperlink anchor on ismap and the source of the coordinates for usemap. The former requires a MAP file, which consists of a collection of coordinates and hyperlinks. The latter is a block of HTML tags within an HTML file. The file can contain other content, or it can exist solely for containing the map information.

The various active regions of the image map are described for the browser using <MAP> and <AREA> tags. This information is usually placed inside the HTML file with the <IMG> tag that requires it, although it can be placed in a separate file if the same map is used on several pages.

To ensure compatibility across browsers and platforms, ismap and usemap can be used on the same image. This allows the browser to choose which way it wants to interpret the image map, giving the document the maximum chance for compatibility.

Background Images

Although they are not directly related to the <IMG> tag, this seems like a good place to mention the use of background images for Web pages. These images are placed on a page through an attribute of the <BODY> tag. The syntax is

```
<BODY background="[URL]filename">
```

where *URL* is the location of the image specified with a complete URL including server name and path or a relative URL with or without a path, and *filename* is the name of the image to use. For more information on relative pathnames, see Chapter 11, "Linking Documents and Images."

TIP

For compatibility, limit the file type to GIF or JPEG. Image files in these formats are supported by virtually every graphical browser in use.

The image is placed in the background of the Web page, similar to hanging wallpaper in a bedroom. If the image is smaller than the space occupied by the browser window, it's tiled to fill the entire background. Use care to ensure that the image isn't so bold or distracting as to make the rest of the page unreadable. For more information on using background and other color attributes of <BODY>, see Chapter 7.

Using Images as Substitute Content

With the wide variety of browsers and capabilities on the Web, it's increasingly hard to make one page work for everyone. This is especially true when including content such as Java applets, ActiveX controls, and Netscape plug-ins. If a browser doesn't recognize your special content, it ignores it and either leaves a big hole or a blank space in your layout.

You can use browser tag incompatibility to your advantage by placing an image tag just before the closing tag of the specialized content. The format is

```
<APPLET attributes height=contentHeight width=contentWidth>
<PARAM attributes>
…other applet-specific lines…
<IMG src="URL" height=imageHeight width=imageWidth>
</APPLET>
```

where the applet tag is any of the specialized content container tags and the parameter tags are any of the tags subordinate to the container. As a matter of style and convention, the <IMG> tag is placed immediately preceding the closing container tag, and the content and image size attributes have the same value.

When this set of tags is encountered by a browser that doesn't support applets, the following is what happens:

- The opening container tag is not understood, so the browser ignores it and does nothing.
- Likewise, the subsequent tags that support the container tag are ignored. So far, the browser has done nothing but throw away the material that it doesn't understand.
- When the <IMG> tag is encountered, the browser suddenly has something it can work with. It interprets this tag and loads the image into the document in the space that was originally designed for the specialized content.
- The closing container tag is reached. Because it closes something the browser never understood, it is ignored, too.

This technique ensures that everyone who views your page has something to see, even if it's a large graphic that says, "You need a Java-compatible browser for this page." Another common usage is to use a screen capture from a video or VRML for the image. Even though the image is static, the user still gets a taste of what was intended.

Summary

Use of inline images in Web pages is part of what made the World Wide Web such a popular place to spend time. Use of the image tag depends on only one attribute, `src`, to identify the graphics file. Although plug-ins and different browsers support a wide variety of image formats, GIF and JPEG are the most popular and are supported in virtually every setting.

With the advent of HTML 3.2, additional capabilities within the `<IMG>` tag are officially extended to make it easier to control the appearance and behavior of graphics on the Web page. Using alignment and spacing attributes enables you to left- or right-justify an image while causing the surrounding text to wrap around the margins.

Mapping attributes set the graphics as an image map. Graphical image maps are one of the more popular uses for graphics files, including menu bars and full-screen images for site navigation.

Additional attributes of the `<IMG>` tag are covered in the next chapter, including those used to insert virtual reality worlds and aviation clips. Read on for more information on making your graphical Web page really graphical.

CHAPTER 13

Integrating Multimedia and Other File Types

by Rick Darnell

IN THIS CHAPTER